

A 3D rendered scene in a blocky, pixelated style reminiscent of the game Minecraft. The scene is set on a green, grassy terrain with brown dirt blocks. In the foreground, a player character with a blue shirt and purple pants stands on a grassy ledge. To the left, a pink pig-like creature is visible. In the background, a white and red building is partially visible. In the lower half of the image, several hostile mobs are shown: a green Creeper, a brown Zombie, a brown Skeleton, and a brown Spider. The overall lighting is soft, suggesting a dusk or dawn setting.

Improving embodied LLM agents' capabilities through collaboration

Baptiste Colle

Improving embodied LLM agents' capabilities through collaboration

by

Baptiste Colle

Student Name	Student Number
Baptiste Colle	5056756

Daily supervisor: C. Raman
Thesis advisor: M. Reinders
Project Duration: September 2023 - June 2024
Faculty: Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS), Delft

Cover: Minecraft wallpaper by Mojang under CC BY-NC 2.0 (Modified)
Style: TU Delft Report Style, with modifications by Daan Zwaneveld

Acronyms

AI – Artificial Intelligence
BDI – Belief-Desire-Intention
DAG – Directed Acyclic Graph
FN – False Negative
FP – False Positive
LLM – Large Language Model
LoRA – Low-Rank Adaptation
NLP – Natural Language Processing
NN – Neural Network
QA – Question Answering
RAG – Retrieval-Augmented Generation
RL – Reinforcement Learning
TN – True Negative
ToM – Theory of Mind
TP – True Positive

Contents

1	Abstract	1
2	Introduction	2
3	Related work	5
3.1	Minecraft as a learning environment	5
3.2	Large language models	5
3.3	LLM-based agents	6
4	Methodology	10
4.1	Design goals and Environment Updates	11
4.2	Collaborative Voyager Agent Architecture	11
4.3	Difference between Voyager and our Collaborative Voyager	16
5	Experiments	19
5.1	Experiment Setup	19
5.2	Experiment 0: Fine-Tuning as a Solution to the Performance Problem of Open-Source LLMs Instead of Collaboration	23
5.3	Experiment 1: Collaboration Improves Weak Agent Performance	24
5.4	Experiment 2: Learning from Mistakes - Post-Collaboration In-Distribution Task	26
5.5	Experiment 3: Post Collaboration Out-of-Distribution Task	26
5.6	Summary	28
6	Discussion	29
6.1	Insights learned about the field of LLM-agents	29
6.2	Key Findings	30
6.3	Limitation of our research	32
6.4	Future work	32
7	Conclusion	34
	References	35
A	Logbook of agent failures	39
A.1	Examples of failures:	40
A.1.1	Lack of instruction following / internal consistency	40
A.1.2	Hallucinate answers to QA	41
A.1.3	Example of hallucinated success	41
A.1.4	Comparison between open-source LLM and GPT	41
B	Agent prompts	43
C	Agents execution time	44

1

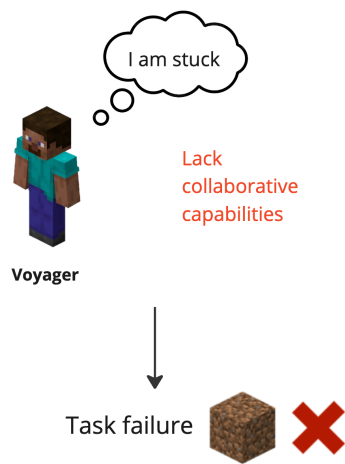
Abstract

The emergence of Language Language Models (LLMs)-based agents represents a significant advancement in artificial intelligence (AI), offering new possibilities for complex problem-solving and interaction within a virtual environment. Our work is based on the Voyager paper [1], which is a state-of-the-art LLM-based agent for Minecraft. However, this system suffers from some significant limitations, such as its reliance on closed-source LLMs and lack of social awareness. Indeed, current open-source LLMs often fail to match closed-source ones in the agent setting, leaving research reliant on third-party closed-source technology [1] [2]. This gap highlights the need for alternative strategies to enhance LLM performance without the high costs associated with fine-tuning. To address these challenges, we propose the Collaborative Voyager, a new architecture designed to enable agent collaboration and social awareness using open-source LLMs. Inspired by the social intelligence hypothesis, which suggests that intelligence emerges from social interactions, we propose collaboration as an alternative learning paradigm for LLMs. This alternative learning paradigm could potentially supplement the expensive fine-tuning currently needed to bridge the performance gap between open-source and closed-source models in the agent setting [2]. Our approach involves developing a framework that allows agents to communicate, understand, and learn from each other, enabling them to correct errors and adapt to new tasks dynamically. By using a memory module, our agent is able to remember interactions and learn from them in order to accomplish a task that it was previously unable to do on its own. Through various experiments, we demonstrate that collaboration significantly enhances the performance of LLM agents in both task completion and adaptability, addressing issues like hallucinations. This study provides insights into developing more sophisticated, adaptable AI systems capable of dynamic interactions and problem-solving. These findings have potential applications extending beyond Minecraft and virtual environments to fields such as robotics, where collaboration and social awareness are crucial.

2

Introduction

Voyager (previous method)



Collaborative Voyager (our method)

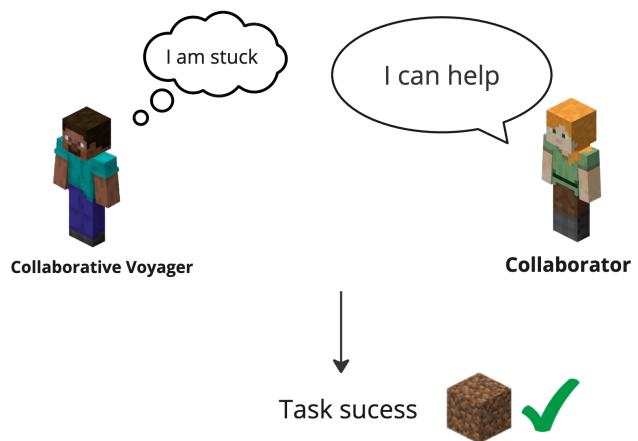


Figure 1: Method overview

The emergence of multi-agent systems and language learning models (LLMs) represents a significant advancement in artificial intelligence, offering new possibilities for complex problem-solving and interaction within virtual environments. Minecraft, a large open-world video game, is an ideal platform for studying these phenomena due to its complex, dynamic nature and requirement for strategic planning and execution. Historically, integrating LLMs into gaming environments like Minecraft has paved the way for innovative research in AI, exploring the boundaries of agent collaboration and autonomous problem-solving.

Problem Statement Current research in LLM-based agents often relies on closed-source LLM, such as GPT4. This makes the development of those agents costly and limited. We believe this makes current research exposed to a dependence on third-party private entities. Research has been done to close the gap with the closed-source model, but this is mainly done on text assistant tasks instead of the agent setting, where an LLM is expected to take some actions in a world. The agent setting is more difficult for LLMs to perform as it is outside their training data and is then out-of-distribution as they are not trained to be an action agent.

These problems are especially pronounced in specialized contexts such as Minecraft. The gap between the capabilities of open-source LLMs and their proprietary counterparts, such as GPT-4, further exacerbates this challenge, revealing a critical area for improvement. Indeed, current open-source LLM often struggles in an embodied environment compared to their closed-source counterpart [1] [2].

During our research, we discovered that open-source LLM cannot accomplish even the simplest task in Minecraft. Wang et al. (2003) [2] propose fine-tuning as a potential way to bridge the performance gap between open and closed-source LLM. However, this remains costly and requires a lot of data that may not be freely accessible.

Research Objectives and Questions At the heart of this research is the social intelligence hypothesis, which posits that intelligence emerges through social interaction within embodied settings [3]. Therefore, we believe that LLM could also learn from interaction and collaboration with others. This hypothesis aligns with the theoretical underpinnings of cognitive development and agent-based modeling, providing a solid foundation for investigating collaborative learning and adaptation in AI agents. Another problem with LLMs is still their limited collaboration capability and social intelligence. Research has shown that by using perspective taking, LLM can understand social cues [4] [5]. Furthermore, there have historically been theories of cognitive development that state that intelligence emerges through embodiment interacting with a social setting Vygotsky [6].

This study seeks to explore how collaborative interactions, grounded in social intelligence and perspective-taking, can enhance the capabilities of AI agents beyond what is achievable through traditional fine-tuning methods. By advancing our understanding of multi-agent collaboration, this research aims to contribute to the development of AI systems that are not only more effective in specific tasks but are also socially aware.

This research aims to bridge the knowledge gap concerning AI agent collaboration in Minecraft, focusing on enhancing agent capabilities through perspective-taking, especially in the open-source LLM domain. The primary research question that guides this investigation is: *How can collaboration improve the problem-solving abilities of LLM-powered agents?*

Hypothesis Driving the Research The research is driven by several hypotheses aimed at exploring different facets of agent performance and collaboration:

- **Collaboration Hypothesis:** Direct collaborator interaction and guidance will significantly correct and enhance the problem-solving capabilities of AI agents.
- **Adaptive Learning Hypothesis:** AI agents can learn from collaborator interventions and adapt to new tasks more effectively, indicating a form of learning and behavioral adaptation.

To investigate our research question and hypothesis, we developed a new method based on the Voyager paper [1], which is a state-of-the-art LLM-based agent for Minecraft, however, it suffers from severe limitation. Indeed, as we explore in our experiment chapter 5, Voyager cannot function without a closed-source LLM, even on the simplest tasks. Furthermore, it cannot collaborate or backtrack from a wrong reasoning. Therefore, the agent's success is left to the quality of the underlying LLM, with no way of learning from others. This limits the final performance of the agent to the underlying LLM model, making open-source LLM fail in the Minecraft domain. Our work, therefore, extends the Voyager agent with social awareness in order to allow open-source LLM to compete with closed-source models.

In this work, we introduce our own socially aware LLM-based agent for Minecraft, the Collaborative Voyager. The behavioral difference between our new agent, Collaborative Voyager, and the original Voyager can be seen in Figure 1. Indeed, the original Voyager is limited by its internal capability, but our agents can dynamically learn from others to achieve tasks that it was unable to do on its own.

By exploring alternative training methods such as collaboration, this study aims to contribute to developing more sophisticated, adaptable AI agents capable of complex interactions and problem-solving in dynamic environments. Relying on collaboration as a training paradigm instead of fine-tuning, which is data and compute intensive, should reduce the barrier of entry to training LLM agents. We want to determine how collaboration can improve the problem-solving abilities of LLM-powered agents within Minecraft. This involves developing a framework that enables AI agents to dynamically understand and integrate collaborator feedback, thus improving their performance in complex tasks. Our primary objective is to investigate how the insights gained during social interaction could correct and improve the performance of AI agents in Minecraft over time.

The knowledge gained from exploring AI collaboration and adaptive learning in virtual environments can inform the development of AI systems in areas such as robotics, where understanding social behavior is crucial. The research employs a mixed-methods approach, incorporating quantitative experiments to assess the impact of collaboration on agent performance and qualitative analysis to explore the dynamics of this collaboration. Experiments are designed to test hypotheses related to context adaptation, social intelligence, and perspective-taking among AI agents in Minecraft.

The introduction outlines the research background, problem statement, objectives, and theoretical framework, setting the stage for the inquiry. The related work chapter provides a comprehensive overview of existing research on multi-agent systems, LLMs agents, and their application in embodied environments, establishing the context for the study. Our methodology chapter details the experimental design and methodologies employed to investigate the research questions. Our experiments chapter presents the findings from the experiments conducted, analyzing the impact of collaboration on agent performance in Minecraft. The discussion chapter interprets the results, discusses their implications for AI research, agent collaboration, and the broader field of AI technology, and outlines future research directions. The thesis concludes with a summary of the key findings and encapsulates the research journey.

This research has significant practical implications beyond the realm of Minecraft. Improvements in multi-agent collaboration within this game can lead to advancements in cooperative problem-solving. Understanding the dynamics of agent collaboration could lead to the development of versatile social agents capable of identifying and complementing the weaknesses of their collaborators, thus enhancing overall efficiency and effectiveness. The expected outcome of this research is a deeper understanding of the dynamics involved in collaboration. Insights gained from this study could pave the way for developing multipurpose socially aware agents adept at recognizing and augmenting each other's weaknesses.

3

Related work

3.1. Minecraft as a learning environment

Minecraft, with its intricate and dynamic environment, presents a perfect sandbox for AI agents. Its nonstationary nature, coupled with high-dimensional and open-ended landscapes, offers a realistic and challenging setting for AI development. Unlike environments with predefined reward functions, Minecraft is a large open-world environment with no predefined precise goal. Due to its partial observability nature, collaboration in Minecraft is particularly difficult. A notable aspect of Minecraft is the complex dependency trees required to craft items, which introduces long-term dependency challenges that are often unmet by current AI models. For example, the success rate in crafting a diamond pickaxe, a task with multiple substeps and dependencies, is significantly lower in AI agents compared to human players [2]. This gap highlights the limitations of current models in managing long-term dependencies and complex skill trees.

Minecraft has been a long-standard benchmark in agent development in the RL community. Malmö [7], a yearly competition in Minecraft, focuses on specific tasks in Minecraft instead of a general-purpose agent benchmark. Minecraft is a multiplayer game; therefore, collaboration is an integral component of the gameplay experience. Malmö, for example, releases particular challenges in which collaboration between different agents is evaluated. Narayan-Chen et al. (2019) [8] created a dataset to explore the dynamic of creating a 3D structure with multiple participants. However, current research on collaboration in Minecraft is still limited [9]. This enables Minecraft to be a suitable sandbox for evaluating general reasoning and collaboration of agents.

In addition, Minecraft's considerable player base enables the gathering of a large amount of data for training [10] [11]. This provides rich resources for AI training, further emphasizing its suitability as an AI development platform.

RL-based Agents in Minecraft The Malmö system has served as a benchmark for reinforcement learning (RL) agents. These agents typically focus on small, specific tasks. However, their effectiveness diminishes in Minecraft's open world, where more complex tasks are encountered. This limitation is primarily due to the vast and varied environment that Minecraft offers, where traditional RL methods struggle to scale and manage long-term dependencies.

To address these challenges, current methods have shifted towards imitation learning, using large-scale pretraining on human demonstrations, as highlighted in VPT [12] and further explored in the Minedojo project [10]. Miniclip from Minedojo trains a model based on videos to act as a reward function, allowing the development of an RL-based system on top of such methods. These methods allow the bootstrapping of RL systems to work in Minecraft's open world.

3.2. Large language models

Large Language Models (LLMs) have emerged as a pivotal breakthrough in the field of artificial intelligence, offering unprecedented capabilities in natural language processing (NLP) generation. They are based on the transformer architecture introduced by [13], which enables these models to handle

long-range dependencies in text. LLMs are pre-trained on a vast corpus of text data, allowing them to generate coherent, contextually relevant text across a wide range of tasks without task-specific training [14].

Prompting Techniques Due to the rise of LLM, prompt engineering has emerged as a new subfield of NLP. Comparisons between foundational LLMs and domain-specific LLMs, like in the medical domain [15], have shown that the proper prompting techniques can enable foundational models to outperform domain-specific ones. This is achieved by unlocking and effectively utilizing the extensive information stored in the foundational models' weights. This approach demonstrates that, with strategic prompting, foundational LLMs can be adapted to a wide range of tasks and environments, rivaling or even surpassing the performance of more specialized models [16].

To further improve the performance of LLMs, advanced prompting techniques are used. Techniques such as self-reflection [17], chain of thought [18], or tree of thoughts prompting [19] allow LLMs to critically evaluate and refine their responses. This introspection is similar to human cognitive processes of reflection and reconsideration, particularly valuable in scenarios where initial responses may require further development or adjustment. These advanced prompting techniques improve the decision-making process of LLM and, therefore, their performance on specific tasks.

Adding Knowledge to an LLM Retrieval-Augmented Generation (RAG) [20] is a prompting technique that combines the powers of retrieval-based and AI models to allow language generation to be grounded in facts. This methodology works by first retrieving relevant information from an extensive database or corpus and then feeding this information into a AI model to produce more informed and accurate outputs. The retrieval component allows the system to access a vast array of information on a wide variety of topics, enhancing the model's ability to generate responses that are both relevant and factually accurate.

Fine-tuning refers to the process of adjusting the parameters of a pre-trained model on a new, typically smaller, dataset to tailor the model to specific tasks or improve its performance on particular domains. This approach leverages the general knowledge learned by the model during its initial training on large datasets and applies it to more specialized tasks. By adjusting the model to focus on the features most relevant to the specific tasks, fine-tuning helps to achieve higher accuracy and performance without the need to train a model from scratch [21]. LoRA (Low-Rank Adaptation) [22] is a fine-tuning method that uses a low-rank matrix to update the weights of a model. This drastically reduces the number of trainable parameters, making the fine-tuning process cheaper and faster.

There is still an open debate on which technique, RAG or fine-tuning, is most relevant for a given situation. This is why we used both in our work to explore the difference between them better [23].

Social Reasoning in LLM Theory of Mind (TOM), the capability to attribute mental states to others, is also a significant consideration in LLM systems. By understanding and predicting the intentions, beliefs, and emotions of other agents, an LLM can better navigate interactions and collaborate. This understanding is crucial for effective communication and joint problem-solving. BigToM [5] proposes a method to evaluate the performance of LLM on ToM tasks. Wilf et al. (2023) [4] further show that perspective-taking improves the capability of LLM on ToM tasks. Indeed, understanding and developing these aspects in AI agents could lead to even more nuanced and sophisticated interactions, both among AI agents themselves and between AI agents and human collaborators.

The concept of majority voting is another effective method in multi-agent LLM systems [24]. Here, different agents offer different perspectives on the same problem, and a consensus is reached through a voting mechanism. This approach ensures that the final decision or solution is well-rounded and considers multiple facets of the problem at hand.

3.3. LLM-based agents

LLM-based agents utilize an LLM to solve complex tasks. The LLM acts in the world by integrating with different external components, often through APIs. In constructing LLM agents, the LLM acts as the controller managing the sequence of operations required to fulfill tasks or respond to user requests. LLM agents can generally be broken down into a profile, memory, planning, and action module [26]. The profile determines the system's behavior, setting a persona for the LLM agents and providing context

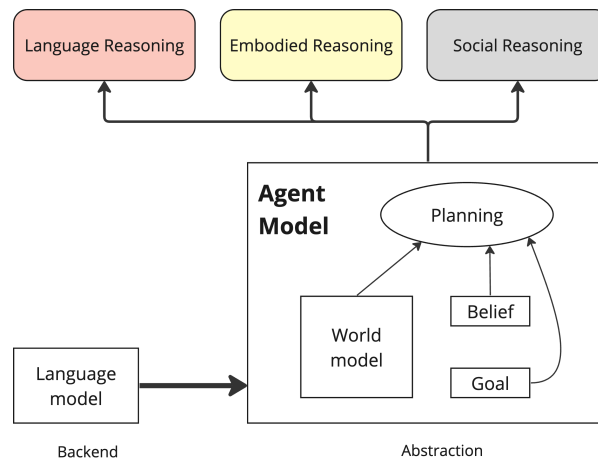


Figure 2: LLM agents architecture recreated from NeurIPS tutorial [25]

for its responses. Memory enables the agents to retain relevant experiences, often coupled with a retrieval system to enhance decision-making. Based on this knowledge, the LLM formulates a course of action, and its textual outputs are then translated into actions. Figure 2 from Hu et Shu (2023) shows an example of a breakdown of LLM agents. As this field is quite new, there are multiple representations and definitions of the components of an LLM-based agent. Hu et Shu (2023) focuses on the intersection between world models and LLM-adjacent [27]. This thesis investigates the social reasoning capability of LLM-based agents in an embodied world (Minecraft), which is still an under-researched area.

Embodiment Embodiment theory suggests that the physical form of an AI system plays a critical role in its ability to learn and adapt to its environment. According to Pfeifer et al. (2007) [28], the physical body of a robot can influence its cognitive processes, arguing that intelligence emerges in the interaction between a brain, a body, and the environment. This perspective highlights the importance of sensorimotor experiences in the development of cognitive abilities, suggesting that embodied AI could lead to more adaptive and context-aware systems.

Furthermore, the concept of embodiment is closely linked to the idea of situated cognition, which posits that cognition is fundamentally influenced by the physical and social context in which it occurs. Suchman (1987) [29] emphasizes the role of the environment in shaping the actions and decisions of intelligent agents, suggesting that AI systems need to be embedded in their operational contexts to achieve higher levels of intelligence and autonomy.

Continual Learning Continual learning is a crucial aspect of artificial intelligence that enables models to learn continuously, adapting to new information without forgetting previous knowledge. This process contrasts with traditional machine learning models, which are typically trained on a fixed dataset and can struggle with new or evolving data. Continual learning for LLM remains an open challenge [30] [31].

A key challenge in continual learning is catastrophic forgetting, where the model forgets previously learned information upon learning new data. Various strategies have been developed to mitigate this, such as Synthetic Experience Replay [32], which generates synthetic data mimicking past experiences to maintain model performance. This technique allows the model to rehearse old tasks while learning new ones, thus preserving its ability to perform previously learned tasks.

Another innovative approach is Reflexion, which integrates verbal reinforcement learning into language agents [17]. In Reflexion, the language model integrates verbal feedback, enabling it to refine its subsequent answers. This method enhances the model's adaptability and decision-making capabilities by allowing it to learn from its mistakes and successes continuously. This also mimics human learning, where verbal feedback is an important part of the learning process.

LLM-based Agents in Minecraft We will now focus on LLM agents in the videogame domain, namely Minecraft. As there is a discrepancy between LLM textual output and the game's expectation of

keyboard-based input, the current LLM-based systems need to bridge this gap. One approach is to use code to interact with the Minecraft world. This approach was explored by Voyager [1] and Jarvis [2]. The LLM agent is asked directly to produce code that can then be used to control the agent in the game. An alternative is GITM [33], which uses a neural network controller to map the text output from LLM to keyboard and mouse movements.

Another approach for agents in Minecraft is to extend current RL systems with insights from LLM. For example, Plan4MC [34] uses an LLM to construct a skill graph of the different actions learned from RL exploration. Deckard [35] uses an LLM to create an abstract world model (DAG), which is then verified with traditional RL techniques.

Multi LLM Based-Agents System The social intelligence hypothesis [3] postulates that collaboration and social dynamics have contributed to the development of human intelligence. This theory suggests that the complexities of social life, rather than environmental challenges alone, have been the main driving force behind the evolution of higher intelligence.

We have seen similar dynamics with LLM agents. By introducing more agents, multi-LLM agents' systems are often able to outperform their individual counterparts [36] [37]. MetaGPT [36] explores the idea of using different personas in role-playing to achieve better LLM performance compared to single-agent systems. By simulating different viewpoints, these agents can collectively analyze a problem more holistically, leading to more robust solutions. Adopting multiple personas allows an LLM to view a problem from different perspectives, mimicking the human ability to consider various viewpoints. This often leads to more effective problem-solving. In the realm of conversational AI, extending LLMs to interact with multiple participants poses unique challenges, namely when to talk. Wei et al. (2023) [38] delve into this, focusing on research into turn-taking dynamics and determining when agents should interject or respond in a multiparty conversation.

Social Reasoning in LLM Agents We discussed previously social reasoning in LLM. However, there is currently a large gap in the agent settings for socially aware LLM. Most work focuses on ToM capability in text settings, as described previously. There is some work that focuses on agents cooperating while being socially aware, but this is still done in a text setting [24]. Furthermore, we found few works investigating LLM in a collaborative embodied setting [39]. We believe there is still a lot to be learned about social reasoning in LLM agents and that embodiment is critical to test more advanced forms of collaboration. Our thesis focuses on this, which we believe is yet still an under-explored part of the current literature.

The Voyager architecture Our work is primarily based on the Voyager paper [1] and is an extension of it. Voyager is an LLM-based Minecraft agent that uses GPT-4 to create code that interacts with the world. We chose it because, at the start of the thesis, it was the state-of-the-art agent for Minecraft. Today, it remains one of the best-performing architectures for LLM-based agents in Minecraft.

Figure 3 illustrates the architecture of the Voyager agent. The agents use an actor-critic method reminiscent of the RL method to act in the world. At each time step, the agent will generate a code to act in the world. The subsequent world state information is then fed to a critic module that will create a critique on how the code generated could be improved if necessary until the task succeed. The actor prompt is also extended with skills and a self-question-answer module that allows the code generation to be more grounded and accurate. The self-question answering module is similar to a chain of thoughts, where instead of directly generating the code, the agent will first generate QAs (question-answer) about Minecraft. An example of such QA is "Where do you find dirt in Minecraft? You can find dirt in the following biomes: ...". Those QAs are then used to inform code generation. The skills module refers to the Minecraft Crafting Skills Tree. The skill library is a repository that stores executable code snippets representing specific actions or sequences of actions. These skills are designed to be reusable and combinable, allowing the actor to construct complex behaviors from simpler ones. The skill library grows incrementally as the agent explores and learns, enabling it to handle increasingly sophisticated tasks over time. This component is crucial to mitigating the effects of catastrophic forgetting, a common challenge in continual learning systems.

The iterative prompting mechanism proposed by Voyager allows the agent to refine its action plans through a feedback loop that incorporates environment feedback, execution errors, and self-verification. This mechanism allows Voyager to adjust its strategies based on direct interaction with the Minecraft

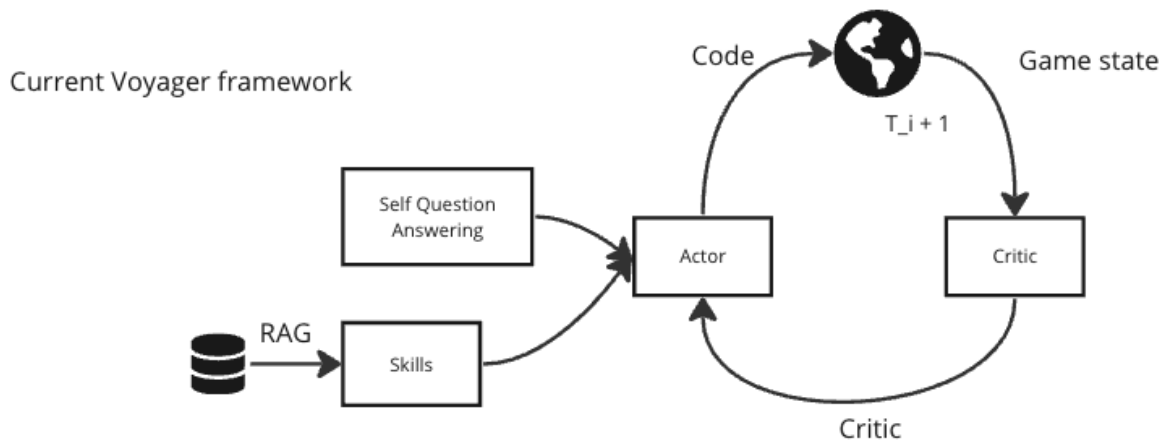


Figure 3: Voyager architecture

environment, learning from successes and failures alike. The iterative nature of this process ensures that the agent can overcome obstacles and improve its performance incrementally, leading to lifelong learning.

The Voyager does suffer from some notable limitations. We believe the most important shortcomings of the agent are:

- Hallucinations
- Reliance on closed-source LLMs
- Lack of world representation
- Not socially aware
 - Lack the ability to receive communication
 - Cannot learn from interaction

Voyager generates incorrect action codes or proposes unachievable tasks (e.g., crafting non-existent items in Minecraft). Those are LLM “hallucinations.” Some examples of those failure cases can be seen in our Appendix A, where we analyzed the most common error in the Voyager agent’s reasoning.

The Voyager architecture relies on closed-source LLM to power its agent. Indeed, as stated by the author of the paper, “VOYAGER requires the quantum leap in code generation quality from GPT-4, which open-source LLMs cannot provide”. This makes the cost of Voyager high and makes the system reliable on a third-party proprietary technology that is beyond our control. We are trying with this thesis to close the gap between closed-source and open-source LLM. Right now, open-source LLMs are too weak to power the Voyager framework, as we will later quantify in our experiment chapter 5.

The current Voyager framework lacks any type of world model and only relies on skills to generate deeper knowledge about Minecraft. We believe that by introducing a world model we can allow the agents to better understand its environment and its beliefs about it.

Lastly, the Voyager agent works in isolation; it is not socially aware. It cannot receive a message from another entity or integrate knowledge from a collaborator. Inspired by the social intelligence hypothesis, our work will investigate whether collaboration can improve a weak LLM (open-source), closing the gap with its proprietary counterparts (closed-source LLM).

4

Methodology

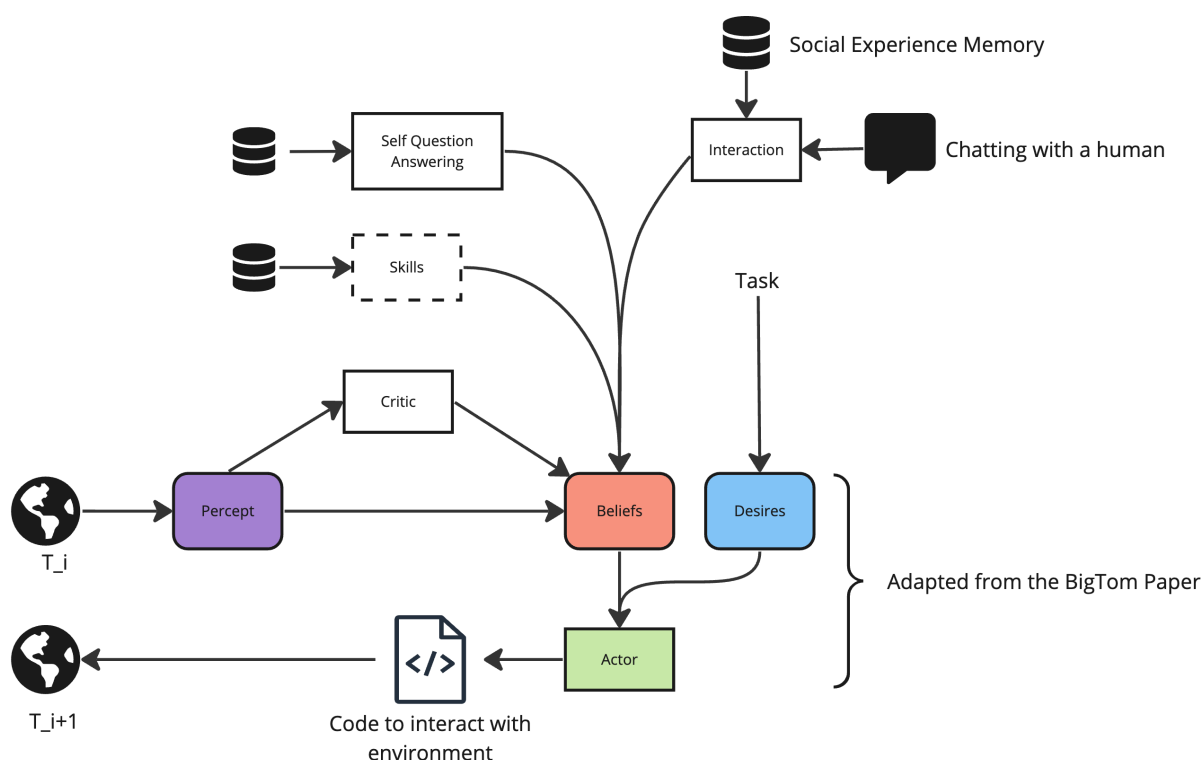


Figure 4: Architecture of our Collaborative Voyager agent

Collaborative Voyager

In this chapter, we will discuss the core of our method. We will describe the architecture of our proposed collaborative agent as seen in Figure 4. This chapter will explain step by step the composition of this architecture and the design choices that we made. The different parts of the architecture will be explained individually, and we will finally explain in section 4.3 how they all fit together to create a socially aware Minecraft agent inspired by Voyager. Those changes were made in order to enable us to study our research question and, more generally, the subject of agent collaboration.

Selection of Open-Source LLMs Our initial step was to identify a suitable open-source LLM that could potentially replace GPT-4 for our purposes. Given our initial lack of benchmark, this involved

empirical testing of several models. Through this process, we identified Open-Orca [40] as the most promising candidate, based on its performance on the open LLM leaderboard from Hugging Face and its compatibility with system prompts. This is a fine-tuned model of Mistral 7B [41]. Furthermore, Mistral-based models offer a longer context window and lower parameter count, which is crucial for real-time inference, especially in a setting such as Minecraft.

4.1. Design goals and Environment Updates

We identified four objectives that would make the Voyager agent more socially aware:

1. Ability to receive messages
2. Ability to represent and internalize those communications
 - Ground the LLM reasoning in the theory of mind framework to allow perspective-taking.
3. Ability to interpret its communication
4. Ability to learn from interactions

Our method aims to offer elements that allow the Voyager framework to be extended so that it can now collaborate and be socially aware.

Multi-agent Sandbox The Voyager paper did not provide any way of creating concurrent agents in Minecraft. As our research initially aimed to investigate agent collaboration, we needed to modify the framework to enable the creation of an arbitrary amount of agents. We made extensive modifications to the Voyager sandbox to allow multi-agent interactions.

The step was not trivial, as the whole codebase revolved around a NodeJS server that could only control a single agent. We refactored the code to allow the server to create and manage multiple agents in parallel. We modified the framework to track the state of multiple agents and allow the agents access to the state of the other agents, such as which task they are currently doing.

Now that we have a sandbox that allows the creation and management of multiple agents, we need to make our agents capable of collaboration.

4.2. Collaborative Voyager Agent Architecture

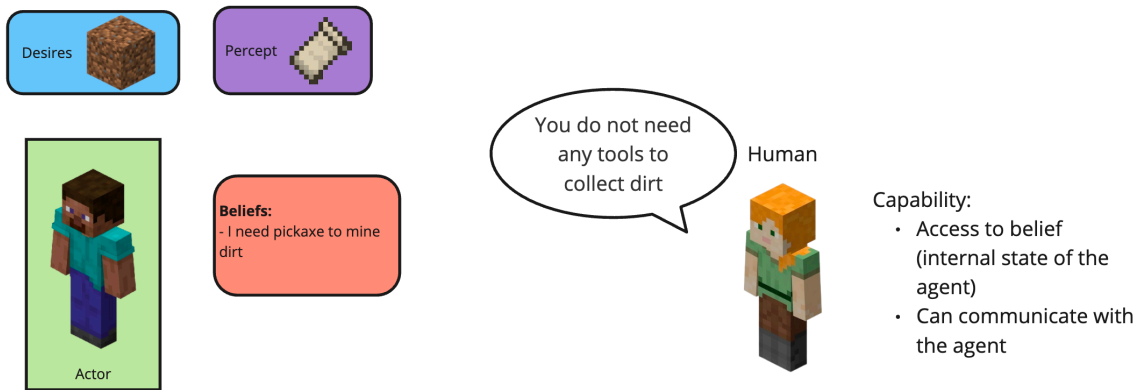
Our research develops a collaboration framework on top of the Voyager agent, aiming to demonstrate that augmenting current LLM systems with collaboration can significantly improve performance. This framework involves several key components:

- Chat Module
- Interaction LLM
- World and Partner Perspectives Representation
- Distilled Memory and Snapshots

Chat Module To collaborate, our agent needs the “ability to receive messages” as outlined in Section 4.1. Minecraft contains an in-game chat. We decided to rely on it to allow agents to collaborate. This mimics human players who also use the Minecraft chat function to communicate and organize themselves. The original Voyager does not have this capability and is only a solo agent. Our Collaborative Voyager thus needs to be able to communicate with its different collaborators.

We expanded the Voyager framework to include an asynchronous messaging system, allowing agents to exchange words and incorporate our collaborator feedback into their belief system to be used in subsequent actions. This communication channel is essential for adjusting AI behaviors based on real-time chat messages, ensuring that the agents’ actions are continuously refined in response to collaborative interactions. This modification was crucial to facilitate real-time interaction between collaborators. To implement this feature, we used the Mineflayer API to extract Minecraft chat data. Moreover, we executed this process in a multi-threaded manner to avoid blocking the agent execution, ensuring that it can perform its tasks while simultaneously listening and incorporating collaborator feedback. However, now that we have a way to communicate, we are still lacking a way to represent our

Step 1



Step 2



Step 3 Task success



Figure 5: An example of collaboration

world model and how to act in the world. Indeed, we want our system to be able to represent perspectives in the theory of mind framework to enable collaboration.

BigToM World and Partner Perspectives Representation Now that we have a chat system, the question of how we represent and internalize those discussions remains open. Thus, we need our agent to have the “ability to represent and internalize those communications” as outlined in section 4.1.

To represent the internal beliefs of the agents, we chose BigToM as it offers a way to ground LLM reasoning for our Collaborative Voyager. Indeed, it proposes a framework based on the theory of mind that allows agents’ beliefs to be updated through collaborative experiences. We chose this template as it was inspired by our own human reasoning and aligned with the classical belief-desire-intention (BDI) model originally proposed in the 90s [42].

The original Voyager lacks any way to represent the agent’s world model. Indeed, the original Voyager did not create a world model; instead, it relies purely on skills that can be composed. Furthermore, as identified in the Voyager shortcomings in Appendix A, the system cannot backtrack wrong reasoning and thus cannot correct itself. For example, by wrongly assuming that the system needs a shovel to pick up dirt, all subsequent codes will include this information even if the system is told this information is erroneous. Thus, our Collaborative Voyager needs to be able to correct wrong assumptions

We need to give a world model to the LLM. Our objective grounds LLM into the theory of mind, which aligns with our social intelligence hypothesis. This offers a way to represent the agent’s beliefs of the

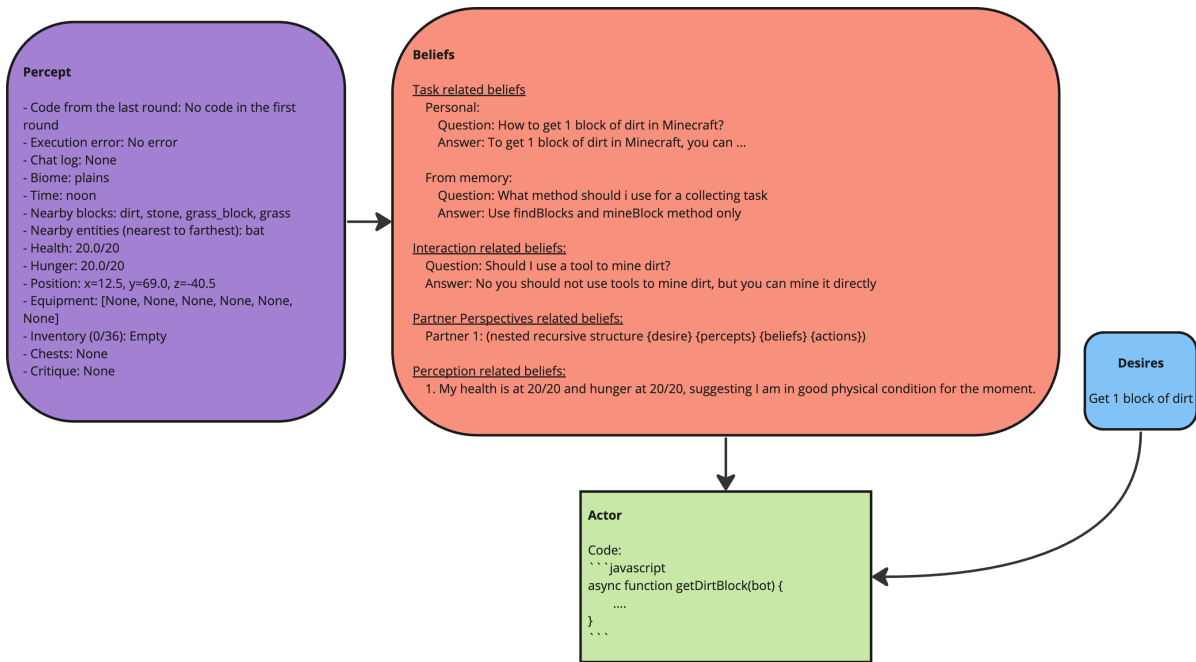


Figure 6: BigToM template explained

world formally and allows the agent to take perspective. Indeed, we also represent the other agents' beliefs to enrich our world understanding and better collaborate. We let an LLM fill out the BigToM template mapping free-form text to a graph.

The introduction of the BigToM causal template allows agents to form a structured representation of their environment, encapsulating their beliefs, desires, perceptions, and intended actions within a unified framework grounded in the theory of mind. The BigToM template empowers agents with the ability to reason about their surroundings and the outcomes of their actions more effectively, mirroring human ToM capability.

Figure 6 shows an example of the BigToM template at step t . The task from the Voyager paper forms the desire for what the agents aim to achieve. Perception is information that agents get around the world from its different sensors (the Minecraft API). The perception and the code critic also done in the original Voyager paper create a set of beliefs. These beliefs are in a free-form text format. The beliefs and desires are then fed to the action agent that will output the Minecraft code. The agents also create a BigToM template for the other collaborators.

Explanation of the template:

- **Desire:** The task the LLM wants to accomplish, for example, getting dirt.
- **Beliefs:** The system's beliefs about the world, such as how to do a task or why a code execution failed.
- **Percept:**
 - This is the state of the game, how the system senses the world.
 - Extracted through the Minecraft API.
 - Examples of data: inventory, nearby blocks, execution errors of the code, critical actions on code perception.
- **Actor:** Module that acts on the world environment. An LLM that takes the beliefs and desires and outputs code that results in embodied actions.

The actor module acts on the world environment by producing code snippets. The actor module's LLM takes the beliefs and desires and output code that results in embodied actions thanks to the Mineflayer API. The actions are taken based on the BigToM template. After the code execution, the agent updates its beliefs. As we will see in the interaction LLM section 4.2, beliefs are also updated through

collaboration. Those constant, up-to-date beliefs about the world guide the agent in subsequent actions.

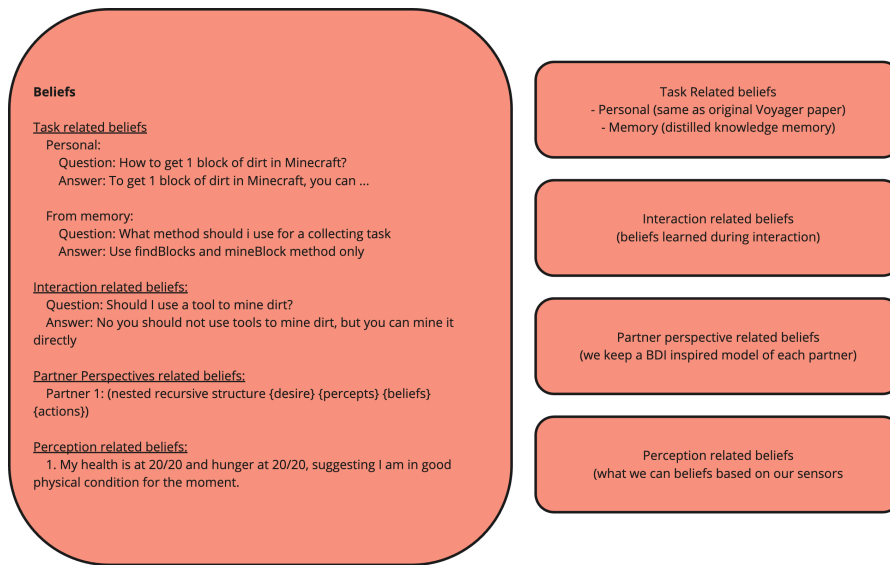


Figure 7: Beliefs breakdown

Figure 7 shows how beliefs are organized in more detail. Due to the architecture of the RAG system, we store all long-term information in the form of QAs. The memory module section will explain how this information is queried and stored to enable learning. We divide the beliefs into four categories: 1) task related beliefs 2) interaction related beliefs 3) partner perspective beliefs, 4) perception related beliefs. The task related beliefs are based on the original Voyager paper, where the LLM uses self-reflection to define how it should approach the task. The memory served to store long-term insights learned through play. We introduce the interaction beliefs based on the message received in the chat while the agents are attempting the task. Perceptions-related beliefs are beliefs directly related to the sensory information received via the Minecraft API and the code critic. We also represent the partners' perspectives by keeping a BigToM representation of each partner. We keep for each collaboration a partner perspective that is similar to the agent's own BigToM template. Thus our representation contains a nested template for each collaborator in order to determine the intention of the agents and their views on the world. This is done so that the collaboration is done with a mental representation of the other agents in the environment.

To summarize, by using BigToM, we can formalize the system's beliefs and allow them to be updated constantly. We now have a representation for the agent of its world through our BigToM-inspired causal template and a way for the agents to receive communication through our chat module. However, we are currently missing the link between the communication received through the chat and our world representation.

Interaction LLM Now that we have added the ability to communicate and represent the agent's internal world state, we are missing the link on how to internalize and integrate the communication into the BigToM template. We therefore need our agent to have the "ability to interpret its communication" as outlined in section 4.1. Our interaction LLM relies on the chat module previously described. It takes a collaborator's message and updates the beliefs to better align with the collaborator. The interaction LLM module modifies the causal template as shown in Figure 8. This makes the beliefs more grounded, thanks to perspective-taking.

The interaction LLM component is tasked with processing the insights gained from the Chat Module and integrating them into the agent's decision-making process. Examples of prompts that are used can be found in Appendix B. By modifying the BigToM causal template based on collaborative feedback, this module ensures that the agents' internal models reflect the nuanced understanding and strategic

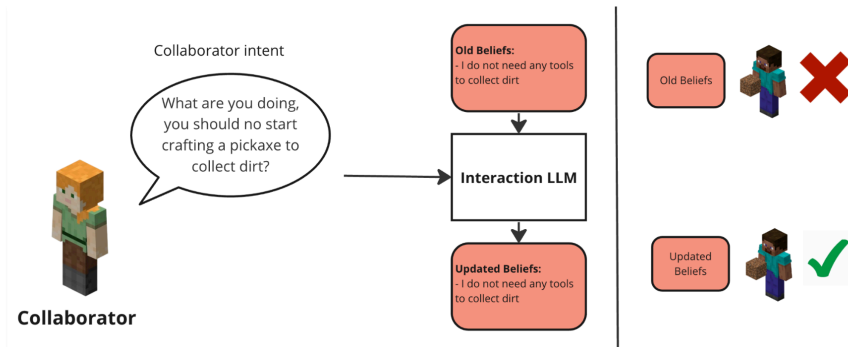


Figure 8: Interaction LLM usage

adjustments shared by collaborators.

Persistent snapshot of the BigToM template To offer better visibility into our agent’s inner workings and enable its learning capability. We record every BigToM intermediate template. This can be equated to the recording of each cognitive phase the agent traverses during task execution. Our final collaborative agent does not use this system to inform future task execution due to the limited context window of the agent.

The current system does not integrate this memory for subsequent calls. Indeed, our LLM context is limited to 8000 tokens. However, newer LLMs could include those snapshots to inform future decisions. Those snapshots enable the reconstruction of the whole cognitive reasoning of the LLM, including the actions taken and the consequences of those actions. We believe this is useful for driving future development in the direction of LLM which can learn and reason on top of their own mistakes. To still fit the insights learned during playing, we need to condense those findings into a distilled knowledge that saves the most important aspect of the system.

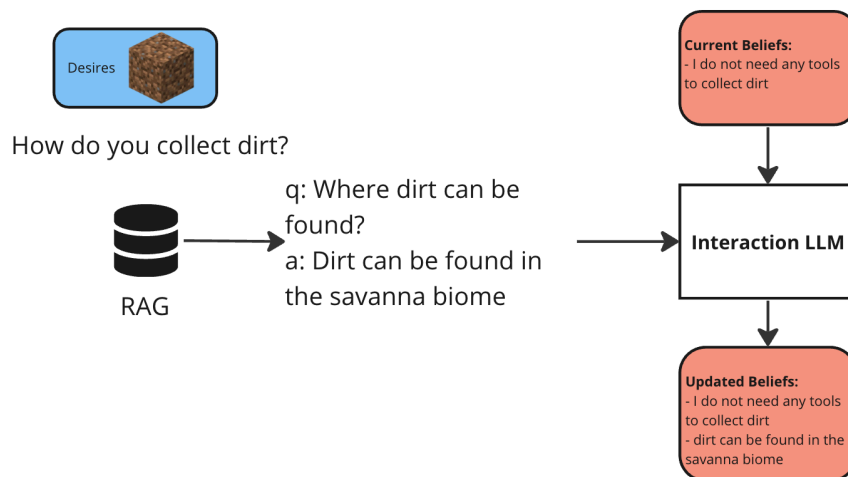


Figure 9: RAG retrieval

Distilled Knowledge Memory To finally complete our Collaborative Voyager agent, we need it to have the “ability to learn from interactions” as outlined in section 4.1. We implemented a Retrieval-Augmented Generation (RAG) system that enables the agent to learn how to solve issues previously addressed through collaboration over time. This distilled knowledge memory allows us to inject insights learned during collaboration into the LLM prompt.

Incorporating learning and memory into the Collaborative Voyager Framework, the Interaction Memory component enables AI agents to retain and utilize the knowledge acquired through collaboration over time. Using a Retrieval-Augmented Generation (RAG) system, this component allows agents to build upon previous collaborative experiences, progressively enhancing their performance and problem-solving capabilities. This aspect of the framework not only facilitates immediate improvements in task execution but also contributes to the agents' long-term adaptive learning and behavioral evolution, reflecting the Adaptive Learning Hypothesis proposed in this thesis.

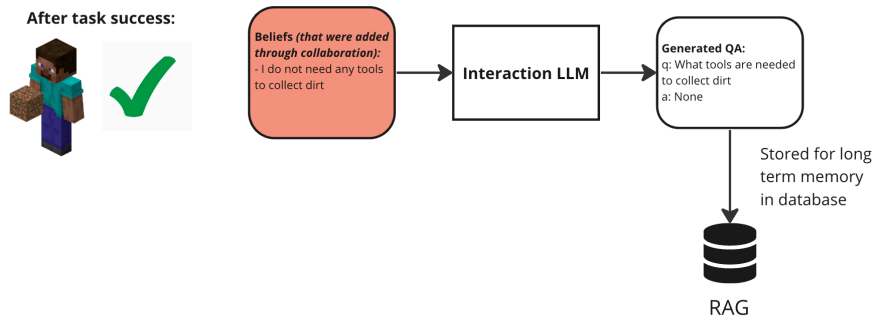


Figure 10: RAG adding

Figure 10 shows how new information is added to the memory. The beliefs are transformed into QAs as this is the format expected by the RAG system. We then add those QAs to the RAG. Lastly, with our interaction with LLM, we filtered down the answers that were retrieved. To remove duplicates and select the relevant QA. You can find the relevant prompt in Appendix B. This memory, therefore, informs future code snippet generation to be more grounded thanks to the collaborator insights gained during previous play sessions. This distilled knowledge is then retrieved on-demand to inform the behavior of the agent.

4.3. Difference between Voyager and our Collaborative Voyager

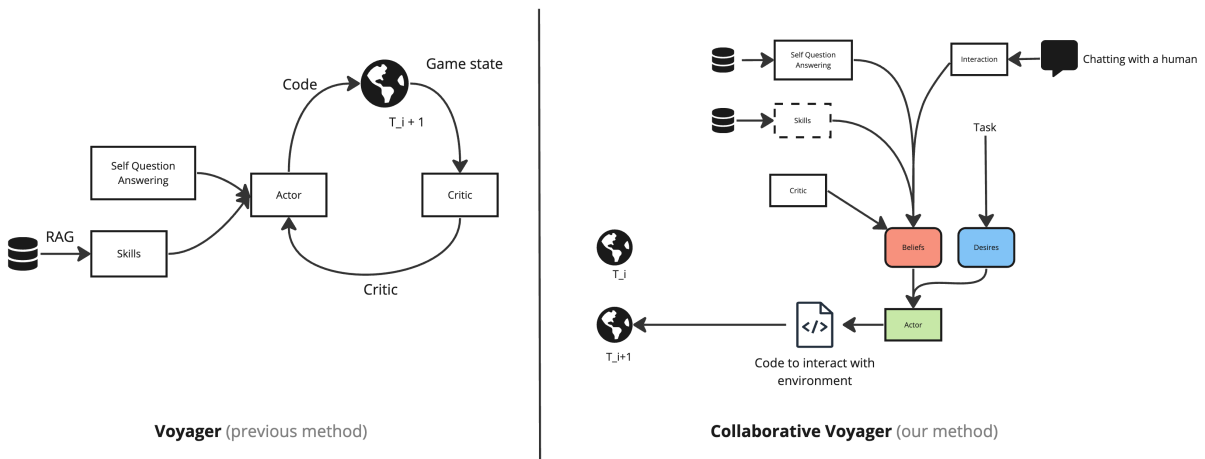


Figure 11: Architectures of Voyager and our Collaborative Voyager

Figure 11 shows that we performed a complete refactoring of the Voyager agent to make him socially aware and able to collaborate. In Section 4.1, we defined four objectives that would make a Voyager agent socially aware: 1) the ability to receive messages, 2) the ability to represent and internalize those communications, 3) the ability to interpret its communication, 4) the ability to learn from interactions.

Figure 12 shows in detail the different elements of our method. By introducing the following modules, we created a socially aware Voyager:

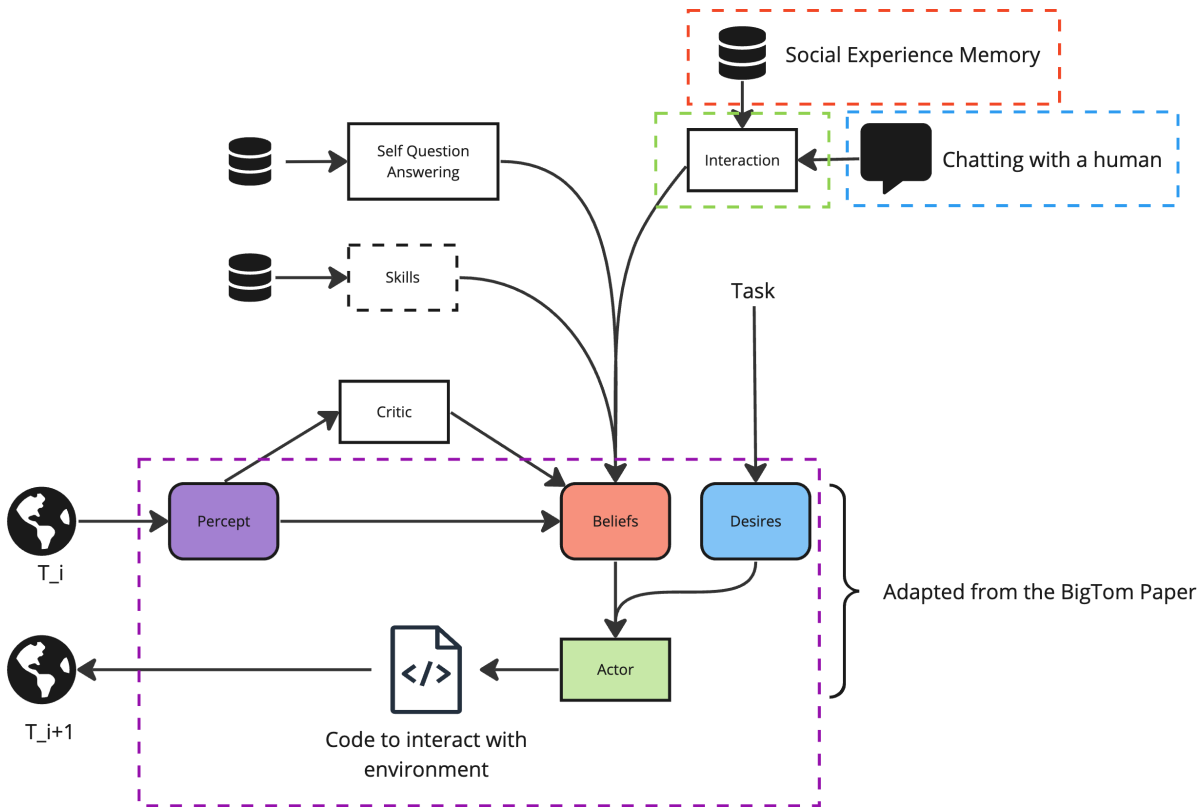


Figure 12: Collaborative Voyager architecture in detail

1. **Chat module:** (*in blue*) it gave the Voyager the ability to receive external communication. In the form of chat messages via the Minecraft in-app chat
2. **BigToM world representation:** (*in purple*) As the agents need to work with others and be able to represent the world in a socially aware context, we rely on the BigToM template to save the agent's internal state. We adjusted this template to the specifics of the Minecraft setting. This allowed us to represent the agent world model and its beliefs about its collaborators while being grounded in the theory of mind literature.
3. **Interaction LLM:** (*in green*) To fill out the BigToM template and internalize the communication the agent receives, we use an LLM. Furthermore, the latter is also responsible for transforming the communication into QAs for the memory component that we introduced.
4. **Social experience memory:** (*in red*) To learn from interaction, we added a memory component based on Voyager's RAG system for its skill library. We store the insights learned from communication during play in our memory. This is then used to enrich the LLM prompt of the agents, such as, for example, allowing the code generation to be more grounded.

The Collaborative Voyager Framework presents an approach to enhancing LLM-powered agents' capabilities through human collaboration. This allowed us to investigate how our research question "How can collaboration improve the problem-solving abilities of LLM-powered agents?"

Figure 13 shows the system in action. Firstly, the agent has a false belief. The collaborator has some insight on how to fix the bad behavior of the weak agent. The collaborator then talks to the agent to give it some advice on potential solutions. Thanks to the interaction LLM, the feedback is internalized, and the previously false belief is corrected. This collaborative insight then leads to the task's success. The agent will finally save this insight into his memory and retrieve it when needed to accomplish future tasks more effectively. You can also see in figure 14 the agent working on the dirt task in Minecraft with the associated chat open where the collaborator communicates with the agent; in this picture, the collaborator is me.

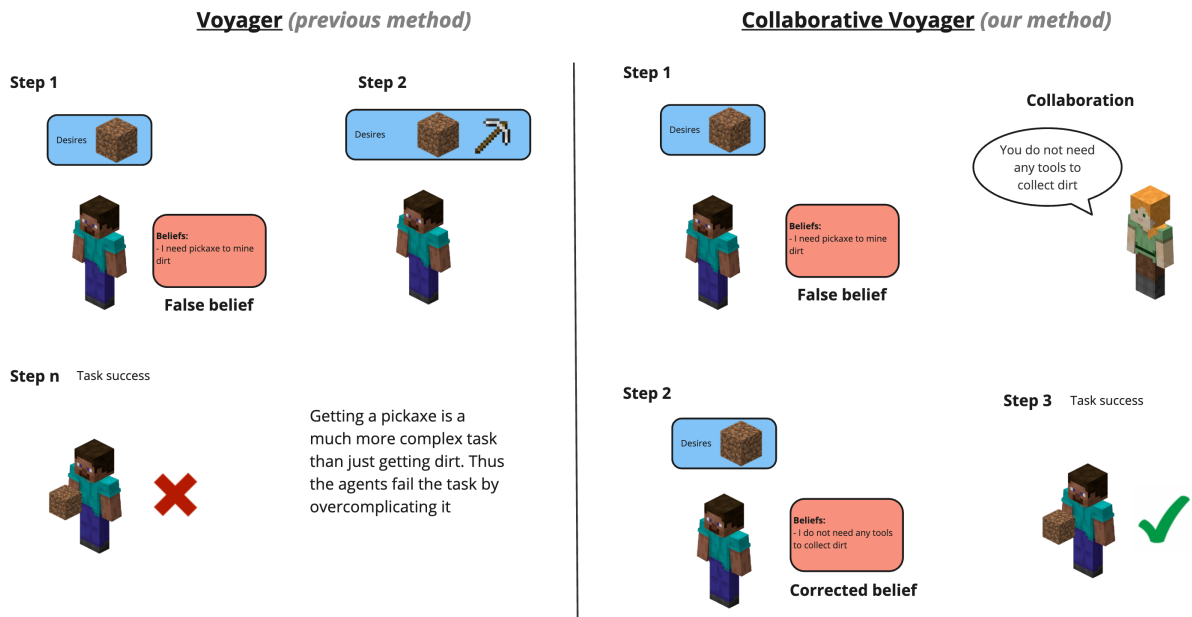


Figure 13: Example of a wrong belief fixed through collaboration



Figure 14: The agent working on getting dirt while talking to a collaborator

5

Experiments

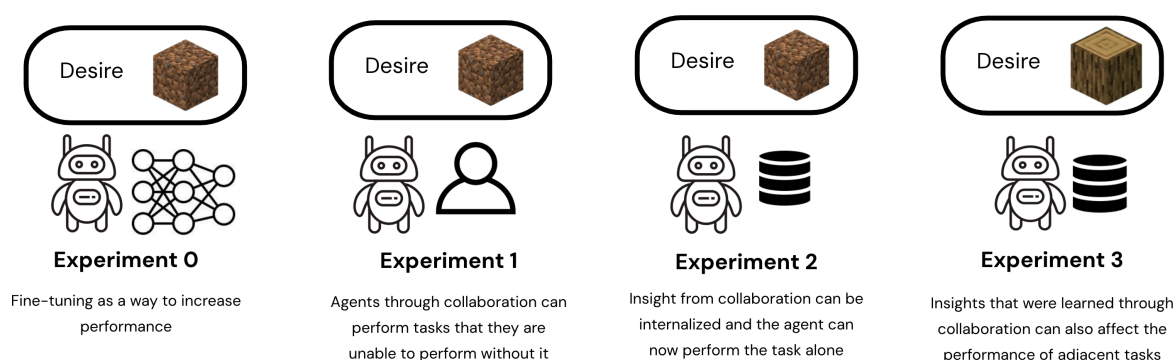


Figure 15: Overviews of the experiments

This chapter details the experimental phase of our research, aimed at testing our hypotheses related to enhancing AI agent capabilities in Minecraft. Initially motivated by the limitations of open-source LLMs in executing basic tasks within the Voyager framework, our exploration took us through several stages of fine-tuning and method adjustments. Figure 15 shows the four experiments that we will conduct to validate the hypothesis that collaboration could significantly improve single-agent performance, potentially laying the groundwork for more advanced socially aware agents.

5.1. Experiment Setup

Figure 16 shows the setup of a single run of an experiment. This is then repeated 15 times to obtain more robust data. In this example, the agent is tasked with “collecting dirt.” We let the agent attempt the task by generating a code snippet that interacts with the world. The bot has five attempts to do so before considering the task a failure. Additionally, we introduce a timeout for the task attempt of 10 minutes. Indeed, sometimes, the agent creates infinite loops in his code snippet; by introducing a timeout, we can capture those task failures. We stop when the bot thinks that it has managed to perform the task. This figure is an example of one run; we do 15 independent runs to then measure the performance on the task in a statistically significant way. In the spirit of open science, each run is associated with a log that saves the entire state of the agents at any time step. In addition, the code is accessible in our GitHub repository. Lastly, all the experiment data were saved into a single file so that our analysis could be reproduced.

Before we discuss the different metrics we will measure, we must distinguish between agent perception and the real-world state. Indeed, agents may mistakenly assume that they completed a task when they did not. This difference is related to LLM hallucination problems.

Figure 17 shows how we collect metrics. Indeed, after the agent is done with its task, we check the game state, which serves as ground truth. With the game state, we can directly check the agent’s

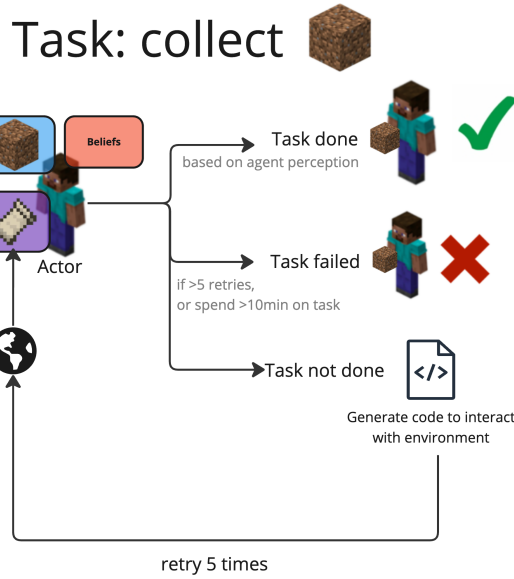


Figure 16: Detailed example of one run of the agent for an experiment

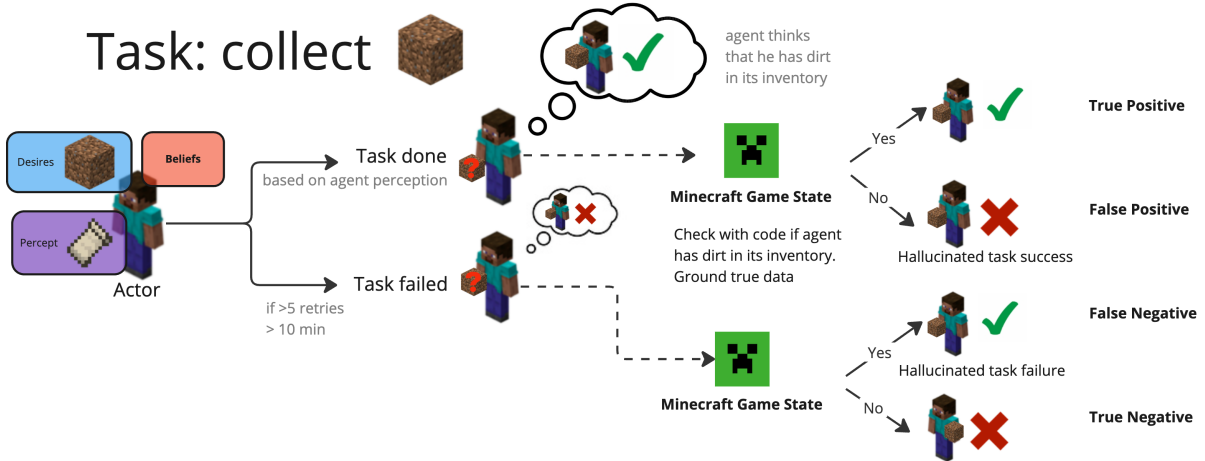


Figure 17: Example of one run of the agent for an experiment with the associated collected metrics

inventory to determine if its perception is correct. To give a full example, the agent may think it has successfully executed the task and got dirt, but when we check its inventory, it is empty. Thus, we have a false positive. We also called this the “hallucinated task success” as the agent wrongly believes that it managed to succeed at the task when it did not. The false negative, called the “hallucinated task failure,” is when the agent thinks that it failed at the task when it has, on the contrary, succeeded at it and has dirt in its inventory.

Figure 18 shows the confusion matrix for the agent. The thought bubble represents what the agent thinks about the world. In its hands is depicted what the agent has in its inventory. We use the Minecraft API to get the agent’s inventory. This gives us a confusion matrix about the attempted task.

Evaluation Metrics Our evaluation framework focuses on several metrics to assess the effectiveness of our methodology.

- True Positive (TP): Reflects the proportion of correctly identified positive instances. How often did the agents get dirt out of 15 runs while thinking they had dirt?
- False Positive (FP): Measures the rate of incorrect identifications. How often did the agents not

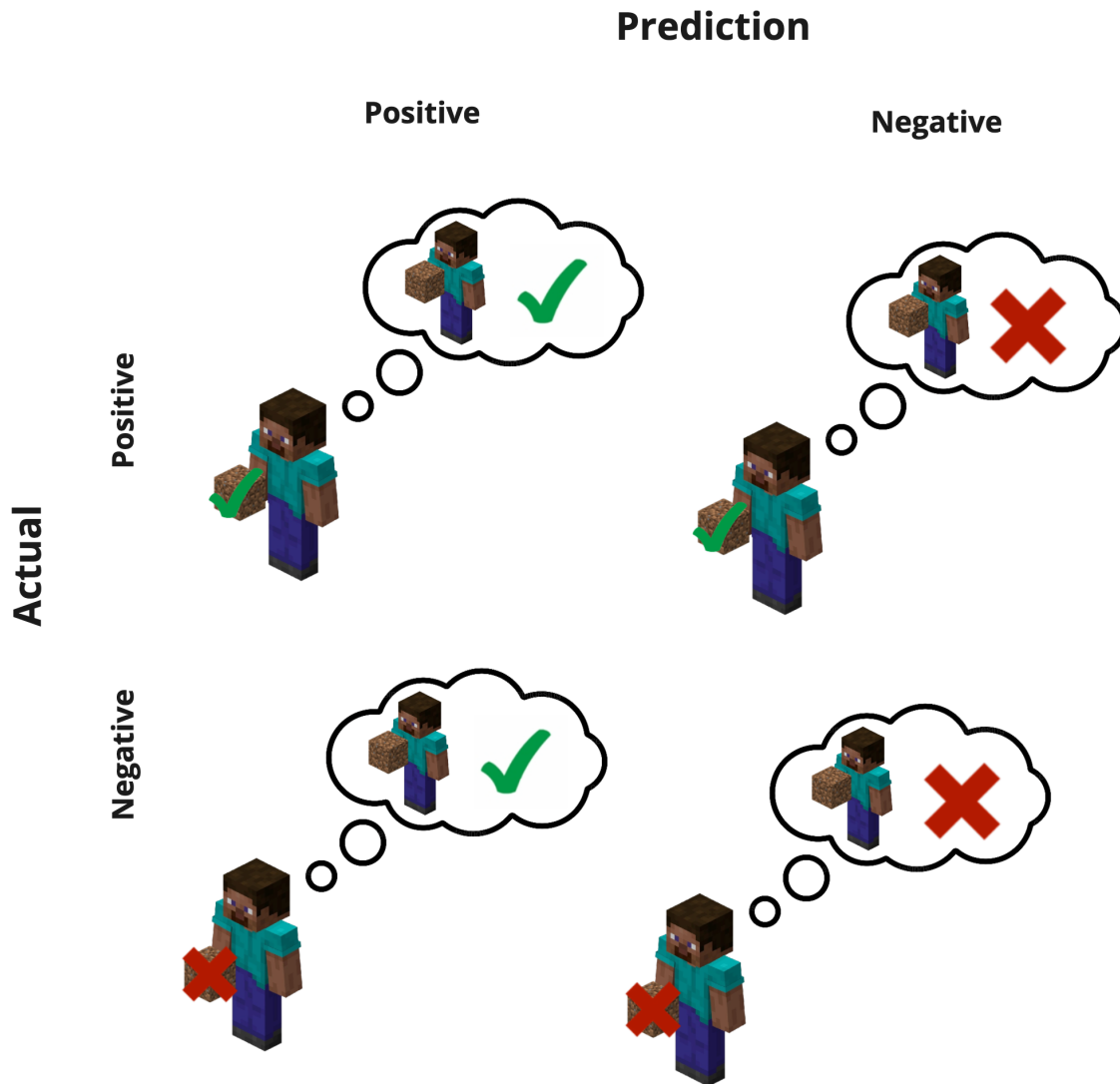


Figure 18: Confusion matrix

get dirt out of 15 runs while thinking they got dirt?

- False Negative (FN): Indicates incorrectly classified as negative instances. How often do the agents get dirt out of 15 runs while thinking they did not get dirt?
- True Negative (TN): Represents the instances correctly identified as negative. How often did the agents not get dirt out of 15 runs while thinking they did not get dirt?
- Task Success (TP+FN): Quantifies the overall tasks performed, including true positives and false negatives. How often do the agents get dirt out of 15 runs?
- Average Time to Success: Provides insight into the average duration taken to achieve successful outcomes. The average time to complete a task successfully (TP + FN).
- Average Time per Round: Evaluates the average time expended per iteration or round of task completion. The average task completion time (TP + FP + FN + TN).

Expected Outcomes To understand the impact of collaboration, we tested our Collaborative Voyager agent on different tasks with and without collaboration. This intervention includes providing strategic advice, correcting errors, and offering creative solutions through the chat module. The intervention group included agents who received feedback through the Chat Module. We also introduce a control group: the original Voyager architecture, ensuring that any performance improvements can be attributed to

the introduction of our method. The baseline is established using AI agents that did not receive any form of interaction.

Baseline (with no collaboration) Firstly, we wanted to compare our new Collaborative Voyager setup with the original Voyager architecture, checking that our new method does not lead to any performance loss or gain. We compare both architectures with three models. GPT-4 from OpenAI [43] and Mistral [41], Mixtral [44] from Mistral AI. We evaluated the model's performance on the dirt collection task and judged the task's success by checking the agent inventory for dirt.

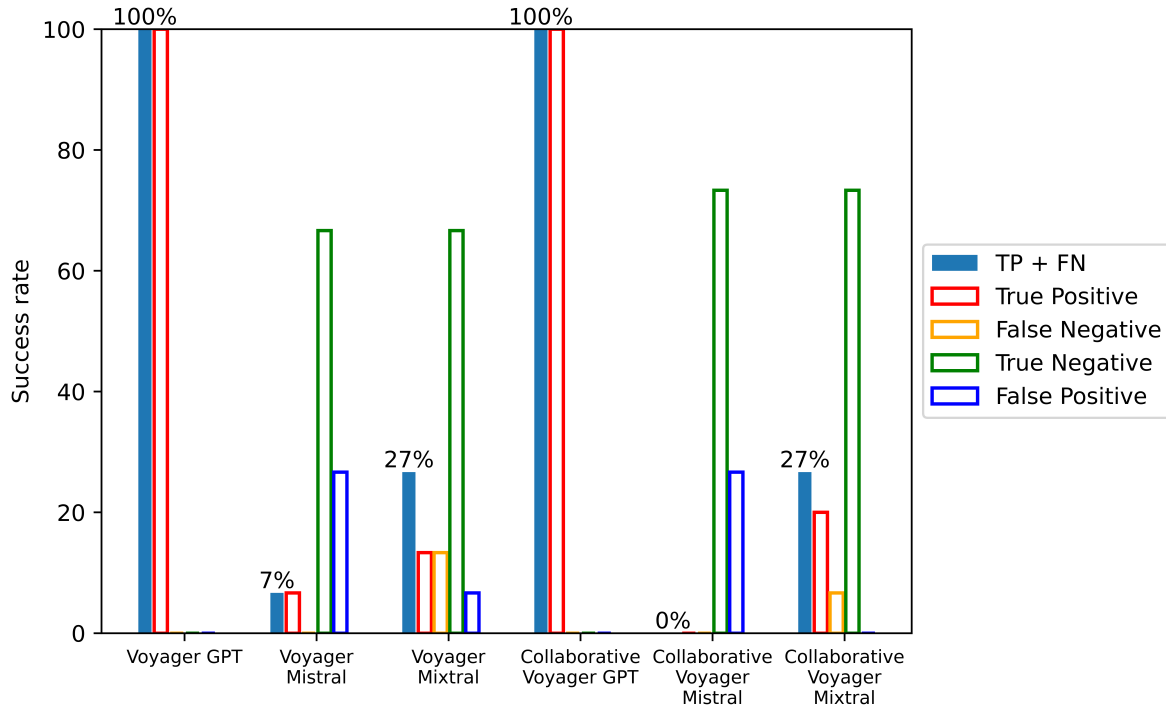


Figure 19: Comparison of Voyager and Collaborative Voyager with a selection of LLM

Results Figure 19 shows that the performance of both architectures is similar when no collaboration is done. Thus, we test our Collaborative Voyager without any type of collaboration and thus only evaluate the performance in a solo setting. We let the agent perform the collect dirt task 15 times to obtain our results and then computed the task's success. We can see that the performance of both systems is globally similar. The only notable difference is that Mistral performs better with Voyager than our Collaborative Voyager (7% vs. 0%). This means that one of the 15 trials was successful. We can attribute this difference to the small number of runs and that with more runs the performance of both would also match. Therefore, we can conclude that the performance of Voyager and Collaborative Voyager are similar in the absence of collaboration.

The following experiments are designed to provide information on how collaboration can enhance the capabilities of AI agents in complex environments. We want to investigate task completion times, success rates, and reduced error rates with and without collaboration. By carefully following this extended experiment setup, the research aims to provide a comprehensive understanding of the potential for collaboration to improve AI agent performance in Minecraft, thereby offering broader implications for developing collaborative AI systems in various domains.

5.2. Experiment 0: Fine-Tuning as a Solution to the Performance Problem of Open-Source LLMs Instead of Collaboration

In this experiment, we explored the potential of fine-tuning to enhance the capabilities of our Mistral-powered agent within the Minecraft environment. Our initial assumption proposed that fine-tuning could be a straightforward and effective strategy to enable an agent to perform basic tasks in Minecraft, such as collecting dirt. To test this assumption, we conducted two distinct rounds of fine-tuning, each utilizing different data sources aimed at augmenting the agent’s performance. The fine-tuning was done on Mistral instead of Mixtral, as the new model was only available after our fine-tuning experiment had already started. To perform our fine-tuning, we used LoRA [22] as this provided a cheap and fast way to tailor our base LLM for the Minecraft domain.

Fine-Tuning v1: GPT Voyager Execution Logs For the first version of our fine-tuning, we used execution logs [45] from a GPT-4 powered Voyager agent to train our model. These logs contain the different text inputs used during runs of Voyager agents and the GPT-4 responses to them. Our objective was to transfer some of the GPT’s capabilities from the Voyager domain to our Mistral model. By fine-tuning with these logs, we could improve Mistral’s performance in a niche domain such as Minecraft with examples of correct reasoning done by GPT. This approach was motivated by the belief that the GPT-powered Voyager’s experiences, encapsulated in the logs, could provide valuable insights and strategies for navigating the game world more effectively.

Fine-Tuning v2: Incorporating Wiki Data and Mineflayer API Documentation In the second round of fine-tuning, we expanded our data sources to include information from the Minecraft Wiki [10] and the Mineflayer API documentation [46]. This was done to directly integrate comprehensive Minecraft knowledge and Minecraft programming interfaces into Mistral’s knowledge base. By doing so, we aim to equip the agent with a richer understanding of the game’s mechanics and a more nuanced ability to interact with the game world through the Mineflayer API.

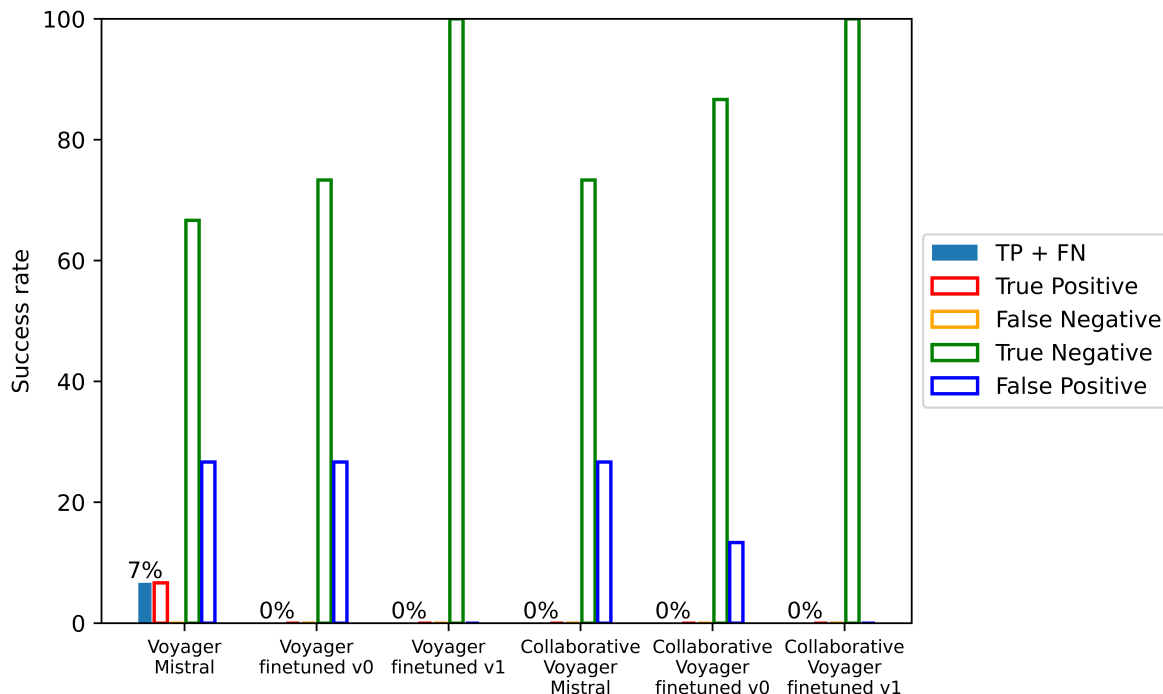


Figure 20: Comparison of the performance of Voyager and Collaborative Voyager after fine-tuning

Results Figure 20 shows that the fine-tuning did not allow the agent to perform the task. We see that fine-tuning even decreases performance, with the default Mistral achieving to get dirt once out of 15 runs. As displayed in our Appendix C table 2, we also notice that the fine-tuning process introduces a lot more timeout than the default Mistral for the v2 of fine-tuning. We believe this is due to the lengthy nature of text on the Wiki or Mineflayer codebase, which makes the model less succinct in its answer. Furthermore, another criticism of the v2 version is that pretraining is done after the model has been fine-tuned. Thus, the model is catastrophically forgetting the instruction tuning part of its training. This suggests that these new data sources have detracted from the model's efficiency, leading to less concise and effective decision-making. This approach may have inadvertently diminished the instruction-tuning capability due to the fine-tuning process, further contributing to the lackluster performance. However, we show a qualitative decrease in Minecraft facts-related hallucination, with the model better able to answer questions about the Minecraft world and how to achieve certain tasks. There was, however, still a lot of hallucination on the Mineflayer code; this could be attributed to the small usage of this library globally, making it difficult to produce quality and complex code due to the lack of complex and real-world examples.

The results did not align with our expectations. Ultimately, we decided to continue our research with the default Mixtral configuration rather than the fine-tuned versions, as they did not provide any benefits. Thus, our approach to fine-tuning was insufficient to bridge the performance gap compared to the GPT-4 model.

5.3. Experiment 1: Collaboration Improves Weak Agent Performance

This is the main experiment of our research. We investigated whether collaboration can help a weak LLM agent become stronger. We define a weak LLM as an LLM struggling with nuanced tasks and performing significantly below state-of-the-art models. In our case, this represents our open-source LLM that is not able to match the performance of GPT in the Voyager framework. Our goal is to determine whether collaboration can address the limitations of weak AI agents in performing basic tasks in Minecraft. To do so, we conducted a series of 15 runs that focused on a simple resource collection task: collecting dirt. This experiment compared the agent's performance with and without collaboration. During each run, another collaborator is present to help the agent that we are testing.

We have two types of collaborators.

- **Collaborative Voyager Agent:** We spawn two instances of our Collaborative Voyager agent, where they must help each other to solve the task. We expect those two agents to collaborate to solve the task better. Indeed, they both have perspective-taking capabilities; thus, they can relate to and help their collaborator.
- **Human Collaborator:** We also used a human collaborator to determine if a stronger collaborator could elicit an increase in performance in our weak agent. Humans are also social creatures and thus can help each other in need. Therefore, they represent what the future of a socially aware AI collaborator could achieve. This is a first step to demonstrate the feasibility of our method.

Results Figure 21 shows that our multiple Collaborative Voyager setup failed to increase performance. However, partnerships with humans show that collaboration can improve a weak LLM. We see that performance with human collaboration is much better than the baseline, where AI collaboration did not improve performance.

We believe that the inability of the multiple Collaborative Voyager setup to increase performance is due to the reduced social capability of LLM compared to humans. Indeed, the current LLMs do not match human social intelligence and often fail in ToM tasks for the time being. LLMs are not trained in cooperative settings and do not yet know how to collaborate fully. Indeed, we have seen that current open-source LLMs already struggle in embodied environments; thus, collaborating in such environments is difficult for them. Due to lack of time, we were not able to do a full study on why the multi-agent setup failed to perform. Other potential reasons could be that the agents do not know when to intervene, as discussed later in our future work section 6.4. Furthermore, current LLMs are not trained to be an embodied collaboration, so this is an out-of-distribution task that would require

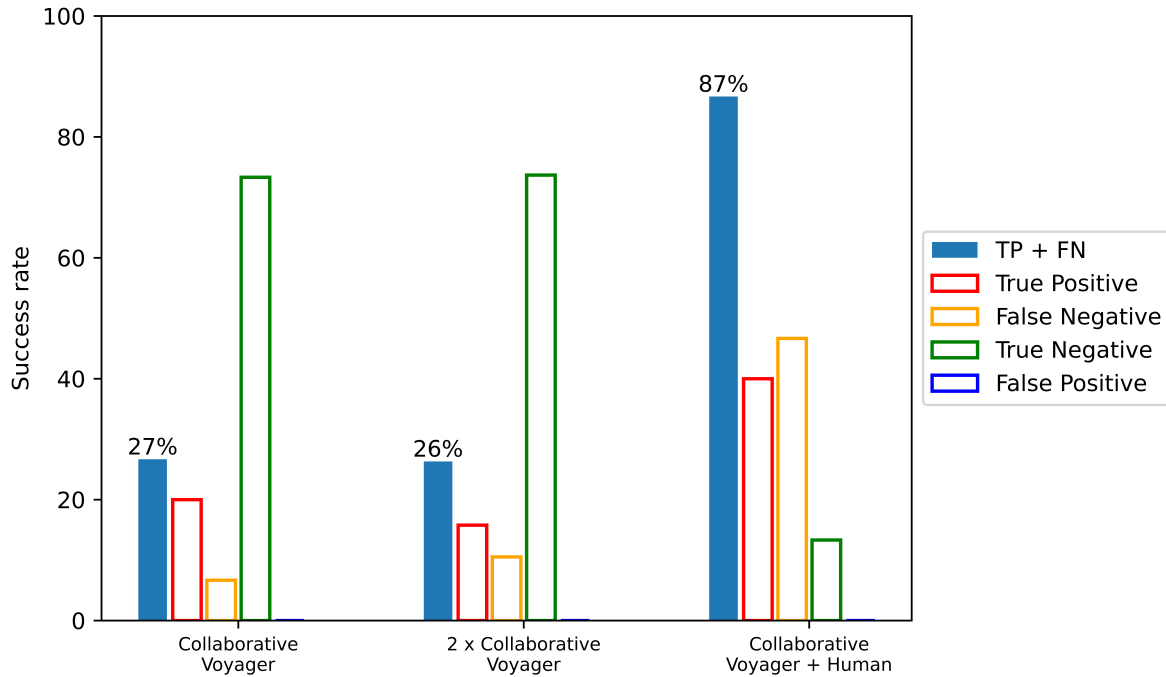


Figure 21: Comparison of Collaborative Voyager's performance in the presence of a collaborator

further investigation. Indeed, current insights are highly vague and general, preferring correctness over actionable insights.

One interesting result that can be seen in table 3 is that the multiple collaborative Voyager setup spends less time per task than the other setups: 88 seconds compared to 227 seconds and 209 seconds. for the solo Collaborative Voyager and the Collaborative Voyager and Human setup, respectively. As the performance of the multiple collaborative setup is not better than the baseline and the number of false negatives is higher, it could mean that both agents wrongly agree precipitately that the task has failed when it did not. This could be another reason why the multiple collaborative Voyager setup fails to improve performance.

As human collaboration is effective, we can rely on humans to demonstrate the feasibility of our method and show that a good collaborator can help a weak agent while LLM catches up to human capability. However, one notable restriction that will be further elaborated in our discussion section is that only one human collaborator was tested. I was the only one to play with the system, as the goal was to do a feasibility study to determine if collaboration could improve performance. We believe this still shows that weak embodied LLMs can learn how to improve their abilities in the presence of a good collaborator.

This experiment underscores the potential of collaboration to compensate for AI agents' shortcomings. The improvement suggests that real-time collaboration can significantly refine AI agents' decision-making processes, even without advanced fine-tuning. We, therefore, have demonstrated the feasibility of our method. In fact, with collaboration, we could guide the agent to collect dirt when it was previously unable to do it on its own.

Example of Collaboration Appendix A details all the common errors the agent commits while solving a task. However, through collaboration, we can guide the agent so that it can complete its task. These are examples of human guidance messages that were sent to the agent.

1. Guide the agent to make the most simple plan: We tell the agent to avoid equipping tools for mining and to directly mine blocks without using tools. This prevents the agent from not being able to complete a task because it makes it unnecessarily complicated

– *Chat message sent:* “Do not try to equip any tools. Just try to mine the block directly.”

- *Explanation*: Tools are not required for mining dirt and grass blocks, allowing the agent to focus on the mining action instead of crafting an unnecessary tool first.
2. Simplify the function code: Design the function as simple as possible, avoiding unnecessary complexity.
 - *Chat message sent*: “Make function as simple as possible”
 - *Explanation*: A streamlined function reduces the risk of errors and improves the agent’s performance by focusing only on the essential needed actions.
 3. How to use specific methods and fix coding issues: The code critic is quite vague. The agent is often confused about how to fix a given problem.
 - *Chat message sent*: “bot.dig method only take a single argument.”
 - *Explanation*: Due to a lack of knowledge in the Mineflayer API, the agent does not know how to fix the code even after being given the runtime error.
 4. Focus on specific methods: By focusing the agents on certain methods, we can limit the search space and guide the agent.
 - *Chat message sent*: “use findBlocks and mineBlock method only”
 - *Explanation*: A lot of potential code can mine a block of dirt, but most are over-complex. We can help the agent focus on the right and easiest way to solve the problem, making the agent more robust.

5.4. Experiment 2: Learning from Mistakes - Post-Collaboration In-Distribution Task

In this section, we test how well our agents could learn from collaboration. We compare three different settings:

- **No-Collaboration Agent**: Represents the agent’s state that did not collaborate.
- **Live Collaborative Agent**: Reflects the agent’s state during active, real-time collaboration with a human.
- **Post-Collaborative Agent**: Indicates the agent’s state after the collaboration has occurred while using the memory module.

Again, we did 15 runs of the agents with the task of collecting dirt and assessing the success rate.

Results In figure 22, we see that having distilled knowledge from collaboration increases performance. Indeed, we see that without collaboration, the agent can rarely perform the task (27%). This is in sharp contrast to the agents that have collaborated. The performance of the post-collaborative agent is better than the live interaction one (93% vs 87%). This was surprising; the live interaction performed worse than expected. I expected the runs with a human to improve, given that mistakes can be corrected in real-time. However, I believe that memory ultimately performs better because the beliefs in memory are validated, and the system knows how to use them as they previously led to task success. This contrasts with the live human guidance provided, where LLM may be unsure how to interpret the messages. This experiment shows that AI agents can learn from collaboration and improve their performance in subsequent runs.

5.5. Experiment 3: Post Collaboration Out-of-Distribution Task

In this experiment, we want to investigate whether collaborating on a task can improve the performance of an adjacent task. Indeed, our agents have been trained to collect dirt via collaboration. We determined that it can indeed learn insights that allow it to now collect dirt on its own. We decided to compare whether the agent’s post-collaboration on the dirt task is better than a pre-collaboration agent on gathering wood. Indeed, both tasks are similar; they do not require any tools and are gathering resources tasks.

Our two different agents are:

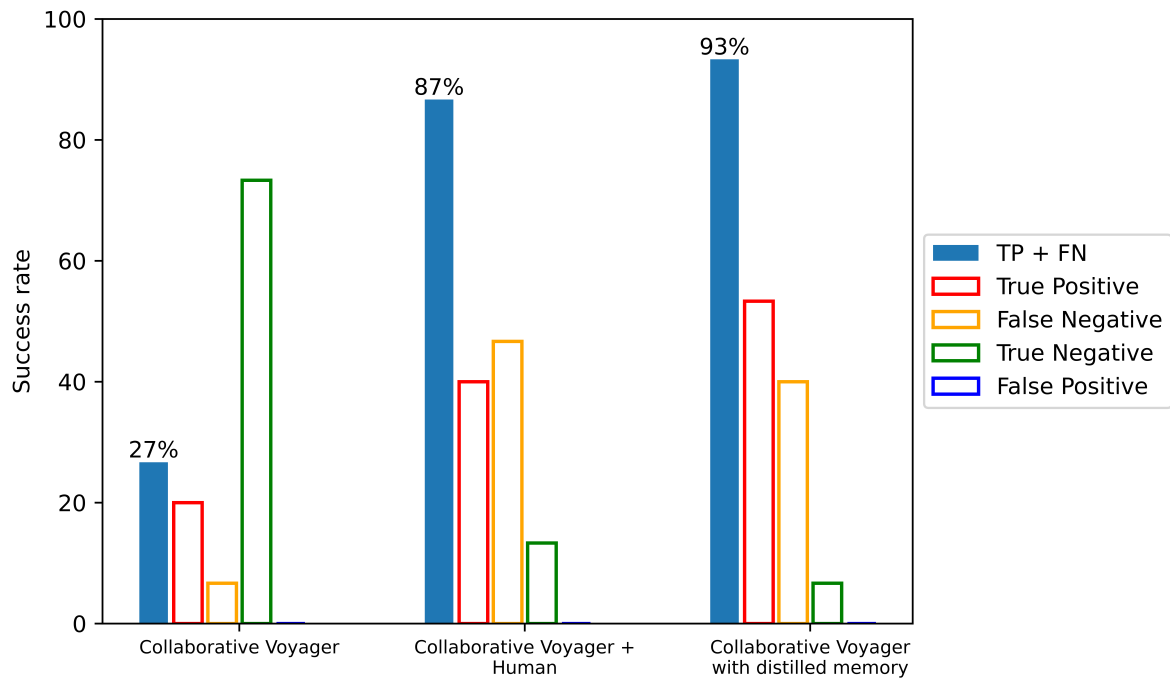


Figure 22: Comparison of Collaborative Voyager's performance with the distilled knowledge memory enabled

- **No-Collaboration Agent:** Represents the agent's state that did not collaborate.
- **Post-Collaboration Agent:** Indicates the agent's state after the collaboration has occurred. The agent collaborated on the task of collecting dirt but is now evaluated on the task of gathering wood.

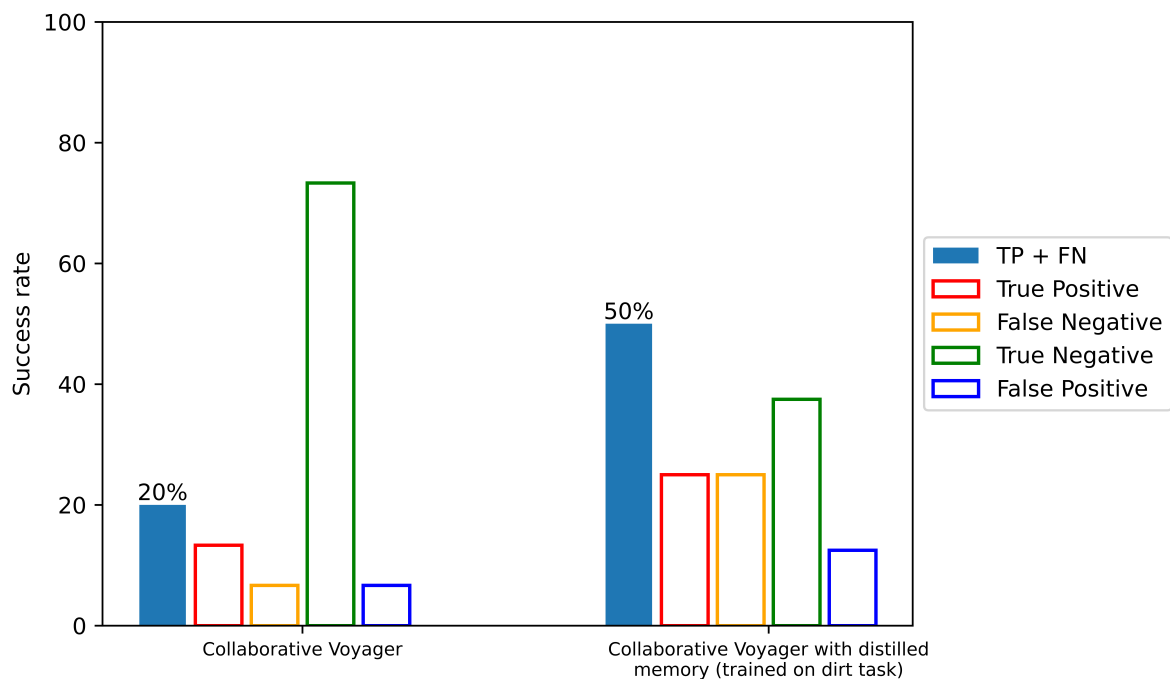


Figure 23: Comparison the Collaborative Voyager's performance on out-of-distribution tasks (collecting wood)

Results Figure 23 shows that the agent that has collaborated on an adjacent task now performs better. This shows that collaboration insights can be transferred between tasks, creating emergent capability. The insights learned from one task can be applied to other settings. We see a performance improvement from 20% to 50% success rate.

By collaborating on a task (collecting dirt), the agent is now better at other tasks, such as (collecting wood). Of course, this is still limited, as we did not compute the performance across various collecting tasks due to time constraints. We leave this for future research.

However, we note that the performance of the post-collaboration agent on task “wood” (out-of-distribution) is relatively lower than with the post-collaboration agent on task “dirt” (in-distribution) (87% vs. 50%). The tasks are essentially the same except that in one, the target is “dirt,” and in the other, it is “wood.” This could be due to the format of the beliefs in the QA system being too rigid to generalize to examples involving “wood.” Also, for future work, it would be interesting to collaborate on more tasks and then measure the performance on out-of-distribution tasks as a function of the number of different collaborative tasks that the agent has participated in.

5.6. Summary

Our goal was to investigate long-term behavioral changes in AI agents subjected to collaboration interventions, assessing their ability to adapt to new tasks and challenges. We saw that an agent that has collaborated now performs better on the task it collaborated on and other adjacent tasks.

We observed behavioral changes and improvements in performance over time. The agents exhibited significant behavioral adaptation and consistent performance improvement, demonstrating a greater ability to execute tasks more effectively in the long term.

These results indicate the potential of collaboration not only for immediate performance enhancement but also to induce long-term behavioral changes in AI agents. This opens new avenues for the development of AI systems that can adapt and excel in a wide range of tasks through social interaction. This suggests a promising avenue for improving AI performance through socially aware agents, highlighting the potential for more sophisticated learning mechanisms in AI systems.

6

Discussion

With our method, we showed that collaboration enables agents to fix their wrong beliefs about the world in which they are in. This demonstrates the feasibility of an alternative training method that helps with bridging the performance gap between open-source and closed-source LLM. Using a memory module can help our socially aware agents remember, leading to lifelong learning. This impacts the long-term behavior of the agents and enables them to do tasks that they were not able to achieve on their own.

6.1. Insights learned about the field of LLM-agents

During our thesis, we extensively investigated the field of LLM agents. Therefore, we can comment on the limitations of current methods and exciting new developments.

Call for a Benchmark on Minecraft There is a lack of standardized benchmarks for Minecraft. Malmo [47] is the most comprehensive Minecraft benchmark out there, and it has been in development for more than five years. The number of tasks that are evaluated is limited, and it does not try to assess the general performance of an agent compared in an open-ended setting. Furthermore, Malmo lacks any global leaderboard.

Reflection on the Current State of LLM-agents There is a significant performance difference gap between GPT4 and open-source LLM for Minecraft agents [2] [1]. To close that gap, the Jarvis paper investigates fine-tuning as a way to add Minecraft-specific knowledge to open-source LLMs. We tried this approach with our fine-tuning experiment in section 5.2 but were unable to replicate those results.

Additionally, both closed-source and open-source models suffer from a lack of grounding in embodied environments. They were trained to be text models, but not actors in an environment [48] [49]. However, their performance is remarkable, as LLM-powered agents are evaluated on out-of-distribution tasks (being an LLM-based agent). Errors are more pronounced in niche settings such as Minecraft, where specialized knowledge is essential, making LLM-based models unable to perform at a human level.

Furthermore, the current limitation of LLMs makes this problem worse, such as imperfect planning abilities [50]. In addition, they tend to hallucinate, especially in out-of-distribution domains, where they generate responses that are not just incorrect but often irrelevant or nonsensical [51]. LLM do not always know what they do not know; they usually prefer to answer a question and give confidently a wrong answer [52]. This is particularly problematic in Minecraft as it is a niche topic, therefore factual errors accumulate, making the model unable to perform simple tasks. Open-source LLM suffers further from poorer performance in agent settings due to their more limited training datasets compared to closed-source LLM such as GPT4 [2] [53]. Another limitation of LLM-based agents is the speed of the LLM. In my opinion, LLM inference is still too slow for real-world embodiment. Indeed, Voyager and Collaborative Voyager take minutes to achieve tasks that take seconds for humans to do.

Core Issues Surrounding Embodied Agents' Collaboration After having studied this subject for my thesis, I believe that some of the core issues surrounding making embodied agents truly collaborative are:

- **Understanding and Reasoning Capability:** One primary challenge is developing agents that can effectively understand and integrate information from their environment and other agents. This involves not only processing external inputs but also contextualizing them within the given task and the current state and objectives of the agent. This requires a high level of reasoning, namely in a multi-agent system where those agents need to coordinate.
- **Social Awareness:** Effective collaboration is based on the ability of the agents to communicate and understand each other's intentions, capabilities, and actions. For agents, this means not only deciphering messages from other agents but also clearly expressing their intentions and needs. We believe that a truly collaborative agent must have a profound social understanding and capacity.
- **Adaptability and Learning Capability:** Agents must dynamically adapt to new information, strategies, and environmental changes. Collaboration is often dynamic, with partners changing and objectives evolving. In addition, agents need to learn from successful and unsuccessful interactions to improve their collaborative efforts continuously.

By focusing on these strategies, the research and development of embodied agents can progress toward creating more autonomous, efficient, and collaborative systems capable of functioning in complex multi-agent environments. This approach not only advances the field of embodied AI but also paves the way for broader applications where dynamic collaboration is critical.

6.2. Key Findings

The following key findings were identified during our research and experiments:

- Open-source agents LLM fails to match closed-source LLMs in Minecraft
- The role of perspective-taking in agent setting
- Memory plays an important role in driving performance
- Collaboration as an alternative learning strategy

Open-source agents LLM fails to match closed-source LLMs in Minecraft Before delving into the details of our results, we wish to share our experiences with open-source LLMs within the Voyager framework. As the related work mentions, Jarvis [2] demonstrates that open-source LLMs perform significantly worse than GPT in embodied environments. This is also confirmed in our baseline experiment in section 5.1. Anecdotally, getting grass was a much more complicated task than planned. This is one of the simplest tasks that a novice player can achieve in a couple of minutes of gameplay with Minecraft, but open-source LLMs cannot perform this task consistently. Collecting dirt is one of the simplest tasks in Minecraft. Indeed, collecting dirt does not require any tools, and dirt is found everywhere. However, when replacing GPT with other open-source LLMs, we found ourselves unable to even collect dirt, suggesting a profound performance gap. Our initial objective was not focused on collecting dirt, but this task became such a challenging endeavor that we decided to delve deeper into how to enable an open-source agent to accomplish it.

We employed numerous techniques and eventually identified a method that allowed the agent to collect dirt successfully. This performance gain was achieved through collaboration.

The role of perspective-taking in agent setting Creating socially aware agents that understand their participants is crucial for effective collaboration. In multi-agent environments, where agents must work together, the ability to take perspectives is particularly important. Integrating this ability into AI agents can significantly enhance their problem-solving capabilities. By grounding the world state into the BDI framework [42], as demonstrated through the use of BigToM, agents can better understand their collaborators. This framework allows agents to form structured representations of their environment and create an always-evolving world model. The ability to internalize and act on the perspectives of others enables them to learn from each other. AI systems are then not only more effective in specific tasks but also capable of adaptation and continual learning through social interactions.

Memory plays an important role in driving performance Memory is a critical factor in enhancing the performance of AI agents, particularly given the current limitations of LLM context length and the impossibility of infinite memory. Using RAG techniques allows agents to select and retrieve relevant facts and information from past collaborations, facilitating learning from mistakes. Post-collaboration agents often perform better in subsequent tasks due to the retention of distilled knowledge from previous interactions, underscoring the importance of memory and experience in AI learning processes. Furthermore, our experiment on real-time collaboration has shown that immediate human feedback can sometimes be less effective than pre-validated memory. This finding suggests that validated knowledge provides more consistent performance improvements, as agents can rely on previously successful strategies and corrections. Thus, incorporating robust memory mechanisms into AI systems is essential for achieving continuous learning and long-term adaptability.

Collaboration as an alternative learning strategy Collaboration serves as a promising alternative training strategy for AI agents, offering several advantages over traditional fine-tuning methods. Our experiments collectively indicate that collaborative learning can significantly enhance performance without the need for retraining. Through collaboration, agents can continually learn and adapt, extracting valuable insights from their partners. This process not only improves their ability to solve immediate tasks but also enables better out-of-distribution generalization. For instance, collaboration led to improved performance in tasks that were similar but not identical to those initially trained on, demonstrating the potential for collaborative learning to enhance the versatility and adaptability of AI agents. Overall, leveraging collaboration as a training strategy can foster more robust and flexible AI systems capable of continuous improvement and effective handling of diverse and unforeseen tasks.

Review of our Hypothesis The two hypotheses that we postulated in our work are:

- **Collaboration Hypothesis:** Direct collaborator interaction and guidance will significantly correct and enhance the problem-solving capabilities of AI agents.
- **Adaptive Learning Hypothesis:** AI agents can learn from collaborator interventions and adapt to new tasks more effectively, indicating a form of learning and behavioral adaptation.

Due to the limited time, we could not test the full extent of our method on both hypotheses. Therefore, we reduced the potential range of tasks to only collecting blocks, and we focused on collecting dirt and wood as they are some of the most common tasks in Minecraft.

Our experiment 2, “Collaboration Improves Weak Agent Performance,” as seen in Section 5.3, aims to answer our “Collaboration Hypothesis.” By collaborating, our weak agents increase their success rate on the dirt-collecting task; therefore, it shows that, indeed, with a collaborator, a weak agent can improve. By using the BigToM causal template, our agent can better understand and interpret its environment and the mental states of other agents, which enhances collaboration.

Both our experiments 3 and 4, “Learning from Mistakes - Post-Collaboration in-distribution,” and “Post collaboration out-of-distribution Capability,” respectively, intend to answer our second hypothesis: “Adaptive Learning Hypothesis.” We saw that by having a memory module, the agent can now do a task on its own that it was not able to do previously, thanks to the insights learned during collaboration. Furthermore, those insights were applicable to adjacent tasks. This shows that the collaboration leads to long-term behavioral changes in the agent, validating our second hypothesis.

Initially, we believed that fine-tuning on Minecraft specific data would be able to close the gap between open-source and closed-source LLM in the Voyager framework. However, this was unsuccessful.

Our experiments reveal the significant impact of AI collaboration on improving single-agent performance in Minecraft, addressing the initial limitations of open-source LLMs. These findings not only validate our hypothesis but also illustrate the potential for collaboration to enable continual learning in LLM.

How Our Methods Answer The Research Question Our research question is *How can collaboration improve the problem-solving abilities of LLM-powered agents?* as stated in our section 2.

Our method shows that by using the messages from the Minecraft chat to change the internal beliefs of an agent, we can improve its performance. We added to the agent the ability to receive, represent,

internalize, and interpret its communication in order to learn from interactions as seen in Section 4.1. This made our Voyager socially aware. With those new capabilities, our Collaborative Voyager can leverage free-form text from other collaborators to learn and be better at problem-solving.

Later work could focus on more complex task solving, such as multiple-step processes such as building a house or exploring caves in Minecraft. We concentrate on atomic tasks that a human player can achieve in less than a minute in Minecraft; therefore, it would be interesting to test more complex setups. Furthermore, current LLMs are extremely slow; thus, running our system takes multiple minutes to do simple tasks that a human Minecraft player can do in seconds. We hope that future work on inference speed could solve those issues and enable the tackling of more complex tasks.

Implication of Socially Aware AIs The implications of our research extend beyond Minecraft, offering insights into the broader field of AI. By demonstrating the effectiveness of collaboration in overcoming the limitations of current AI models, our work suggests a promising direction for future research and development in AI systems designed for complex, dynamic tasks. In terms of future directions, this research lays the groundwork for exploring more sophisticated models of interaction and developing more socially aware AIs.

6.3. Limitation of our research

While providing valuable insights into the dynamics of collaboration in Minecraft, this study is subject to several limitations that should be considered when interpreting the results due to time and computing constraints.

Single Human Participant First, collaboration with a human was done with a single participant, the researcher. This is not ideal, as I have a more profound knowledge of the system than an external participant. Furthermore, it would be better to have a sample of humans playing with the system to provide various and diverse collaborative feedback to the agent. A solution to this problem would be to use crowd workers to interact with the system.

Limited Number and Type of Tasks Used to Evaluate Performance We only tested our agents on two tasks: collecting dirt or wood. This is not ideal as it does not give us a holistic view of the different learning dynamics of the agent. Furthermore, it makes our claims weaker, as we were only able to validate our hypothesis on a tiny subset of tasks that Minecraft offers, thus reducing the statistical significance of our results. Furthermore, the tasks tested with our agents are not sufficiently diverse. We only consider collecting resources tasks. In addition, those tasks are atomic in Minecraft, as they are the most straightforward set of tasks in Minecraft. Therefore, we did not study the composability aspect of the original Voyager paper. However, this choice is motivated by the current state of open-source LLM. Open-source LLM in the Voyager framework fails at the simplest task (collecting dirt). Thus, we managed at least to improve the performance of this with our method.

Impact of Contradicting Beliefs Our study lacks a study on the impact of contradicting beliefs. The insights of the collaboration could contradict each other. Thus, beliefs could negatively influence each other. This problem is further exacerbated if we have multiple collaborators. For example, “try to make your code as simple as possible” and “make sure your code covers as many edge cases as possible” are valuable beliefs. In our work, we provide all relevant insights to the LLM and let him decide which to apply. However, we can imagine that certain contradictory beliefs could negatively influence task performance and that the LLM does not know how to deal with them. It would be interesting to study this phenomenon in more detail in future work.

6.4. Future work

In the development of our Collaborative Voyager agent, several essential software engineering practices have been employed to ensure that the system is not only robust and effective in its current state but also designed to accommodate future advancements and challenges. These practices are crucial in facilitating future work and allowing the agent to adapt and evolve without necessitating significant

redesigns. We wanted to enable subsequent improvements without the need for fundamental changes to the agent itself.

- **Modularity and Flexibility:** The agent's architecture is designed to be highly modular. This modularity allows individual components of the agent, like memory modules, to be updated or replaced independently as advancements in AI are developed. It ensures that the system is adaptable and extensible, facilitating the incorporation of new features without extensive reworking.
- **Use of Feature Toggles:** We used feature toggles to manage the introduction of new functionalities and experimental features without affecting the core operation of the agent. This technique allows us to enable or disable features dynamically, making ablation studies easier.
- **Documentation:** We documented our new components, allowing future developers to understand our rational design and the inner workings of each component.

The original Voyager codebase is large, so we left it as intact as possible. We, therefore, only apply the discussed principles to the modifications we did to the original architecture.

The findings of this thesis open several promising avenues for further research into collaboration in Minecraft and similar embodied environments. To extend current knowledge and overcome the limitations identified in this study, future research can be directed in the following areas:

Removing Access to Internal Belief The current system allows the collaborator to access the agent's internal beliefs. This is not ideal as this is privileged information that is not typically accessible, for example, in human-human collaboration. We do not necessarily know the internal state of a partner when collaborating with him. A solution to this problem is to have two-way communication with the agent. Thus, it could ask us for help and give us the necessary information to help him find a solution. This would also allow the agent to communicate with its collaborator to understand the intent better. Due to time constraints, we were unable to implement this.

Training of Better Collaborator Our experiment showed that humans can provide good insight to induce weak agents. Our current setup relies on the human as a collaborator in a manner similar to a Wizard of Oz experiment. We have collected interactions with the human. Thus, it would theoretically be possible to train an AI to imitate a human collaborator, removing our Wizard of Oz setup. This would show that AI can help them autonomously, laying the groundwork for more complex socially aware multi-agent systems. This would also require studying "when to intervene," as this remains an open challenge in agent settings. Indeed, current research only evaluates this challenge in a conversation setting which does not require any action coordination [38]

Dynamics in Larger Group Size of Collaborator The agents only have one collaborator in our research. Further work could analyze the impact on the number of participants and agent performance. For example, one can imagine that having a single collaborator allows for deeper collaboration; this is in comparison with having a multitude of collaborators, where you can see a diversity of points of view. This mimics the human learning paradigm; for example, is it better to learn from a private tutor who knows their student deeply or from an ensemble of teachers?

Adversarial Settings In our work, we assume a non-adversarial setting. However, in real life, bad actors can deceive agents. It would be interesting for our Collaborative Voyager not directly to trust its collaborator but have a trust system to ensure that a collaborator has good intentions. Furthermore, our setup is fully collaborative; it would be interesting to test a more complex group dynamic, such as a competitive setting, where the agents have different objectives and do not wish to share all their knowledge fully.

7

Conclusion

This thesis offers a comprehensive exploration of the potential for improving the problem-solving capabilities of open-source LLM-powered agents through collaboration, specifically within the dynamic and challenging environment of Minecraft. Indeed, current open-source LLMs fail to match their closed-source counterpart. Therefore, we offer collaboration as an alternative learning method that would allow us to close this gap. We covered the current literature, experimental validations, and the development of novel methodologies to address collaboration challenges to create more socially aware agents.

Our primary research objective was to determine whether and how collaborative interactions between agents could enhance their ability to perform tasks. The results obtained from a series of designed experiments provide compelling evidence that collaboration significantly enhances the capabilities of LLM-powered agents. These findings emphasise the importance of social interaction in the cognitive development of AI systems, resonating with the social intelligence hypothesis, which propose that intelligence does not arise in isolation but through complex social interactions.

Through the introduction of our Collaborative Voyager architecture, we demonstrated that agents could improve their task performance and adaptability by leveraging real-time interactions with peers. This architecture allowed the agent to dynamically adjust its strategy based on the insights of collaborators. Furthermore, those new insights are saved, enabling our weak agents to become stronger over time. By enabling agents to conceptualize and internalize the perspectives of their collaborators, through the use of the BigToM template, we witnessed an improvement in how these agents navigated the Minecraft environment, suggesting that the ability to “think” from another’s point of view can be crucial for complex problem-solving tasks.

However, this research also illuminated several limitations and challenges. The number and type of tasks evaluated limits the effectiveness and the study of collaboration. Additionally, while collaborative efforts did improve agent performance in controlled settings, scaling these results to more long-term horizon problems remains a challenge. Furthermore, the thesis outlines several avenues for further research. Expanding the diversity and number of collaborative interactions, extending the complexity of tasks, and training more socially aware LLM could provide deeper insights into the scalable application of collaborative AI. Furthermore, exploring adversarial settings where agents must collaborate under competitive pressures could reveal new dimensions of agents’ behaviors and strategy developments.

In conclusion, this thesis contributes to a deeper understanding of the potential of AI systems to develop and operate within socially interactive frameworks. The implications of this research extend beyond the realms of Minecraft, offering valuable insights for the development of AI applications in areas such as robotics, where collaboration is key. We want AI systems that are not only intelligent and adaptive but also capable of working alongside humans to tackle the complex challenges of the real world.

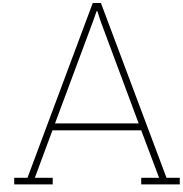
References

- [1] Guanzhi Wang et al. *Voyager: An Open-Ended Embodied Agent with Large Language Models*. arXiv:2305.16291 [cs]. May 2023. DOI: 10.48550/arXiv.2305.16291. URL: <http://arxiv.org/abs/2305.16291> (visited on 09/25/2023).
- [2] Zihao Wang et al. *JARVIS-1: Open-World Multi-task Agents with Memory-Augmented Multimodal Language Models*. arXiv:2311.05997 [cs]. Nov. 2023. DOI: 10.48550/arXiv.2311.05997. URL: <http://arxiv.org/abs/2311.05997> (visited on 12/04/2023).
- [3] *Social Intelligence Hypothesis - an overview* | ScienceDirect Topics. URL: <https://www.sciencedirect.com/topics/psychology/social-intelligence-hypothesis> (visited on 04/16/2024).
- [4] Alex Wilf et al. *Think Twice: Perspective-Taking Improves Large Language Models' Theory-of-Mind Capabilities*. arXiv:2311.10227 [cs]. Nov. 2023. DOI: 10.48550/arXiv.2311.10227. URL: <http://arxiv.org/abs/2311.10227> (visited on 04/17/2024).
- [5] Kanishk Gandhi et al. *Understanding Social Reasoning in Language Models with Language Models*. arXiv:2306.15448 [cs]. Dec. 2023. DOI: 10.48550/arXiv.2306.15448. URL: <http://arxiv.org/abs/2306.15448> (visited on 04/17/2024).
- [6] Melisa Moore. "Vygotsky's Cognitive Development Theory". en. In: *Encyclopedia of Child Behavior and Development*. Ed. by Sam Goldstein and Jack A. Naglieri. Boston, MA: Springer US, 2011, pp. 1549–1550. ISBN: 978-0-387-79061-9. DOI: 10.1007/978-0-387-79061-9_3054. URL: https://doi.org/10.1007/978-0-387-79061-9_3054 (visited on 04/16/2024).
- [7] Alyssa Hughes. *Presenting the winners of the Project Malmo Collaborative AI Challenge*. en-US. June 2017. URL: <https://www.microsoft.com/en-us/research/blog/malmo-collaborative-ai-challenge-winners/> (visited on 12/04/2023).
- [8] Anjali Narayan-Chen, Prashant Jayannavar, and Julia Hockenmaier. "Collaborative Dialogue in Minecraft". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by Anna Korhonen, David Traum, and Lluís Màrquez. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 5405–5415. DOI: 10.18653/v1/P19-1537. URL: <https://aclanthology.org/P19-1537> (visited on 12/04/2023).
- [9] Danielle D. Potts, Kate MacFarlane, and Lynne Hall. "Support Rather Than Assault – Cooperative Agents in Minecraft". In: BCS Learning & Development, July 2021, pp. 133–138. DOI: 10.14236/ewic/HCI2021.13. URL: <https://www.scienceopen.com/hosted-document?doi=10.14236/ewic/HCI2021.13> (visited on 12/04/2023).
- [10] Linxi Fan et al. *MineDojo Internet Knowledge Base (Wiki)*. June 2022. DOI: 10.5281/ZENODO.6640448. URL: <https://zenodo.org/record/6640448> (visited on 05/13/2024).
- [11] Stephanie Milani et al. *BEDD: The MineRL BASALT Evaluation and Demonstrations Dataset for Training and Benchmarking Agents that Solve Fuzzy Tasks*. arXiv:2312.02405 [cs]. Dec. 2023. DOI: 10.48550/arXiv.2312.02405. URL: <http://arxiv.org/abs/2312.02405> (visited on 02/19/2024).
- [12] Bowen Baker et al. *Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos*. arXiv:2206.11795 [cs]. June 2022. DOI: 10.48550/arXiv.2206.11795. URL: <http://arxiv.org/abs/2206.11795> (visited on 12/06/2023).
- [13] Ashish Vaswani et al. *Attention Is All You Need*. arXiv:1706.03762 [cs]. Aug. 2023. DOI: 10.48550/arXiv.1706.03762. URL: <http://arxiv.org/abs/1706.03762> (visited on 04/16/2024).
- [14] Tom B. Brown et al. *Language Models are Few-Shot Learners*. arXiv:2005.14165 [cs]. July 2020. DOI: 10.48550/arXiv.2005.14165. URL: <http://arxiv.org/abs/2005.14165> (visited on 06/15/2023).

- [15] Harsha Nori et al. *Can Generalist Foundation Models Outcompete Special-Purpose Tuning? Case Study in Medicine*. arXiv:2311.16452 [cs]. Nov. 2023. DOI: 10.48550/arXiv.2311.16452. URL: <http://arxiv.org/abs/2311.16452> (visited on 12/04/2023).
- [16] Jenish Maharjan et al. *OpenMedLM: Prompt engineering can out-perform fine-tuning in medical question-answering with open-source large language models*. arXiv:2402.19371 [cs]. Feb. 2024. DOI: 10.48550/arXiv.2402.19371. URL: <http://arxiv.org/abs/2402.19371> (visited on 04/16/2024).
- [17] Noah Shinn et al. *Reflexion: Language Agents with Verbal Reinforcement Learning*. arXiv:2303.11366 [cs]. Oct. 2023. DOI: 10.48550/arXiv.2303.11366. URL: <http://arxiv.org/abs/2303.11366> (visited on 01/22/2024).
- [18] Jason Wei et al. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. arXiv:2201.11903 [cs]. Jan. 2023. DOI: 10.48550/arXiv.2201.11903. URL: <http://arxiv.org/abs/2201.11903> (visited on 04/17/2024).
- [19] Shunyu Yao et al. *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*. arXiv:2305.10601 [cs]. Dec. 2023. DOI: 10.48550/arXiv.2305.10601. URL: <http://arxiv.org/abs/2305.10601> (visited on 04/17/2024).
- [20] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. arXiv:2005.11401 [cs]. Apr. 2021. DOI: 10.48550/arXiv.2005.11401. URL: <http://arxiv.org/abs/2005.11401> (visited on 04/17/2024).
- [21] Jeremy Howard and Sebastian Ruder. *Universal Language Model Fine-tuning for Text Classification*. arXiv:1801.06146 [cs, stat]. May 2018. DOI: 10.48550/arXiv.1801.06146. URL: <http://arxiv.org/abs/1801.06146> (visited on 04/17/2024).
- [22] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. arXiv:2106.09685 [cs]. Oct. 2021. DOI: 10.48550/arXiv.2106.09685. URL: <http://arxiv.org/abs/2106.09685> (visited on 04/17/2024).
- [23] Angels Balaguer et al. *RAG vs Fine-tuning: Pipelines, Tradeoffs, and a Case Study on Agriculture*. arXiv:2401.08406 [cs]. Jan. 2024. DOI: 10.48550/arXiv.2401.08406. URL: <http://arxiv.org/abs/2401.08406> (visited on 04/17/2024).
- [24] Jintian Zhang, Xin Xu, and Shumin Deng. *Exploring Collaboration Mechanisms for LLM Agents: A Social Psychology View*. arXiv:2310.02124 [cs]. Oct. 2023. DOI: 10.48550/arXiv.2310.02124. URL: <http://arxiv.org/abs/2310.02124> (visited on 01/05/2024).
- [25] *NeurIPS 2023 Tutorial*. en-GB. URL: <https://sites.google.com/view/neurips2023law> (visited on 05/15/2024).
- [26] Lei Wang et al. *A Survey on Large Language Model based Autonomous Agents*. arXiv:2308.11432 [cs]. Sept. 2023. DOI: 10.48550/arXiv.2308.11432. URL: <http://arxiv.org/abs/2308.11432> (visited on 10/25/2023).
- [27] Jiannan Xiang et al. *Language Models Meet World Models: Embodied Experiences Enhance Language Models*. en. arXiv:2305.10626 [cs]. Oct. 2023. URL: <http://arxiv.org/abs/2305.10626> (visited on 05/15/2024).
- [28] *How the Body Shapes the Way We Think*. en-US. URL: <https://mitpress.mit.edu/9780262537421/how-the-body-shapes-the-way-we-think/> (visited on 04/17/2024).
- [29] Lucy A Suchman. "PLANS AND SITUATED ACTIONS." en. In: (). URL: http://bitsavers.trailing-edge.com/pdf/xerox/parc/techReports/ISL-6_Plans_and_Situated_Actions.pdf.
- [30] Tongtong Wu et al. *Continual Learning for Large Language Models: A Survey*. arXiv:2402.01364 [cs]. Feb. 2024. DOI: 10.48550/arXiv.2402.01364. URL: <http://arxiv.org/abs/2402.01364> (visited on 05/15/2024).
- [31] Çağatay Yıldız et al. *Investigating Continual Pretraining in Large Language Models: Insights and Implications*. arXiv:2402.17400 [cs]. Feb. 2024. DOI: 10.48550/arXiv.2402.17400. URL: <http://arxiv.org/abs/2402.17400> (visited on 05/15/2024).

- [32] Cong Lu et al. *Synthetic Experience Replay*. arXiv:2303.06614 [cs, stat]. Oct. 2023. DOI: 10.48550/arXiv.2303.06614. URL: <http://arxiv.org/abs/2303.06614> (visited on 05/15/2024).
- [33] Xizhou Zhu et al. *Ghost in the Minecraft: Generally Capable Agents for Open-World Environments via Large Language Models with Text-based Knowledge and Memory*. arXiv:2305.17144 [cs]. June 2023. DOI: 10.48550/arXiv.2305.17144. URL: <http://arxiv.org/abs/2305.17144> (visited on 12/06/2023).
- [34] Haoqi Yuan et al. *Plan4MC: Skill Reinforcement Learning and Planning for Open-World Minecraft Tasks*. arXiv:2303.16563 [cs]. Mar. 2023. DOI: 10.48550/arXiv.2303.16563. URL: <http://arxiv.org/abs/2303.16563> (visited on 12/04/2023).
- [35] Kolby Nottingham et al. *Do Embodied Agents Dream of Pixelated Sheep: Embodied Decision Making using Language Guided World Modelling*. arXiv:2301.12050 [cs]. Apr. 2023. DOI: 10.48550/arXiv.2301.12050. URL: <http://arxiv.org/abs/2301.12050> (visited on 12/04/2023).
- [36] Sirui Hong et al. *MetaGPT: Meta Programming for Multi-Agent Collaborative Framework*. arXiv:2308.00352 [cs]. Aug. 2023. DOI: 10.48550/arXiv.2308.00352. URL: <http://arxiv.org/abs/2308.00352> (visited on 10/12/2023).
- [37] Yashar Talebirad and Amirhossein Nadiri. *Multi-Agent Collaboration: Harnessing the Power of Intelligent LLM Agents*. arXiv:2306.03314 [cs]. June 2023. DOI: 10.48550/arXiv.2306.03314. URL: <http://arxiv.org/abs/2306.03314> (visited on 04/17/2024).
- [38] Jimmy Wei et al. *Multi-Party Chat: Conversational Agents in Group Settings with Humans and Models*. arXiv:2304.13835 [cs]. June 2023. DOI: 10.48550/arXiv.2304.13835. URL: <http://arxiv.org/abs/2304.13835> (visited on 01/05/2024).
- [39] Xudong Guo et al. *Embodied LLM Agents Learn to Cooperate in Organized Teams*. arXiv:2403.12482 [cs]. Mar. 2024. DOI: 10.48550/arXiv.2403.12482. URL: <http://arxiv.org/abs/2403.12482> (visited on 05/15/2024).
- [40] *Open-Orca/Mistral-7B-OpenOrca · Hugging Face*. Oct. 2023. URL: <https://huggingface.co/Open-Orca/Mistral-7B-OpenOrca> (visited on 04/22/2024).
- [41] *mistralai/Mistral-7B-v0.1 · Hugging Face*. URL: <https://huggingface.co/mistralai/Mistral-7B-v0.1> (visited on 04/22/2024).
- [42] Anand S. Rao and Michael P. Georgeff. "Modeling rational agents within a BDI-architecture". In: *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*. KR'91. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Apr. 1991, pp. 473–484. ISBN: 978-1-55860-165-9. (Visited on 05/15/2024).
- [43] *GPT-4. en-US*. URL: <https://openai.com/gpt-4> (visited on 04/22/2024).
- [44] *mistralai/Mixtral-8x7B-Instruct-v0.1 · Hugging Face*. URL: <https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1> (visited on 04/22/2024).
- [45] *swen128/Voyager_checkpoint: Trained checkpoint and tracing log for Voyager*. URL: https://github.com/swen128/Voyager_checkpoint (visited on 04/17/2024).
- [46] *PrismarineJS/mineflayer*. original-date: 2011-01-23T09:41:55Z. May 2024. URL: <https://github.com/PrismarineJS/mineflayer> (visited on 05/13/2024).
- [47] *Project Malmo. en-US*. URL: <https://www.microsoft.com/en-us/research/project/project-malmo/> (visited on 04/22/2024).
- [48] Yijun Yang et al. *Embodied Multi-Modal Agent trained by an LLM from a Parallel TextWorld*. arXiv:2311.16714 [cs]. Mar. 2024. DOI: 10.48550/arXiv.2311.16714. URL: <http://arxiv.org/abs/2311.16714> (visited on 04/22/2024).
- [49] Weihao Tan et al. "True Knowledge Comes from Practice: Aligning Large Language Models with Embodied Environments via Reinforcement Learning". en. In: Oct. 2023. URL: <https://openreview.net/forum?id=hILVmJ4Uvu> (visited on 04/22/2024).
- [50] Karthik Valmeekam et al. "Large Language Models Still Can't Plan (A Benchmark for LLMs on Planning and Reasoning about Change)". en. In: (). URL: <https://openreview.net/pdf?id=wUU-7XTL5X0>.

-
- [51] Jia-Yu Yao et al. *LLM Lies: Hallucinations are not Bugs, but Features as Adversarial Examples*. arXiv:2310.01469 [cs]. Oct. 2023. DOI: 10.48550/arXiv.2310.01469. URL: <http://arxiv.org/abs/2310.01469> (visited on 04/16/2024).
- [52] Zhangyue Yin et al. “Do Large Language Models Know What They Don’t Know?” In: *Findings of the Association for Computational Linguistics: ACL 2023*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 8653–8665. DOI: 10.18653/v1/2023.findings-acl.551. URL: <https://aclanthology.org/2023.findings-acl.551> (visited on 04/22/2024).
- [53] Toufique Ahmed et al. *Studying LLM Performance on Closed- and Open-source Data*. arXiv:2402.15100 [cs]. Feb. 2024. DOI: 10.48550/arXiv.2402.15100. URL: <http://arxiv.org/abs/2402.15100> (visited on 04/16/2024).



Logbook of agent failures

This appendix presents the key findings from developing and testing the Voyager and Collaborative Voyager systems. We analyze failure cases from our experiments that have informed our understanding of these learning models. These failures, as revealing as successes, offer insights into the limitations and potential enhancements of our AI agents. By documenting these missteps, we aim to contribute to AI research, improving the resilience and flexibility of language model agents in complex environments.

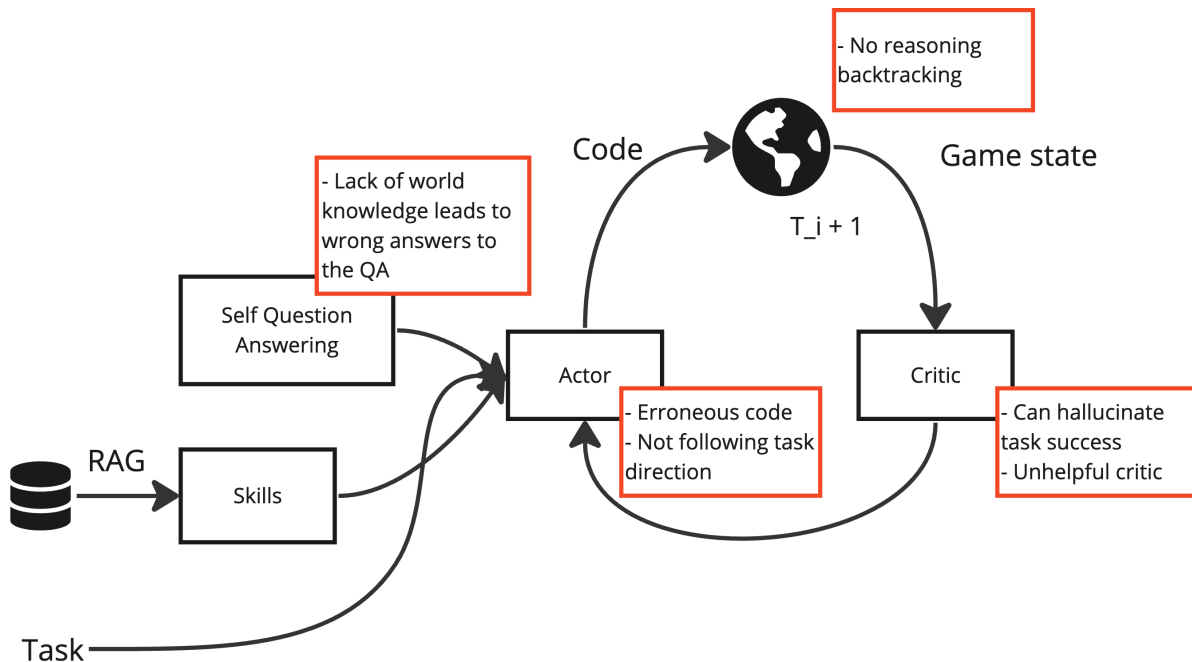


Figure 24: Voyager architecture shortcomings

Figure 24 illustrates several common errors encountered by the original Voyager agent, reflecting fundamental limitations in its operational logic and decision-making process:

1. **Lack of World Knowledge Leads to Wrong Answers to QA:** This error arises when the agent does not possess sufficient contextual information or specific domain knowledge, leading to incorrect or irrelevant answers during question-answering tasks. This is especially a problem in the Minecraft domain.
2. **No Reasoning Backtracking:** The agent's inability to re-evaluate or reverse its previous reasoning steps when faced with new information or errors leads to persistent mistakes. For example, when an agent thinks it needs a shovel to get dirt, it cannot backtrack this insight even if it realizes it is wrong.

3. **Erroneous Code:** The actor model often produces erroneous code that is sometimes non-sensical, for example, by adding hallucinate intermediary steps.
4. **Not Following Task Direction:** This error occurs when the agent misinterprets or disregards the instructions, leading to actions that do not align with the set objectives. For example, instead of doing the given task, the agent will decide to do another random task.
5. **Can Hallucinate Task Success:** The agent may incorrectly conclude that a task has been completed successfully or misinterpret the state of the environment. This is detailed more in our section 5.1
6. **Unhelpful Critic:** When the feedback mechanism does not provide constructive or accurate guidance, the agent's learning and adjustment processes are impeded. The critic module is also plagued by random hallucinations, making the guidance process unreliable.

A.1. Examples of failures:

Hallucination

(a) Functions

This is not a function... bad usage of the API.

Solution: We tried fine-tuning with Mineflayer API, did not help qualitatively. An alternative would be RAG.

(b) Lack of Minecraft understanding in QA

Grass and dirt are equivalent. For example, in the context of Minecraft, an open-source LLM might erroneously interpret grass and dirt as two different concepts when they are essentially the same. This reflects a fundamental misunderstanding of the game's elements. *Solution:* We tried fine-tuning with Minecraft Wiki did not help qualitatively. An alternative would be RAG.

Previous erroneous context makes the LLM weaker

- Bad previous context
 - Example: trying to tell the bot to use hand to pick up dirt. However, because the previous code used a shovel, the agent cannot reset itself to use a hand.
 - *Solution:* allow interaction agent to clean context and restart task with latest reflections.
 - Literature: similar examples in literature are backtracking a wrong path in a reflection tree [19].

A.1.1. Lack of instruction following / internal consistency

```

1 Agent0 **Action Agent ai message**
2 Explain: The task is to get a dirt or grass block in Minecraft. The bot has provided
   information about its surroundings, including the biome, time of day, and nearby blocks
   and entities. However, the code does not complete the task as it is missing steps.
   Specifically, it does not mine the dirt or grass block using the appropriate tool (hand)
   or check if the inventory has the required item before attempting to mine.
3
4 Plan: To complete the task, we will first check if the bot has the required items in its
   inventory. If it does not have a diamond pickaxe or an iron pickaxe, we will craft one
   using the appropriate materials. Then, we will use the pickaxe to mine the nearest dirt
   or grass block.
5
6 Code:
7
8 ```jsx
9 // helper functions (only if needed, try to avoid them)
10 ...
11
12 // main function after the helper functions
13 async function getDirtOrGrassBlock(bot) {
14   // Check inventory for required items
15   const diamondPickaxe = bot.inventory.find((item) => item.name === "diamond_pickaxe");
16   const ironPickaxe = bot.inventory.find((item) => item.name === "iron_pickaxe");
17
18   if (!diamondPickaxe && !ironPickaxe) {

```

```

19 // Craft diamond or iron pickaxe if not found in inventory
20 const diamondOre = bot.inventory.find((item) => item.name === "diamond_ore");
21 const ironOre = bot.inventory.find((item) => item.name === "iron_ore");
22 if (diamondOre || ironOre) {
23   await bot.craftItem(diamondOre, ironOre);
24 } else {
25   console.error("Insufficient items in inventory to craft pickaxe.");
26   return;
27 }
28 }
29
30 // Mine nearest dirt or grass block
31 const targetBlock = bot.findBlocks(dirt, grass).find((block) => {
32   const distance = Math.sqrt(Math.abs(bot.position.x - block.x));
33   return distance < 32;
34 });
35
36 if (targetBlock) {
37   await bot.mineBlock(targetBlock);
38 } else {
39   console.error("Could not find any dirt or grass blocks nearby.");
40 }
41 }
42 ...

```

Listing A.1: Lack of instruction following

This snippets demonstrate random explore until you find a dirt block. However, the dirt block is already next to the player. Furthermore, there is an infinite loop because the generated AI code is buggy. Overcomplicated code makes failure easier.

A.1.2. Hallucinate answers to QA

```

1 Agent0 *****Action Agent ai message*****
2 Explain: The task is to get a dirt or grass block in Minecraft. The code provides the
   necessary steps to achieve this task, including finding a nearby deepslate_iron_ore block
   and mining 5 of them using a diamond pickaxe, and then exploring the area to find a
   deepslate_lapis_ore block and mining it with an iron pickaxe.

```

Listing A.2: QA hallucination

Example of hallucination. There is no link between finding a deepslate iron ore block and getting dirt.

A.1.3. Example of hallucinated success

```

1 *****Critic Agent ai message*****
2 {
3   "reasoning": "You want to get a dirt block. Since you are in the savanna biome, there is
4     grass and grass_block nearby. You can successfully dig it up using your bare hands.",
5   "success": true,
6   "critique": ""
7 }

```

Listing A.3: Critic Agent AI message

The agent did not accomplish the task but hallucinated the fact that he did

A.1.4. Comparison between open-source LLM and GPT

The benefit of GPT is that it is straight to the point, with no extra necessary logic, less hallucination, etc. Compared to the open-source LLM, it hallucinates a random position in the world where there is dirt (-17, 71, 134). This is non-sensical.

Listing A.4: Open source LLM answer

```

1 Agent0 *****Action Agent ai message*****
2 Explain: To get a dirt block in Minecraft,
  you can use the `bot.dig` method
  without any tools. The method takes
  one argument, the block type, and
  returns the block if it's
  successfully mined, or `null`
  otherwise. In this case, we check if
  the player has the necessary items (
  dirt) in their inventory, and then
  mine for a dirt block at coordinates
  (-17, 71, 134). If the block is not
  found, we throw an error message.
3
4 Plan:
5
6 1. Check if the player has the necessary
  items in their inventory.
7 2. Mine for a dirt block at coordinates
  (-17, 71, 134) using `bot.dig`.
8 3. Add the dirt block to the player's
  inventory if it's successfully mined.
9
10 Code:
11 ```javascript
12 async function getDirtBlock(bot) {
13   // Check if player has necessary items
14   if (!bot.inventory.contains('dirt')) {
15     await collectNecessaryItems(bot);
16   }
17
18   // Mine for dirt block
19   const dirt = await bot.dig('dirt',
20     -17, 71, 134);
21   if (!dirt) {
22     throw 'Failed to mine dirt block';
23   }
24
25   // Add dirt block to inventory
26   bot.inventory.add('dirt');
27 }
28 Note: I've used collectNecessaryItems as a
  helper function, you can replace it
  with your own implementation or omit
  it if you don't need it.

```

Listing A.5: GPT4 answer

```

1 Plan:
2 1) Check the bot's inventory to see if it
  already has a dirt block. If it does,
  the
3 2) If the bot does not have a dirt block,
  find the position of a nearby dirt
  block.
4 3) Mine the dirt block using the `
  mineBlock` function.
5 4) Confirm that the dirt block has been
  added to the bot's inventory.
6
7 Code:
8 ```javascript
9 async function getOneDirtBlock(bot) {
10   // Step 1: Check if the bot already
  has a dirt block in its inventory
11   const dirtBlock = bot.inventory.
  findInventoryItem(mcData.
  itemsByName.dirt.id);
12   if (dirtBlock) {
13     bot.chat("I already have a dirt
  block.");
14     return;
15   }
16
17   // Step 2: Find the position of a
  nearby dirt block
18   const dirtPosition = bot.findBlock({
19     matching: mcData.blocksByName.dirt
  .id,
20     maxDistance: 32
21   });
22
23   // Step 3: Mine the dirt block
24   if (dirtPosition) {
25     await mineBlock(bot, "dirt", 1);
26     bot.chat("1 dirt block obtained.")
  ;
27   } else {
28     bot.chat("Could not find a dirt
  block nearby.");
29   }
30 }

```

Figure 25: Comparison of open-source vs closed-source LLM quality

B

Agent prompts

This appendix offers a collection of prompts for the Collaborative Voyager agent, which powers our agent to facilitate AI interaction and collaboration. This catalog serves as a reference for replicating our experiments. The showcased prompts also provide insights into design considerations and communication strategies for socially aware AI systems.

Create Perception Beliefs

Listing B.1: Create a list of beliefs based on sensory data

```
1 Prompt: You are a Minecraft agent. Create a list of beliefs from your sensory perception. I will provide data from your sensor, and you will need to make a list of beliefs.
```

Check Context Based on Beliefs

Listing B.2: Evaluate if the AI-generated context is correct or erroneous

```
1 Prompt: You are a Minecraft agent. You have a generated context created by an AI. The goal is to evaluate if the context is correct or erroneous. You receive a list of beliefs in the form of questions and answers that are assumed to be accurate. Flag the context as incorrect if you find a contradiction with the beliefs. Respond with True or False to indicate if the context is correct.
```

Transform Message into QA

Listing B.3: Transform a received message into a QA format

```
1 Prompt: You are a Minecraft agent. Your goal is to transform a message received in the chat into a QA (question and answer). I will provide you with examples of QA and one message that needs to be transformed into a QA.
```

Update Context if It Is Wrong

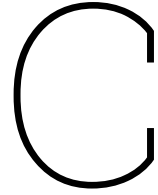
Listing B.4: Provide an answer based on a set of given beliefs

```
1 Prompt: You are a Minecraft agent. Your goal is to answer the provided question based on a set of beliefs. You receive a list of beliefs in the form of questions and answers that are assumed to be correct. Answer the question at the end based on the provided beliefs. Start your answer with "Answer: ". If you don't know the answer, respond with "Answer: Unknown".
```

Filtering after RAG Retrieval

Listing B.5: Output a list of selected candidate QAs that are most relevant to the task

```
1 Prompt: Your task is to output a list of selected candidate QAs most relevant to the task. Format this list as a LaTeX appendix.
```



Agents execution time

This appendix provides time measurements captured during the execution of our experiments. Each table is associated with the experiments described in the chapter 5: experiment. All times are in seconds; the setup is the same as the experiments. N/A signify that the model did not achieve any success during the run. Success is once again defined as TP and FN. The most noticeable trends were explained in the main body, but we left the rest of the measurements for completeness.

Model	Average Time to Success	Average Time per Round
Voyager GPT	132	132
Voyager Mistral	84	219
Voyager Mixtral	66	216
Collaborative Voyager GPT	144	144
Collaborative Voyager Mistral	N/A	252
Collaborative Voyager Mixtral	126	227

Note: All times are specified in seconds.

Table 1: Experiment 0: Comparison of Voyager and Collaborative Voyager execution time with a selection of LLM

Model	Average Time to Success	Average Time per Round
Voyager Mistral	84	219
Voyager finetuned v0	N/A	308
Voyager finetuned v1	N/A	600
Collaborative Voyager Mistral	N/A	252
Collaborative Voyager finetuned v0	NA	263
Collaborative Voyager finetuned v1	NA	600

Note: All times are specified in seconds.

Table 2: Experiment 1: Comparison of the execution time of Voyager and Collaborative Voyager after fine-tuning

Model	Average Time to Success	Average Time per Round
Collaborative Voyager	126	227
2 x Collaborative Voyager	89	88
Collaborative Voyager + Human	176	209

Note: All times are specified in seconds.

Table 3: Experiment 2: Comparison of Collaborative Voyager’s execution time in the presence of a collaborator

Model	Average Time to Success	Average Time per Round
Collaborative Voyager	126	227
Collaborative Voyager + Human	176	209
Collaborative Voyager with distilled memory	125	179

Note: All times are specified in seconds.

Table 4: Experiment 3: Comparison of Collaborative Voyager’s execution time with the distilled knowledge memory enabled

Model	Average Time to Success	Average Time per Round
Collaborative Voyager	119	247
Collaborative Voyager with distilled memory (trained on dirt task)	149	284

Note: All times are specified in seconds.

Table 5: Experiment 4: Comparison the Collaborative Voyager’s execution time on out-of-distribution tasks (collecting wood)