WHY SOLVING MULTI-AGENT PATH FINDING WITH LARGE LANGUAGE MODELS HAS NOT SUCCEEDED YET

Anonymous authors
Paper under double-blind review

ABSTRACT

With the explosive influence caused by the success of large language models (LLM), there has been an extensive amount of recent work showing that foundation models can be used to solve a large variety of tasks. However, there is very limited work that shares insights on multi-agent planning. Multi-agent planning is different from other domains by combining the difficulty of multi-agent coordination and planning, and making it hard to leverage external tools to facilitate the reasoning needed. In this paper, we focus on the problem of multi-agent path finding (MAPF), which is also known as multi-robot route planning, and study the performance of solving MAPF with LLMs. We first show the motivating success of single-agent planning and multi-agent pathfinding in an empty room map without obstacles, then the failure to plan on the harder room map and maze map of the standard MAPF benchmark. We present our position on why directly solving MAPF with LLMs has not been successful yet, and we use various experiments to support our hypothesis. Based on our results, we discussed how researchers with different backgrounds could help with this problem from different perspectives.

1 INTRODUCTION

In the past year since ChatGPT came out, large language models (LLMs) have been shown to go beyond strictly language-related tasks like translation, and to be a powerful tool in all kinds of domains. Training with very rich and diverse datasets, LLMs incorporate a large variety of knowledge and do not require fine-tuning before generating good solutions in many real-world applications. Over time, recent studies have shown that combining LLMs with different ways of prompting can help solve problems that have some aspects of reasoning, including examples from logical problems Yao et al. (2022); Liu et al. (2023) to controlling a robot dog without finetuning Wang et al. (2023a).

Motivated by the success of LLMs, people are studying how well pre-trained models are in all different kinds of domains. Some recent works studied the performance of LLMs on multi-agent problems and showed LLM can also help multi-agent coordination Chen et al. (2023b); Agashe et al. (2023). However, they barely cover multi-agent route planning and did not look specifically into the difficulties in the domain. In this paper, we consider the problem of multi-agent path finding (MAPF), also known as multi-agent route planning. MAPF is the problem of moving a group of agents from their respective start locations to their goal locations without collisions. MAPF can be used directly to formulate real-world applications like warehouse management Sharon et al. (2015); Han & Yu (2020), swarm control Li et al. (2020), among others. In a typical warehouse scenario, around a thousand warehouse robots (agents) could be running simultaneously in one warehouse room (scenario), and each agent needs to plan its path, which could be as long as a hundred timesteps. Previous methods for MAPF can be mainly classified into 1) classic methods like heuristic search and SAT Sharon et al. (2015); Han & Yu (2020), and 2) learning-based approaches that mostly use reinforcement learning Sartoretti et al. (2019). MAPF is a unique problem in multi-agent coordination in that coordination is required within the planning, so it is highly challenging to facilitate the reasoning of coordination by using LLMs to generate useful high-level guidance combined with a low-level path planner to ensure the solution is valid, an approach that will be parallel to this taken in the setting of overcooked Agashe et al. (2023). On the other hand, path planning is one of the easiest parts of planning, and constraints only concern obstacles and moving

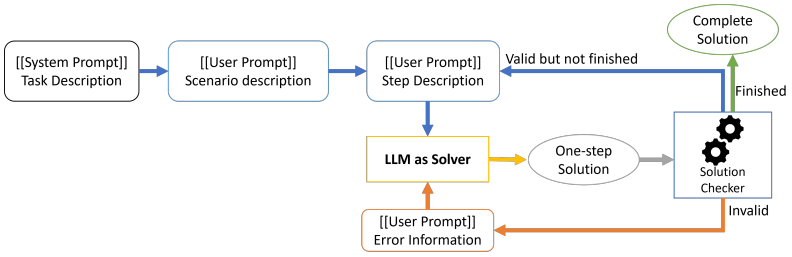

Figure 1: An illustration of our workflow.

continuity. Because of these unique challenges in the MAPF problem, it is unclear how good LLMs will be at solving MAPF.

In this paper, we propose to use an LLM directly to solve MAPF without any additional heuristic guidance from solvers or any additional training. We give the scenario information to the LLM, and let the LLM recommend the actions for each agent every single timestep. We use a high-level checker that checks collisions between agents and obstacles to ensure the solution is valid in each step, provides error messages to the LLM when violations occur, and iterates with the LLM until a valid solution is generated. We first show that single-agent pathfinding can be solvable and then evaluate the performance of LLMs on the empty map, the room map, and the maze map, which are classic maps from the standard MAPF benchmark Stern et al. (2019). We show that LLMs can solve MAPF problems when the scenario is relatively easy, and LLMs fail to generate any valid solution, no matter what the solution quality is, when the scenario becomes harder. While LLMs are evolving extremely fast, and current failures may not apply to the next generation of LLMs, we elaborate on the current failing scenarios and give our position on three aspects as a breakdown of what kind of capability is currently missing in the LLM workflow, namely 1) the capability to understand the scenario, 2) the context length limit, and 3) the reasoning capability. We incorporate a list of experiments featuring various prominent prompt design alternatives, such as image-based and text-only inputs, and with or without single-step observation information, among others. Finally, we provide a discussion on the challenges of using LLMs for MAPF in the real world, and discuss how different researchers could contribute to the problem from different directions. We hope our work can serve as a building block for future research in foundation models for MAPF.

2 LLMs for Multi-agent Path Finding

2.1 Preliminary

The multi-agent path finding (MAPF) problem is the problem of finding a set of collision-free paths for a set of agents in a known environment while minimizing their travel times. Specifically, in this paper, we consider the problem in Stern et al. (2019); Li et al. (2022), which is a four-connected grid map, where each agent is given a start cell and a goal cell. A scenario is defined as the combination of the description of the map, indicating which cells have obstacles, and the start cells and goal cells of each agent. At each timestep, an agent can move to an adjacent cell or stay in its current cell. A collision happens if two agents end up in the same cell at the same timestep. Each agent remains at its goal cell after it arrives until all agents arrive at their goals. The objective is to minimize the makespan of the solution, i.e., minimize the time when the last agent arrives at its goal location.

With the number of parameters of LLMs being exponentially larger than the degree of freedom in small reasoning problems, LLMs have the potential to solve some easy problems in reasoning with certain prompts Wei et al. (2022), and break down harder problems into smaller ones to get suboptimal solutions Yang et al. (2023). When applying to MAPF, we hope LLMs can be an alternative model to the current MAPF reinforcement learning-based models without any additional training.

2.2 Methods

In this paper, we focus on using LLMs to solve the MAPF problem directly. However, it is obvious that current ML models cannot be perfect solvers in their first trial, and we introduce a high-level

collision checker to ensure the plan generated by the LLM is valid. We inform the LLM about the mistake in the current solution, if any. Unlike some previous works Yao et al. (2022); Yang et al. (2023), our checker is not another LLM both because it is extremely easy and efficient to detect collisions by a rule-based detector in linear time, and also because the LLM fails to always correctly identify the collisions. A detailed comparison is provided in the appendix. In this checker, we not only check for agent-to-agent collisions but also check for any collision with the fixed obstacles. We do not provide any additional guidance on how the LLM should resolve this. By default, all the information, like coordinates, is provided in text, and we will later discuss the performance difference caused by different input formats.

Following the common practice of LLMs Kambhampati et al.; Chen et al. (2023b), we build our workflow shown in Fig. 1. As existing learning-based approaches Damani et al. (2021), we give LLMs stepwise local information and let the LLM choose the actions of agents step-by-step. This step-by-step (SBS) generation is different from the popular chain-of-thoughts idea Wei et al. (2022) used in LLMs by not introducing more intermediate reasoning processes in the generation. Instead, it breaks down the whole planning task into smaller single-step tasks, so the LLM does not need to be fully correct before we can use some results from it. We start by giving the LLM the system prompt to become a solver for the MAPF problem. This part is fixed within each map, and different in the map description part in different maps. Then, we start the user prompt by providing the scenario information, which includes where the obstacles are, as well as the start location and the goal locations of each agent. We found that providing LLMs with specific local information about viable actions for the immediate next step significantly helps them avoid collisions with static obstacles, so we provide a single-step observation (SSO) in the prompt by default. Then we start to read the output of the LLM, use the checker to determine whether the output of the LLM is valid, and either output the errors to let the LLM correct on the current step, or tell the LLM to move on to the next step with the information that is specifically for the next step. To address the total token limit, we leverage the fact that MAPF is a Markov decision process where each state is independent of its previous states, and we restart the prompt from scratch, i.e., treat the current agent locations as their starting point, whenever we got a rate limit error. We provide all our prompts in the appendix.

2.3 Experiment Results on MAPF Benchmark

2.3.1 General Experiment Settings

While we have introduced the high-level collision detector, we define a solution generation to be successful if it does not fail because of any of the following:

1. Fails to generate a plan whose number of steps is at most 3 times as the optimal plan.
2. Fails 5 consecutive times in a single step after we provided the first round of feedback, specifying whether the current solution is correct or not.

We do not consider any token limit error because we restart at each step to clean its travel history if the token limit is too long, and 5 times in a single step guarantees that the message length in one step is within the token limit. Each setting in the experiment is tested on 5 different scenarios in the standard MAPF benchmark Stern et al. (2019), each different in terms of the start and goal location combinations of the agents on a given map [1]. By default, we are using the GPT-4-1106-preview model, also known as the GPT-4-turbo model with temperature 0 and seed 42 [2].

2.3.2 Single-agent Path Finding Results

As discussed in previous workValmeekam et al. (2024a), while LLM might not know how to generate an optimal plan, our results show that it could generate some plan regardless of the optimality. As shown in Table. 1, we found that LLM could generate a plan after a few iterations in relatively easy scenarios. We observe that with the growing groud-truth shortest path length, the success rate

[1]The map names are directly from the standard MAPF benchmark [27], for example, "Maze-32-32-2" denotes the map is in the class of maze with size 32*32, and is the second map in the benchmark.

[2]We are testing using the API, where the behavior could be slightly different from the web version of GPT-4, and the results from the LLMs could differ given the stochasticity of LLMs.

Map	Dist	Success Rate (%)	
		OS	SBS
Room	[2, 8)	80	100
	[8, 16)	40	100
	[16, 32)	40	80
	[32, 100)	20	40
Maze	[2, 8)	20	80
	[8, 16)	0	20

Table 1: The success rate of solving single-agent path finding with GPT-4-turbo on ROOM-32-32-4 and MAZE maps with varying ground-truth shortest path distance (Dist).

Map	n	Success Rate (%)	
		OS	SBS
Empty	2	20	100
	4	0	100
	8	0	100
	16	0	60
Room	2	20	100
	4	0	80
	8	0	20
Maze	2	0	0

Table 2: The success rate of solving MAPF with GPT-4-turbo on EMPTY-8-8, ROOM-32-32-4 and MAZE-32-32-2 maps with varying numbers of agents (n).

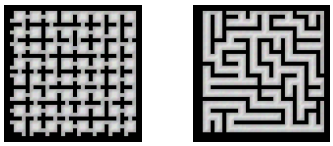

Figure 2: The ROOM-32-32-4 map (left) and the MAZE-32-32-2 map (right). The picture is vertically flipped to match the common knowledge that higher vertical positions indicate greater values.

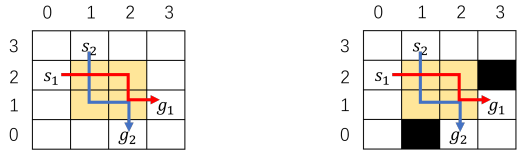

Figure 3: Two examples of symmetry breaking examples, where black denotes a fixed obstacle. In 3a, every pair of shortest path will collide with each other. In 3b, every pair of shortest paths will collide in cell (2, 1).

of LLM drops significantly, especially in the one-shot settings (OS). In the maze scenario, it is hard to find one solution even in the smallest case, and we will discuss the reasons in the next section.

We also found that the cause of failing in pathfinding is different from the cause of failing in general planning tasks where the generated plan is invalid, usually by breaking requisite constraints Valmeekam et al. (2023). In path planning, the only constraint is that one agent needs to move to an adjacent cell and cannot jump across the map. This is a very easy task for LLM to follow, and thus, in our experiments, we never see the LLM failing due to that constraint, even if the only instruction about this constraint is "in a 4-connected grid". Based on our results on single-agent path planning, we believe there is a possibility of generating a plan, regardless of optimality, for MAPF.

2.3.3 MULTI-AGENT PATH FINDING RESULTS

While it is unclear how well LLMs can solve MAPF problems, we start by showing that LLMs can be a promising solver in easy domains. We test the performance of LLMs on the EMPTY map from the MAPF benchmark Stern et al. (2019), which has no obstacles at all and only requires the agents to not collide with each other. The results are shown in Table. 2. We observe that for smaller-scale problems that has fewer agents, GPT-4 can successfully generate valid solutions. However, GPT-4 fails to generate valid solutions when more agents are on the same map.

It has been shown that a step-by-step generation is better in many reasoning-related tasks Wei et al. (2022). However, it is not well-studied whether a step-by-step generation is also helpful in the planning since a complete plan will also require to be given in a step-by-step (SBS) way, and the LLM models need to decide every single step anyway. Therefore, we study a variant that generates the whole plan in one shot (OS) with the help of chain-of-thought prompts and show the comparison in Table. 2. We observe that one-shot (OS) generation fails to generate solutions even in scenarios with 4 agents. This is because the whole plan is quite long, and the total number of possible failures is the product of possible failure at each step, making it inefficient to find a valid solution.

Motivated by the success on the EMPTY map, we tested the same workflow on the ROOM-32-32-1 map and the MAZE-32-32-2 map, which is shown in Fig. 2. The results are shown in Table. 2. In

Step 1:
Agent 1:
Current: (0,2)
Goal: (3,1)
Move: Right to (1,2)

Agent 2:
Current: (1,3)
Goal: (2,0)
Move: Down to (1,2)

After the move, the coordinates of the agents are:
Agent 1: (1,2)
Agent 2: (1,2)

There is a collision between Agent 1 and 2. I will regenerate another solution immediately.

Step 1 (Regenerated):
Agent 1:
Current: (0,2)
Goal: (3,1)
Move: Down to (0,1)

Agent 2:
Current: (1,3)
Goal: (2,0)
Move: Left to (0,3)

After the move, the coordinates of the agents are:
Agent 1: (0,1)
Agent 2: (0,3)

There is no collision. Here is the validated solution:

Agent 1: (0,1)
Agent 2: (0,3)

Figure 4: The output from GPT-4-turbo for the symmetry breaking example in Fig. 3b.

this case, we found that LLMs fail to generate solutions when the number of agents is already quite small, and that LLMs completely fail on the MAZE-32-32-map even with only 2 agents.

3 CAUSE OF FAILURES

Because LLMs are evolving rapidly, it is unclear how long LLMs will still fail. Although our methods do not combine any external tools like low-level planning Agashe et al. (2023), it is important for us to break down the failures and identify the cause of the failures. We observe that LLMs fail only once with 16 agents in the empty map because of consecutive failures in a single step, and all other times, they fail because they use many more steps than they should. More specifically, 77% of the failures occurred because the LLM agents began to oscillate in a specific area of the map, while the remaining failures were due to excessively long detours.

Based on our observations, we break down the reasons into three deeper parts from the perspective of natural language models: limitation from the model, understanding, and reasoning. For simplicity, we first discuss the lack of reasoning capability, which is common in many tasks.

3.1 REASONING CAPABILITY

In this paper, we focus on optimizing the makespan of the MAPF solution, and making the agents prefer to wait rather than detour in small scenarios that do not lead to endless waiting. We calculate the average ratio of makespan to the maximum length among the single agent shortest paths (a standard normalization practice in MAPF as this provides a valid lower bound on the optimal MAPF solution), and in scenarios that are successfully solved, this average is 1.5. This means that LLMs can mostly succeed in scenarios that do not need a lot of waiting, and in most steps, they only need to go in the two directions aligned with the goal's direction. LLM fails in other scenarios because they need path finding in complex environments. A simple example of the failure of path finding is shown in Fig. 3, and we provide the output of the first step in Fig. 4. While waiting for one step can clearly lead to the optimal solution that has a makespan of 5, the LLM chose to move agent 2 to the

Model	n	Success Rate (%)		Avg. Iterations	
		GO	GO+SSO	GO	GO+SSO
GPT-4-8K	2	80	100	2.7	1.6
	4	20	60	3.0	2.3
	8	0	0	N/A	N/A
GPT-4-128K	2	100	100	2.1	1.2
	4	60	80	2.7	1.4
	8	0	20	N/A	2.4

Table 3: The success rates and average iterations per step used until proceed to next step in success scenarios for GPT-4 and GPT-4 Turbo, whose token limits are 8K and 128K respectively, on the room-32-32-4 map with different number of agents(n).

Checker Type	Success Rate (%)
Rule-based	0
Human	40

Table 4: The success rate of o1-preview (OpenAI) with different checker type on 8 agent scenarios ($n = 8$) with GO+SSO in room-32-32-4 map.

left and resulting in a total makespan of 6. This example shows that LLM does not understand what makespan is and how to optimize for makespan.

On the other hand, when we look at the average number of iterations for prompts with both global obstacle observation in the first user prompt and single-step obstacle observations(GO+SSO) in Table. 3, it is still not very close to 1 which means no iterations at all. This is because even if we have explicitly let the LLM list all the coordinates of agents, it does not have the capability to check the answers itself without the use of external tools. More specifically, LLMs are guaranteed to tell whether a list of tuples of numbers has any duplicates. We also observe that the GPT-4-8k model takes more iterations than the latest GPT-4-turbo model. This could either contribute to forgetting earlier information or to the improved capacity of the new model.

This lack of reasoning capabilities is usually solved with tool use in other domains, but MAPF itself is hard because MAPF requires the capability of path finding in a complex map and avoiding collisions between all pairs of agents strategically and efficiently. The coordination between agents is required in each step, and each step only. If we let the tool, in this case, a single-agent planner like A*, include collision avoidance, the problem is already solved as CBS Sharon et al. (2015), where LLM did no help. On the other hand, the tools must know how collisions are supposed to be avoided and add those constraints into the heuristic search algorithm. This paradox makes it very difficult to use LLM with tools used in MAPF.

We have also tested the latest o1-preview model from OpenAI, which has been shown to significantly improve the reasoning and planning capability Valmeekam et al. (2024b). Compared to the standard rule-based checker, which is the default for all other experiments, we introduce a human checker that analyzes the explanation and chooses to retry when the current explanations could lead to future failure. We observe that such an improved checker can also facilitate the generation a lot, which further indicates the current reward model used in training and, potentially, the inference is not correct for MAPF. Examples and more explanations on why o1-preview leads to a worse performance compared to GPT-4-turbo in the default settings are provided in the appendix.

We have also tried tricks that could help improve the reasoning capabilities, including breaking the big paragraph into bullet-point style instruction, removing some 'useless' instruction, and adding a whole example for in-context learning. These tricks did not show any help in any scenarios.

3.2 Context Length Limit

Because the underlying architecture of popular LLMs is transformers, which further rely on self-attention, a longer context in the input will significantly increase the computing complexity in the process. Therefore, in their training process, they set a maximum limit of tokens on the input and trained specifically on them. Current large language models released will also provide a context length limit, which is as long as 200K tokens, and any request with a longer context length will be rejected. This is a long enough length for many tasks and even able to read a textbook, and many users are satisfied with the length. Recent studies have demonstrated that the performance of large models, such as GPT-4 Turbo with 128K context length, is not consistent when processing inputs of

n	MM	TOO	TOM
2	100	100	100
4	20	60	80
8	0	0	20

Table 5: The success rate (%) of different ways of inputting the map information to LLM on ROOM-32-32-4 map with different number of agents (n).

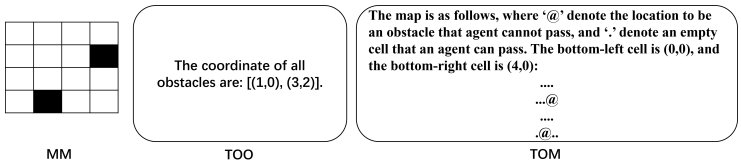

Figure 5: An illustration of the difference between multimodal input (MM), text that describes the whole map (TOM), and text that describes the obstacles (TOO).

8K versus 128K tokens [3]. What is even worse is that in the real-world applications of MAPF, the total number of agents running in the environment could be up to a few thousand. Even in the ideal case that our prompt will grow linearly to the number of agents, an environment with a thousand agents will require a total context length of $250(tokens/agent\ step) * 50(steps) * 1000(agents) = 5M$ tokens, which is much bigger than the current limit. Because the token limit is strict, we have introduced the restart mechanism in our methods, and in our current experiment scale, we have to restart the LLM around every 7-10 steps in large or complex scenarios. We also observed that the number of tokens used grows non-linearly with regard to the number of agents due to the increasing number of iterations, and we have put the numbers in the appendix.

In table. 3, we show our results on the ROOM map, and test with different GPT-4 models with different context lengths, in the setting of global observation (GO) only and the version with single-step observation (SSO). We will talk about these two settings in the next section. We found that when the context length limit grows, the success rate also increases. This is especially helpful when the single-step observation information that tells what valid actions each agent has is not provided. The failures here are not direct failures by exceeding the context length limit, but from the forgetting in restart caused by reaching the context length limits. Because of these restarts, our models often completely forget that they have been to certain locations, and then go back and forth in certain areas with dead ends like the center room in the ROOM map. We have also included the latest o1-preview model from OpenAI in the experiment on the biggest scale [4]. To our surprise, it is even worse than the GPT-4-turbo version. In the appendix, we provide a case study with more detailed explanations.

3.3 UNDERSTANDING OBSTACLE LOCATIONS

MAPF scenarios can be broken down into a pair of start location and goal locations, together with the map information. While understanding the coordinate version of start goal location pairs is relatively easy, understanding the map information is hard. Trained with publicly available text, LLM learns what a specific map means by finding similar contexts online, which mostly comes from other related fields in planning, like solving a maze. However, one problem is that people barely provide any such information online since people have the common knowledge of what to do with a map with code and preprocess the map information in the code rather than explicitly provide the set of where the obstacles are as the original input. Therefore, this leads to a lack of training data with related context about the information in the pictures or in a symbol-based input. As we mentioned earlier, 77% of the overall failure is caused by agents detouring back and forth in a certain area blocked by obstacles. These failures directly indicate that the LLM fails to understand the map information provided.

Because of this, our main prompt included guidance on what action could be taken in the current step to increase the success rate a little bit. In Table. 3, we find that while global observation can achieve a high win rate with the support of our high-level checkers, it also increases the average number of iterations per step. Furthermore, it fails in scenarios where SSO would be effective, as it exceeds the step limits defined in Sec. 2.3.1. In general, LLMs fail in medium complexity maps like ROOM, and completely fail in much harder maps like MAZE, even with the help of SSO.

[3] https://github.com/gkamradt/LLMTest_NeedleInAHaystack

[4] Due to accessibility, this is specifically tested on the webversion.

In all the experiments above, we use pure-text-based input because of the popularity and simplicity of pure-text input. However, this was not the case before: In earlier work of reinforcement learning for MAPF, people transformed the input into image-based input within the field of view and used neural networks like convolutional neural network (CNN) to read the information. Therefore, as shown in Fig. 5, we propose three different methods, which all share the same prompt of describing the problem definition and use different versions of the prompt describing the scenario:

1. A multimodal-based method MM where we use an image to give the picture of the map, and text to describe the problem and the scenario.
2. A text-only method TOO that describes the map by listing the coordinates of the obstacles.
3. A text-only method that describes the map by feeding the text version of the map with '@' denoting the obstacles and '.' denoting an empty space, named TOM.

We test the performance of these three variants, and show the results in Table. 5. We found that with an image-based input, the results are even worse. For example, large models can only have a high-level idea that there is a wall in the middle area, rather than knowing there is a wall exactly from $(3, 10)$ to $(20, 10)$. The failure of TOO further indicates that LLMs do not know how to combine high-level intuitions of where obstacles are with concrete reasoning during planning. Overall, in complex environments such as mazes and real-world city maps, LLMs cannot offer any form of guidance, not necessarily the specific action recommendations tested in this paper, until they can accurately comprehend the relationship between obstacles and the paths of agents.

4 Discussion

We would like to note that the methods in this paper are not designed to either outperform the state-of-the-art classic solvers or surpass the performance of learning-based solvers. Instead, the goal is to demonstrate that LLMs can solve small problems simply through prompting and discussing what is stopping them from solving larger scenarios. Specifically, current heuristc-based algorithms like conflict based search Sharon et al. (2015) can solve all the scenarios tested in the paper in less than 0.1 second. Moving on, we believe the three reasons for failures can each independently relate to different research directions. Improving capability in a long context is already a popular direction in NLP, and many papers are also looking into improving the general reasoning capability. We believe that much ongoing research has the opportunity to improve the performance of LLMs on MAPF, and we are pleased to have more researchers test this unique problem. Also, the recent development of building foundation models that combine language and image generation could potentially help with the reasoning in MAPF. Image generation models like stable diffusion Rombach et al. (2022) have shown to be a strong tool in single agent path finding Janner et al. (2022). While these diffusion models are not designed for multi-agent planning and are even hard to apply to MAPF, strategically using them could partially help the lack of reasoning capability. Regarding potential finetuning, we believe more annotated data are needed to help foundation models connect the abstract description to concrete coordinate-level knowledge of where obstacles are, and how to coordinate between agents. Future research could finetune using their data, or at least open their data to the general public, and let the next version of released models from industry companies include such data by automatically getting data online. Hopefully, such data could help foundation models know that position encodings are more important in planning problems. Besides, our results on current rule-based checker v.s. human checker also indicate that future research could work on building a good checker, i.e., a reward model in large language models, to improve the performance of LLMs.

Besides, although the success rate is one of the most important factors in measuring the performance of a solver, there are also other obstacles that need to be solved before using LLMs in real-world scenarios. First of all, success does not indicate anything about the quality of the solution. In our experiment, we found that the success scenarios usually come from easier scenarios that agents mostly only need to follow their direct shortest path. When planning gets harder, the success rate gets much worse. To improve the reasoning capability, future researchers and users could choose to make a dataset of scenarios and their good solutions publicly available on the internet, so when training with newer data, this specific problem could be directly included in the training set. Furthermore, the current workflow does not include any heuristics or tool use during the generation but only as a solution checker. While we are unclear on how such things should be included, this will be a very

challenging but meaningful direction. For potential researchers going in this direction, we want to remind them to keep looking at the consistency of the deconflict of agents, which could be seen as moving obstacles, and the deconflict of the fixed obstacles on the map. A successful method should consider both difficulties in a similar manner.

On the other hand, latency is always a problem for LLMs when used in real-world production. If one wants to use their own models, which is currently 500 tokens per second after using state-of-the-art system-wise optimization like vLLM Kwon et al. (2023) for LLAMA-2 Touvron et al. (2023). However, this speed is not enough as a typical round of output will have 1500 tokens, and it may take a few rounds before the LLM generates a valid solution. If one wants to use models provided through API like GPT-4 from OpenAI, the latency of the models will depend on a list of factors like network connection and server availability. In our paper, we typically need to wait around 15-30 seconds for one step to get completed, where less than 0.1 seconds are used for local processing.

5 Related Works

LLM for Reasoning Since the release of GPT-3, researchers have studied the effect of how they are performing on diverse sets of problems. Ideas like chain of thought Wei et al. (2022), in context learning Ye et al. (2023) have been shown to significantly improve the performance of LLMs in different tasks related to reasoningFu et al. (2022); Shum et al. (2023). A recent line of work has been introduced to use natural language as feedback in the process to introduce iterations and give LLMs more than one chance to generate correct solutions, which has been shown to significantly help in code generation and reasoning Chen et al. (2023a); Yang et al. (2023); Shinn et al. (2023). Specifically in LLMs for planning, there is a line of work that shows how bad LLMs are in general planning domains Valmeekam et al. (2023; 2024a), and in this paper, we are specifically looking into a specific problem in the planning domain.

LLM for Multi-agent Systems While there is extensive research on using LLMs for many different problems, there is little work that addresses the problem of LLM for multi-agent systems in the beginning. As time proceeds, there has been a list of works that promote research in social behavior by creating multi-agent environment powered by a lot of LLM agents Tan et al. (2023), create dialogue-based games with LLM Schlangen (2023). Recent work has shown that dialogue between multiple agents can help remove factual errors Du et al. (2023); Wang et al. (2023b). When it comes to solving problems related to multi-agent system, there are some works that are related to robotics Zhang et al. (2023); Mandi et al. (2023) while limited to the scale of two to three agents. Chen et al. (2023b) studied whether using a more decentralized controller can help to solve the context length limit problem when the number of agents is higher, and concluded that in a centralized controller environment, creating separate LLMs for different agents does not help improve the success rate. While they succeed in their domains, in this paper, we specifically look at the problem of MAPF, and we show the cause of failure in the MAPF problem because of its unique challenges.

Multi-agent Path Finding Multi-agent path finding (MAPF) is a problem that has won much attention in recent years because of its close relationship to real-world applications. It has previously been solved with more classic methods like heuristic search algorithms Sharon et al. (2015); Li et al. (2021); Okumura (2023), rule-based algorithms Han & Yu (2020), and reduction-based algorithms Surynek et al. (2016). While learning-based approaches have not yet outperformed classic approaches, they have also begun to win a lot more focus for their fast inference time and generalizability. PRIMAL Sartoretti et al. (2019) proposed to learn a policy for MAPF using a combination of reinforcement learning and imitation learning. Following that, a group of works proposed a diverse set of methods from building curriculum Damani et al. (2021) to follow guidance from classic methods in each step Skrynnik et al. (2023). In this paper, we focus on using LLM, which is a learning-based approach but is not specifically fine-tuned for MAPF problems.

6 Conclusion

In this paper, we investigate using LLMs to solve the MAPF problem. We first show that LLMs could solve easy scenarios, while being unable to generate valid solutions when the problem gets harder stably. To make our work applicable to the future, we elaborate on the failures and show

that they can be broken down into three different aspects. We used extensive experiments to support our breakdown and discussed how the difficulties could be addressed by people from different backgrounds.

REFERENCES

Saaket Agashe, Yue Fan, and Xin Eric Wang. Evaluating multi-agent coordination abilities in large language models. *arXiv preprint arXiv:2310.03903*, 2023.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023a.

Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas Roy, and Chuchu Fan. Scalable multi-robot collaboration with large language models: Centralized or decentralized systems? *arXiv preprint arXiv:2309.15943*, 2023b.

Mehul Damani, Zhiyao Luo, Emerson Wenzel, and Guillaume Sartoretti. Primal$_2$: Pathfinding via reinforcement and imitation multi-agent learning - lifelong. *IEEE Robotics Autom. Lett.*, 6(2): 2666–2673, 2021. doi: 10.1109/LRA.2021.3062803. URL https://doi.org/10.1109/LRA.2021.3062803.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *CoRR*, abs/2305.14325, 2023. doi: 10.48550/ARXIV.2305.14325. URL https://doi.org/10.48550/arXiv.2305.14325.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. Complexity-based prompting for multi-step reasoning. *arXiv preprint arXiv:2210.00720*, 2022.

Shuai D. Han and Jingjin Yu. DDM: fast near-optimal multi-robot path planning using diversified-path and optimal sub-problem solution database heuristics. *IEEE Robotics Autom. Lett.*, 2:1350–1357, 2020.

Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Paul Saldyt, and Anil B Murthy. Position: Llms can't plan, but can help planning in llm-modulo frameworks. In *Forty-first International Conference on Machine Learning*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Jiaoyang Li, Kexuan Sun, Hang Ma, Ariel Felner, T. K. Satish Kumar, and Sven Koenig. Moving agents in formation in congested environments. In Daniel Harabor and Mauro Vallati (eds.), *Proceedings of the Thirteenth International Symposium on Combinatorial Search, SOCS 2020, Online Conference [Vienna, Austria], 26-28 May 2020*, pp. 131–132. AAAI Press, 2020. doi: 10.1609/SOCS.V11I1.18525. URL https://doi.org/10.1609/socs.v11i1.18525.

Jiaoyang Li, Wheeler Ruml, and Sven Koenig. Eecbs: A bounded-suboptimal search for multi-agent path finding. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 12353–12362, 2021.

Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J. Stuckey, and Sven Koenig. MAPF-LNS2: fast repairing for multi-agent path finding via large neighborhood search. In *Proceedings of the Thirty-Sixth Conference on Artificial Intelligence (AAAI)*, pp. 10256–10265. AAAI Press, 2022.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization. *arXiv preprint arXiv:2310.02170*, 2023.

Zhao Mandi, Shreeya Jain, and Shuran Song. Roco: Dialectic multi-robot collaboration with large language models. *arXiv preprint arXiv:2307.04738*, 2023.

Keisuke Okumura. Lacam: Search-based algorithm for quick multi-agent pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 11655–11662, 2023.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, T. K. Satish Kumar, Sven Koenig, and Howie Choset. PRIMAL: pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics Autom. Lett.*, 4(3):2378–2385, 2019. doi: 10.1109/LRA.2019.2903261. URL https://doi.org/10.1109/LRA.2019.2903261.

David Schlangen. Dialogue games for benchmarking language understanding: Motivation, taxonomy, strategy. *arXiv preprint arXiv:2304.07007*, 2023.

Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell.*, pp. 40–66, 2015.

Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*, 2023.

KaShun Shum, Shizhe Diao, and Tong Zhang. Automatic prompt augmentation and selection with chain-of-thought from labeled data. *arXiv preprint arXiv:2302.12822*, 2023.

Alexey Skrynnik, Anton Andreychuk, Maria Nesterova, Konstantin Yakovlev, and Aleksandr Panov. Learn to follow: Decentralized lifelong multi-agent pathfinding via planning and learning. *arXiv preprint arXiv:2310.01207*, 2023.

Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, and Eli Boyarski. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the Twelfth Annual Symposium on Combinatorial Search (SoCS)*, pp. 151–159. AAAI Press, 2019.

Pavel Surynek, Ariel Felner, Roni Stern, and Eli Boyarski. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *22nd European Conference on Artificial Intelligence (ECAI)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pp. 810–818. IOS Press, 2016.

Shuhan Tan, Boris Ivanovic, Xinshuo Weng, Marco Pavone, and Philipp Kraehenbuehl. Language conditioned traffic generation. *arXiv preprint arXiv:2307.07947*, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288, 2023. doi: 10.48550/ARXIV.2307.09288. URL https://doi.org/10.48550/arXiv.2307.09288.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models–a critical investigation. *arXiv preprint arXiv:2305.15771*, 2023.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems*, 36, 2024a.

Karthik Valmeekam, Kaya Stechly, and Subbarao Kambhampati. Llms still can't plan; can lrms? a preliminary evaluation of openai's o1 on planbench. *arXiv preprint arXiv:2409.13373*, 2024b.

Yen-Jen Wang, Bike Zhang, Jianyu Chen, and Koushil Sreenath. Prompt a robot to walk with large language models. *arXiv preprint arXiv:2309.09969*, 2023a.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. Unleashing cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. *CoRR*, abs/2307.05300, 2023b. doi: 10.48550/ARXIV.2307.05300. URL https://doi.org/10.48550/arXiv.2307.05300.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Seonghyeon Ye, Hyeonbin Hwang, Sohee Yang, Hyeongu Yun, Yireun Kim, and Minjoon Seo. In-context instruction learning. *arXiv preprint arXiv:2302.14691*, 2023.

Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models. *arXiv preprint arXiv:2307.02485*, 2023.

A Complete Prompts

In Fig. 6, we give our system prompt. Between different maps, the only thing that changed is the text highlighted in blue, and all other parts are the same. We provide the user prompts in Fig. 7 and Fig. 8.

B Token Length Grows

In Fig. 9, we show the average prompt length per agent per step, which includes both the input and the output. We found that with the growing number of agents, this average is also growing, which means that the output length is growing faster than linear. This contributes to the fact that the more agents, the more complex the environment, and thus, the longer explanation and more iterations until a correct plan is needed. We also observe that in failing scenarios where the agents go back and forth, the token length is generally shorter, given that they did not run into the iterative deconflict process, which takes a lot of tokens to solve.

C Comparison Between Rule-based Checker and LLM as Checker

In many domains, people are using LLM as the checker and provide feedback to another LLM, which serves as an actor. While we choose to use a rule-based checker because of its reliability and speed, here we provide some results on how bad the performance of an LLM-based checker could be. We randomly picked 50 different step information that needs the checker to verify from our discussion history. And the success rate is 76%, and the average time spent for each check is 3 seconds. This success rate is why we choose to use the rule-based checker.

You are a decision-maker in a warehouse system that is running as a 4-connected grid-based multi-agent path finding system of [[Map Description]]. You will be provided with the current scenario in which you need to choose actions for each agent, i.e., the detailed information that provides where the obstacles are, together with each agent's current location and goal locations. The objective is to minimize the time of the agent who arrives at its goal location at the latest while keeping no agents from colliding with each other at any timestep. You only need to consider collision between two agents located in the same cell after each move. On each timestep, you could choose one action from moving up, moving down, moving left, moving right, and staying without moving. You should try to avoid collisions not only in the current timestep but also look ahead to a few timesteps if it is important to avoid future collisions.
Remember that to avoid a collision, normally, only one agent needs to change their action from the original action. Nevertheless, your job is not to generate a whole path but to do it step-by-step, and the information on whether a specific action is valid or not is provided in each single step. Remember that the given action is based on the obstacles, so the available action will not change if the agent chooses to stay at the same location. You only need to give a single-step action for each agent to follow; do not give more steps until I tell you to.
First, give your solution in the same format as follows, with some explanation if the agent is not moving or moving in a direction that is not towards its goal. (This is not the actual scenario, but an example of the desired output format. You will be provided with the actual scenario later):
Step 1:
Agent 1:
Current: (1,0)
Goal: (1,0)
Move: Stay, as it has already reached its goal.

Agent 2:
Current: (5,4)
Goal: (5,6)
Move: Up to (5,5)

Agent 3:
Current: (0,7)
Goal: (6,4)
Move: Right to (1,7)

Then, validate your solution by listing all the coordinates of the agents after the move and check if there is any pair of agents that have the same coordinates. If there is a collision by mistake, regenerate another solution immediately and validate again by listing the coordinates of all the agents until you get a collision-free solution. Finally, end your output with your validated solution in a new paragraph with the format of a sequence of : [[Agent_id]]: [[Coordinate]].

Figure 6: An example of the system prompt to specify the MAPF system and the objective. The text highlight in blue will be replaced by map description. For example, in room-32-32-4, the prompt will be 'room-like map with size 32*32'.

D CASE STUDY ON GPT-O1-PREVIEW

In the main paper, we showed that the latest model of (GPT-)o1-preview has a worse performance than the previous GPT-4-turbo. To study this anomaly, we provide a case study here.

As shown in Fig. 10, compared to the previous version, GPT-o1-preview now learns to explicitly try to find an alternative route, and also, it now learns to use its previous history more strategically. After the model has found out that moving right is probably not helpful to him, it will not try to go right again. However, this is making the result worse, given the current capability of the model is not strong enough. We observe that while at location (1, 17), agent 5 could have moved up, GPT-o1-preview still chose to go back to the left. This fails to meet the motivation of trying to find an alternative path when meeting an obstacle. It is noteworthy that after a retry, it chooses to move up and end up in a successful final plan. Similar scenarios happens which leads to the improvement in Table. 10, which shows that GPT-o1-preview has the capability of at least matching the performance of the previous model, but the current result is coming from the randomness of LLMs. Also, this opens up the potential for future research to study how to automatically integrate the regeneration process when provided with a more advanced checker. This checker would need to go beyond merely

Agent 1 is currently in (0,2), and wants to go to (3,1).
Agent 2 is currently in (1,3), and wants to go to (2,0).
The map is as follows, where '@' denotes a cell with an obstacle that an agent cannot pass, and '.' denotes an empty cell that an agent can pass.
The bottom-left cell is (0,0) and the bottom-right cell is (31,0):
....
...@
....
.@..
In the next step:
Agent 1 can move ['stay at (0, 2)', 'right to (1, 2)', 'up to (0, 3)', 'down to (0, 1)'].
Agent 2 can move ['stay at (1, 3)', 'left to (0, 3)', 'right to (2, 3)', 'down to (1, 2)'].

Figure 7: An example of the user prompt for describing the scenario. Text in blue is a scenario-specific prompt, while text in orange is a map-specific prompt. In the experiments on the empty map, only the first blue paragraph will be provided, and all text starting from the black paragraph is removed because there are no obstacles. The text in purple is the single-step observation (SSO) information.

[[Success]]
Good job. Keep moving. In the next step:
Agent 1 can move ['stay at (0, 2)', 'right to (1, 2)', 'up to (0, 3)', 'down to (0, 1)'].
Agent 2 can move ['stay at (1, 3)', 'left to (0, 3)', 'right to (2, 3)', 'down to (1, 2)'].
[[Failure]]
You are wrong. Agents (1,2), and (4,5) are colliding with each other. Please correct the current step.
You are wrong. Agent 2,4 is colliding with obstacles. Please correct the current step.

Figure 8: An example of the user prompt starting from the second step. While here we demonstrate a few options, only one of them, i.e., text in one color, will be provided to the LLM in one iteration.

assessing the final output, like the current checker, and instead evaluate the entire reasoning process. The checker should be able to predict whether the current explanations are likely to lead to failure in future steps, rather than simply determining if the current step is correct. This aligns with evaluating the Q^π value in a Markov decision process, where π represents the generation policy of the large language model (LLM). Unlike V^*, which evaluates the current state based on an optimal policy, Q^π would allow the model to assess whether the current action will result in long-term success or failure in tasks such as MAPF. For instance, the checker should recognize that while the agent could theoretically move back to (1, 17) later, if it chooses to move left to (0, 17) at step 6, the model's policy will not opt for this, indicating the need for regeneration at this step rather than later. It is also noteworthy that even when large reasoning models (LRM) like GPT-o1-preview have success at some point, it will also introduce a much longer inference time and make it even harder to deploy in the real world. We also provide another example in Fig. 11, where GPT-o1-preview has also shown its reasoning of trying one step, and going back. However, this example goes back directly because it sees no option to go to the goal location after the detour. This is one of the examples that LLM still fails even after we incorporate the human checker in the middle because it still does not have a strong capability in path planning.

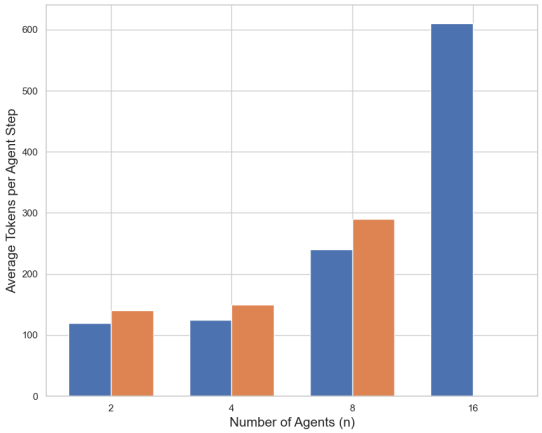

Figure 9: Average token length per agent step on success scenarios.

<Assistant>
Step 5:
...
Agent 5:
Current: (0,17)
Goal: (0,23)
Move: Right to (1,17), aiming to find a path upwards since moving up is blocked.
...
<User>
Keep moving. In the next step, ...
<Assistant>
...
Charting new routes...
Agent 5 is mapping out new paths towards (0,23). Moving left or up from (1,17) could be viable, but obstacles might alter their course. Interessen her own challenges and adjusting trajectory.
...
Step 6:
...
Agent 5:
Current: (1,17)
Goal: (0,23)
Move: Left to (0,17)
...
<User>
Keep moving. In the next step, ...
<Assistant>
Step 7:
...
Agent 5:
Current: (0,17)
Goal: (0,23)
Move: Stay at (0,17), as moving right doesn't help and moving up is blocked.
...

Figure 10: An example of part of the output from GPT-o1-preview model. The main difference compared to previous model in the reasoning is highlighted in red. Italics are the part shown as the thought of GPT-o1-preview in the web version.

<Assistant>
Step 8:
...
Agent 3:
Current: (17,5)
Goal: (17,1)
Move: Up to (17,6)
Explanation: Moving up to find a path that allows decreasing the y-coordinate towards the goal.
...
<User>
Keep moving. In the next step, ...
<Assistant>
Step 9:
...
Agent 3:
Current: (17,6)
Goal: (17,1)
Available moves: ['stay at (17,6)', 'right to (18,6)', 'up to (17,7)', 'down to (17,5)']
Move: Down to (17,5)
...
<User>
Keep moving. In the next step, ...
<Assistant>
Step 10:
...
Agent 3:
Current: (17,5)
Goal: (17,1)
Available moves: ['stay at (17,5)', 'left to (16,5)', 'right to (18,5)', 'up to (17,6)']
Move: Stay at (17,5)
Explanation: No available moves bring Agent 3 closer to its goal without moving away.
...

Figure 11: Another example of part of the output from GPT-o1-preview model.