Minesweeper Agent: Exploring the Reasoning Ability of Large Language Models

Shihan Wu
School of Computer Science and Engineering
University of Electronic Science and Technology of China
shihan.wu.koorye@outlook.com

Abstract

Large Language Models (LLMs) have shown remarkable performance in various natural language processing tasks. However, it is unclear how well they can perform in reasoning tasks that require logical thinking and decision-making. In this paper, we propose a novel task to evaluate the reasoning ability of LLMs: Minesweeper Agent. We design a Minesweeper environment and a set of rules to generate Minesweeper puzzles, the challenges of which require the LLM to reason and make decisions based on the information it has observed. With elaborate design of the environment and prompts, several popular LLMs are evaluated on the Minesweeper Agent task. Our work provides a new perspective to evaluate the reasoning ability of LLMs and sheds light on the potential of LLMs in reasoning tasks. Code is available at https://github.com/Koorye/Minesweeper-Agent.

1 Introduction

Large Language Models (LLMs) have achieved remarkable performance in various natural language processing tasks, such as machine translation [Vaswani *et al.*, 2017], question answering [Devlin *et al.*, 2018], and text generation [Radford *et al.*, 2019]. With the development of LLMs, the researchers focus on the reasoning ability of LLMs, which is crucial for many tasks that require logical thinking and decision-making. However, it is still unclear how well LLMs can perform in reasoning tasks.

The reasoning ability of LLMs is a challenging problem. The existing benchmarks, such as the bAbI dataset [Weston *et al.*, 2015] and the CLEVR dataset [Johnson *et al.*, 2017], are designed to evaluate the reasoning ability of models. However, these benchmarks are limited in the complexity of reasoning tasks. The bAbI dataset contains simple reasoning tasks, such as counting and deduction, while the CLEVR dataset contains visual reasoning tasks. After this, The researchers focused on complex multi-step reasoning tasks that require interaction with the environment, like navigation tasks [Mirowski *et al.*, 2018] and text-based games [Narasimhan *et al.*, 2015]. These tasks are more challenging and require the model to reason and make decisions based on the information it has observed.

Although the existing benchmarks have made progress in evaluating the reasoning ability of models, they still have some limitations. VQA tasks require the model to answer questions based on the information in the image, which is not suitable for evaluating the reasoning ability of LLMs. Text-based games require the model to interact with the environment, which is complex and difficult to evaluate. Therefore, it is necessary to design a new task to evaluate the reasoning ability of LLMs.

Minesweeper is a classic game that requires the player to uncover all the cells on the board without detonating any mines. The player can uncover a cell by clicking on it, and the cell will show a number indicating the number of mines in the neighboring cells. The player can also mark a cell as a mine by right-clicking on it. The game ends when the player uncovers all the cells without detonating any mines. Minesweeper is a challenging game that requires the player to reason and make decisions based on the information it has observed. Therefore, Minesweeper is a suitable task to evaluate the reasoning ability of LLMs. In order to evaluate the reasoning ability of LLMs, we design a Minesweeper environment and a set of rules to generate Minesweeper puzzles. When the LLMs receive environmental feedback and prompt input and gives a response, it can parse the action from the response and then perform the action to obtain the next environmental feedback. The model can reason and make decisions based on the information it has observed. With elaborate design of the environment and prompts, we evaluate the reasoning ability of LLMs on the Minesweeper Agent task. In addition, in order to evaluate the reasoning ability of the model, we designed a series of metrics, including winning percentage, number of steps, number of effective steps, etc. These metrics can evaluate the reasoning ability of the model from different perspectives.

Sufficient experiments show that the Minesweeper Agent task can effectively evaluate the reasoning ability of LLMs. As the number of mines increase, the reasoning ability of the model sharply decreases. Also, different LLM has different performance in Minesweeper games. The results show that the reasoning ability of LLMs in Minesweeper games still needs to be improved. Our work provides a new perspective to evaluate the reasoning ability of LLMs and sheds light on the potential of LLMs in reasoning tasks.

In summary, the contributions of this paper are as follows:

- We propose a novel task - Minesweeper - to evaluate the reasoning ability of LLMs.
- We create a Minesweeper environment and a set of rules to generate Minesweeper puzzles, and design a series of metrics to evaluate the reasoning ability of the model.
- Experimental results show that the LLMs still need to improve their reasoning ability in Minesweeper games.

2 Related Work

2.1 Large Language Models

Large Language Models have achieved remarkable performance in various natural language processing tasks. The researchers have proposed a series of LLMs, such as BERT [Devlin *et al.*, 2018], GPT [Radford *et al.*, 2018], and XLNet [Yang *et al.*, 2019]. These models have achieved great success in various natural language processing tasks, such as machine translation, question answering, and text generation. After this, OpenAI has proposed a series of LLMs, such as GPT-2 [Radford *et al.*, 2019] and GPT-3 [Brown *et al.*, 2020]. These models have achieved remarkable performance in various natural language processing tasks. With the parameter size and data scale increasing, LLMs have shown powerful emergent ability. Even on tasks that are not explicitly trained on, LLMs can still achieve good performance. This phenomenon lays the foundation for the LLMs to serve as a general-purpose AI system.

2.2 Prompt Engineering

Prompt engineering is a technique that uses prompts to guide the model to generate the desired output. The researchers have proposed a series of prompt engineering methods, such as P-tuning [Schick and Schütze, 2020a], Few-Shot Prompt Learning [Schick and Schütze, 2021], and Prompt-based Zero-Shot Learning [Schick and Schütze, 2020b]. These methods have achieved remarkable performance in various natural language processing tasks. To improve the reasoning ability of LLMs, the researchers proposed the Chain-of-Thought (CoT) [Wei *et al.*, 2022], Step-back Prompting [Zheng *et al.*, 2023] and self-correcting prompting methods based on environment feedback. These methods have achieved remarkable performance in various reasoning tasks. In addition, soft prompt tuning [Zhou *et al.*, 2022] methods are becoming popular by inserting learnable vectors into the model. With fine-tuning the model with a small number of learnable vectors, the model can achieve good performance on various tasks. These methods have achieved remarkable performance in various natural language processing tasks.

2.3 LLM-based Agent

Agent is a system that can interact with the environment and make decisions based on the information it has observed. Traditional agent system is based on rule-based methods, which require manual design of rules. With the development of deep learning, the researchers proposed a series of deep reinforcement learning methods, such as DQN [Mnih *et al.*, 2015], A3C [Mnih *et al.*, 2016], and PPO [Schulman *et al.*, 2017]. These methods have achieved remarkable performance in various games, such as Atari games and Go games. In order to improve the reasoning ability of the model, the researchers proposed a series of LLM-based agent methods, without training on the environment, the model can achieve good performance on various tasks. LLM-base agent methods use LLMs to generate responses based on the environment feedback and prompts, and then parse the action from the response and perform the action to obtain the next environmental feedback. Through Chain-of-Thought (CoT) [Wei *et al.*, 2022], problem splitting techniques [Song *et al.*, 2023], LLM-based agent methods can achieve good performance on various reasoning tasks, such as autonomous driving [Chen *et al.*, 2023], navigation [Lin *et al.*, 2024], and robot control [Kannan *et al.*, 2023].

3 Environment

In this section, we introduce the Minesweeper environment and the rules and interactions of the Minesweeper game.

3.1 Overview of the Minesweeper

The Minesweeper game is a single-player puzzle game. The player is presented with a board of hidden areas, some of which contain mines. The player's goal is to uncover all the areas that do not contain mines without detonating any mines. The player can uncover a area by clicking on it, and the area will show a number indicating the number of mines in the neighboring areas. The player can also mark a area as a mine by right-clicking on it. The game ends when the player uncovers all the areas without detonating any mines.

As shown in Figure 1, the Minesweeper environment consists of a board of hidden areas, some of which contain mines. The player can uncover an area, and the area will show a number indicating the number of mines in the neighboring areas. The player can also flag an area as a mine. The game ends when the player uncovers all the areas that do not contain mines.

3.2 Rules and interactions

Figure 1 shows the rules and interactions of the Minesweeper game. The Minesweeper environment consists of a board of hidden areas, there are 4 types of areas in the board:

- **Unrevealed Area**: The area is hidden and has not been revealed.
- **Safe Area**: The area is safe and does not contain a mine. If the player uncovers a safe area, the area will show a number indicating the number of mines in the neighboring areas.
- **Flagged Area**: The area is flagged as a mine. If the player flags a area as a mine, the area will show a flag.
- **Mine Area**: The area contains a mine. If the player uncovers a mine area, the game ends.

With different type of areas, the player can interact with the environment in the following ways:

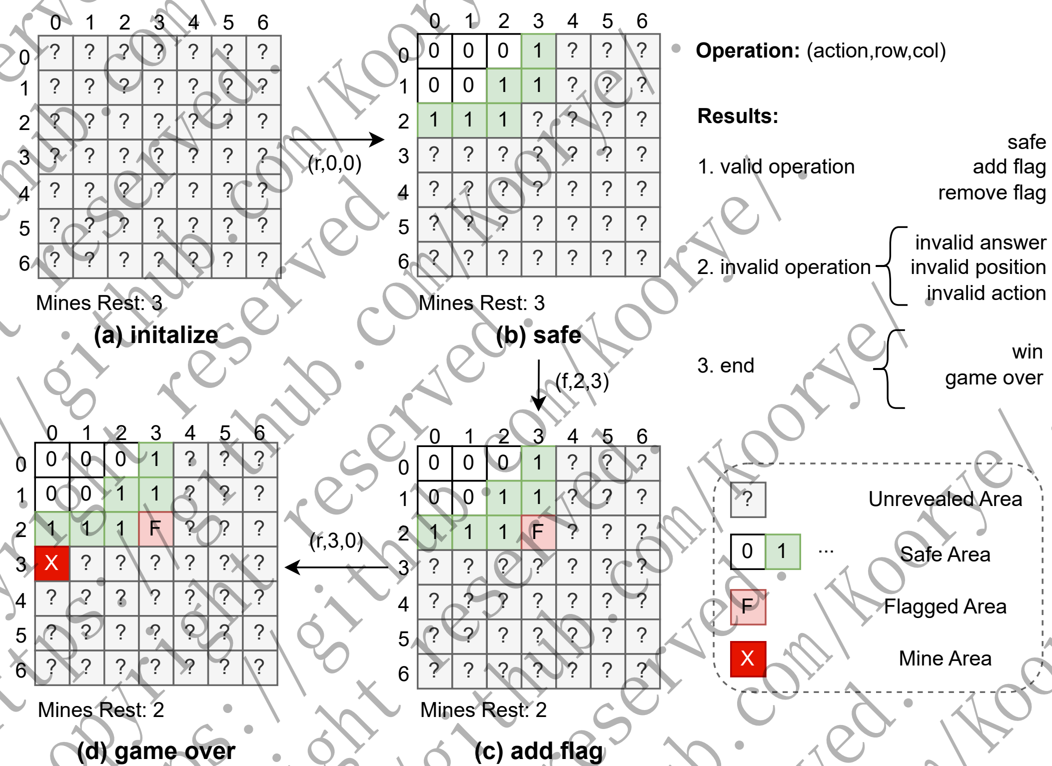

Figure 1: The Minesweeper environment.

- **Uncover Area**: The player can uncover a unrevealed area by clicking on it. If the area is a safe area, the area will show a number indicating the number of mines in the neighboring areas. If the area is a mine area, the game ends.
- **Flag Area**: The player can flag a unrevealed area as a mine by right-clicking on it. If the area is a flagged area, the area will show a flag. If the area is a mine area, the game ends.
- **Unflag Area**: The player can unflag a flagged area by right-clicking on it. If the area is a flagged area, the area will become a unrevealed area.

After performing the above operations in the environment, the player can observe the feedback from the environment and make decisions based on the information it has observed. There are 3 types of feedback in the Minesweeper environment:

- **Valid Operation**: The player performs a valid operation, and the environment returns the feedback, including safe, add flag and remove flag.
- **Invalid Operation**: The player performs an invalid operation, and the environment returns the feedback, including invalid answer, invalid position and invalid action.
- **End**: The game ends, and the environment returns the feedback, including win and lose, including win and game over.

Each operation in the Minesweeper environment is a tuple (action, row, col), where action is the action the player performs, row and col are the row and column of the area the player performs the action on. After the player performs the action, the environment returns one of the feedback mentioned above. The full feedback trigger details can be found in the appendix A.

4 Method

In this section, we introduce the Minesweeper Agent, which consists of 5 components: prompt generation, large language model, answer parser, environment interaction and memory.

4.1 Method Overview

The Minesweeper Agent is designed to evaluate the reasoning ability of Large Language Models (LLMs) in Minesweeper games. As shown in Figure 2, the Minesweeper Agent consists of 5 components: prompt generation, large language model, answer parser, environment interaction and memory (optional). Prompt generation is used to generate the prompt input for the model, including the prompt before, environment information and prompt after. The large language model is the core component of the Minesweeper Agent, which receives the Minesweeper environment feedback and prompt input, and gives a response. The answer parser is used to

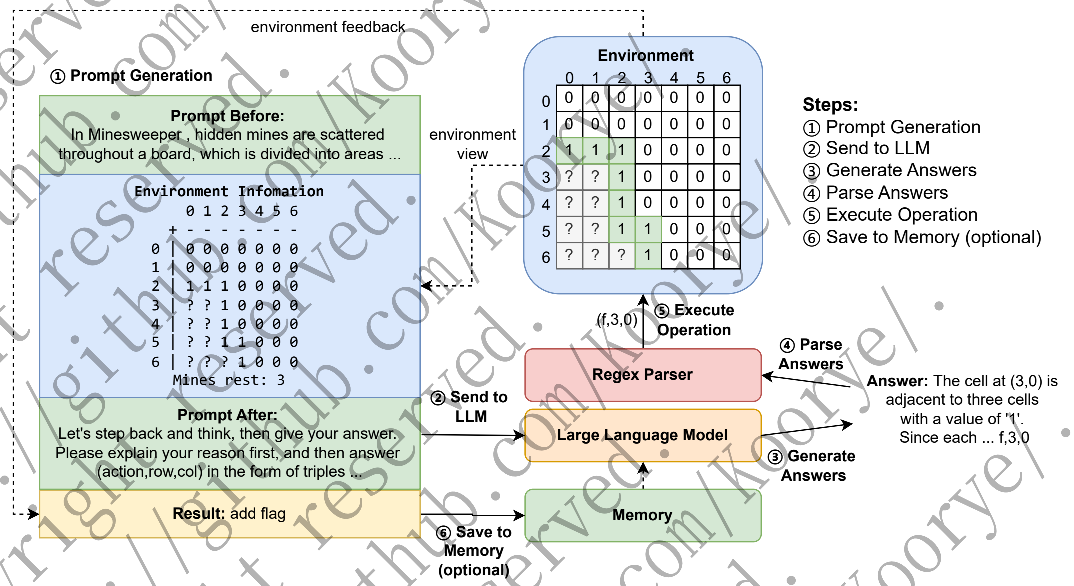

Figure 2: The Minesweeper Agent.

parse the action from the response of the model. The environment interaction is used to generate Minesweeper puzzles and provide environmental feedback. Memory is a optional component in LLM-based agent, which can store the information the model has observed.

With these components, the Minesweeper Agent can reason and make decisions based on the information it has observed. For each step, the Minesweeper Agent receives the Minesweeper environment feedback and prompt input, and gives a answer. The answer parser parses the action from the response, and the Minesweeper Agent performs the action to obtain the next environmental feedback. As the memory enables, the Minesweeper Agent can store the information it has observed and reason based on the information it has observed. The Minesweeper Agent runs step by step until the game ends: win or game over.

4.2 Prompt Generation

Prompt Engineering is a crucial part of fine-tuning large language models. With carefully designed prompts, we can guide the model to perform specific tasks. In the Minesweeper Agent task, we design a set of prompts to guide the model to play Minesweeper games. The prompt consists of 3 parts: prompt before, environemnt information and prompt after. Here are the introdution of the 3 parts of prompt:

- **Prompt Before**: The prompt before is the initial prompt that introduces the Minesweeper game. The prompt before contains the rules and interactions of the Minesweeper game, and the player's goal is to uncover all the areas that do not contain mines without detonating any mines. It can also contain several tips and tricks of the Minesweeper game.
- **Environment Information**: The environment information is the information the model receives from the Minesweeper environment. The environment information contains the board of hidden areas, the number of mines in the neighboring areas, and the feedback from the environment. The environment information is used to guide the model to reason and make decisions based on the information it has observed.
- **Prompt After**: The prompt after is the prompt that asks the model to make a decision. The prompt after contains the action options, such as unreveal an area or flagged an area, and the model needs to choose an action based on the information it has observed. The prompt after can also contain some examples of the correct action.

Full detail of the above parts of prompt can be found in the appendix B. With elaborate design of the prompt, the model can reason and make decisions based on the information it has observed.

4.3 Large Language Model

The Large Language Model (LLM) is the core component of the Minesweeper Agent. The LLM receives the Minesweeper environment feedback and prompt input, and gives a response. Without any training on Minesweeper games, the LLM can reason based on model input. Depend on the LLMs ability to understand the prompt and generate the correct response, the LLM can reason and make decisions based on the information it has observed.

4.4 Answer Parser

The Answer Parser is used to parse the action from the response of the model. The model generates a response based on the prompt input, and the answer parser parses the action from the response. Previous work may have splitted LLM responses, or predicted actions through learnable networks. In this work, we use a simple rule-based answer parser to parse

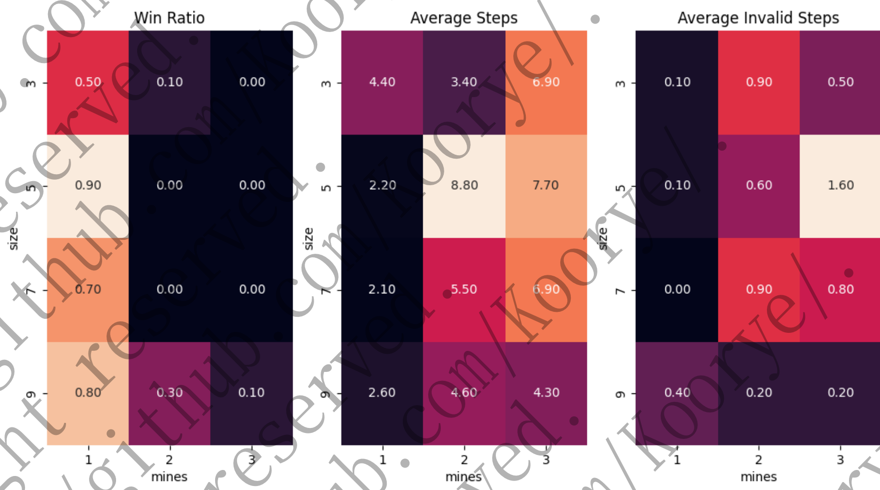

Figure 3: The main results of the Minesweeper Agent.

the action from the response. With regex matching, the answer parser can parse the action from the response and then perform the action to obtain the next environmental feedback. The action should be as a tuple of (action, row, col), where action is the action type, row and col are the position of the action.

4.5 Environment Interaction

The Minesweeper environment is used to generate Minesweeper puzzles and provide environmental feedback. The Minesweeper environment contains the board of hidden areas, each area contains a status of hidden, flagged, or the number of mines in the neighboring areas. The Minesweeper environment provides the environmental feedback to the model, and the model can reason and make decisions based on the information it has observed. The model can parse the action from the response and then perform the action to obtain the next environmental feedback.

In order to interact with the Minesweeper environment, the Minesweeper Agent needs to perform the following actions:

- **View**: The model can view the total status of the board, including status of each area and the mines rest on the board.
- **Execute Operation**: The model can execute the operation via a tuple of (action, row, col), where action is the action type, row and col are the position of the action. The action type can be unreveal an area or flagged an area. After the model executes the operation, the Minesweeper environment will provide the environmental feedback.

With the Minesweeper environment, the Minesweeper Agent can reason and make decisions based on the information it has observed.

4.6 Memory

Memory is a optional component in LLM-based agent, which can store the information the model has observed. With environment feedback, the memory can save a large number of action execution history, and the model can reason and make decisions based on the information it has observed. Based on memory, LLM can effectively correct previous incorrect answers and improve the reasoning ability of the model.

In our method, we developed a simple memory module to store the information the model has observed, including the prompt input and the environmental feedback. Memory helps the model jump out of the local optimal solution and approximate the correct minesweeper game solution.

5 Experiment

In this section, we introduce the experiment setup, implementation details, metrics, main results and ablation study.

5.1 Implementation Details

We implement the Minesweeper Agent based on the online LLM apis, including Qwen [Bai *et al.*, 2023], Spark, ChatGLM [Du *et al.*, 2022], etc. The Minesweeper environment and agent is implemented in Python language. Without any GPUs, the Minesweeper Agent can be evaluted on a local personal computer. About the prompt designs, we introduce some tips and tricks, examples and the Step-back Prompting technique to enhance the performance of the Minesweeper Agent. The details of the prompt designs can be found in the appendix B.

5.2 Metrics

We use the following metrics to evaluate the performance of the Minesweeper Agent:

- **Winning Rate**: The percentage of games that the Minesweeper Agent wins. Winning rate is the most important metric to evaluate the performance of the Minesweeper Agent, as the ultimate goal of the Minesweeper Agent is to win the game.

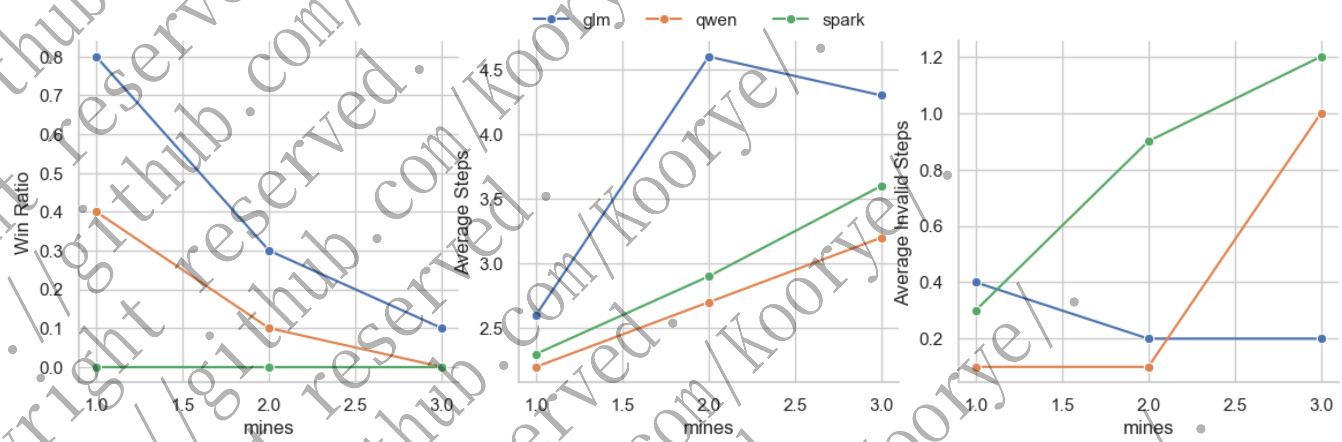

Figure 4: The performance of the Minesweeper Agent with different LLMs.

Mines	Model	Winning Rate	Average Steps	Average Invalid Steps
1	Baseline	0.0	2.2	0.1
	+ Examples	0.0	2.1	0.0
	+ Step Back Prompting	0.8	2.5	0.2
	+ Tips and Tricks	**1.0**	**2.2**	**0.2**
	+ Memory	0.8	2.6	0.4
2	Baseline	0.0	2.4	0.0
	+ Examples	0.0	2.2	0.0
	+ Step Back Prompting	0.0	3.6	0.4
	+ Tips and Tricks	0.1	4.4	1.0
	+ Memory	**0.3**	**4.6**	**0.2**
3	Baseline	0.0	3.3	0.3
	+ Examples	0.0	2.5	0.4
	+ Step Back Prompting	0.0	4.0	0.8
	+ Tips and Tricks	0.0	4.3	1.5
	+ Memory	**0.1**	**4.3**	**0.2**

Table 1: The performance of the Minesweeper Agent with different prompt designs.

- **Average Steps**: The average steps the Minesweeper Agent takes to win or lose a game. The Minesweeper Agent should take as few steps as possible to win a game. However, if the agent loses a game, it should take as many steps as possible to survive. The average steps can reflect the efficiency of the Minesweeper Agent.
- **Average Invalid Steps**: The average invalid steps the Minesweeper Agent takes to win a game. The Minesweeper Agent may get stuck in a loop of ineffective operations, such as repeatedly clicking on the same cell, which is considered as invalid steps. The Minesweeper Agent should take as few invalid steps as possible, it reflects the effectiveness of the Minesweeper Agent.

The above metrics can evaluate the performance of the Minesweeper Agent from different perspectives. The winning rate can evaluate the overall performance of the Minesweeper Agent, while the average steps and average invalid steps can evaluate the efficiency and effectiveness of the Minesweeper Agent.

5.3 Main Results

We evaluate the Minesweeper Agent on a set of Minesweeper puzzles with different map sizes and numbers of mines. The results are shown in Figure 3. It can be seen that the agent's win rate can reach more than 0.5 or even 0.9 when there is only one mine, but the win rate sharply decreases as the number of mines increases, when the number of mines is 2 or 3, the win rate is 0 in most cases. On the other hand, as the number of mines increases, the Minesweeper Agent needs more steps to complete the game. The number of invalid steps also increases with the increase of the number of mines. However, the map size has little effect on the performance of the Minesweeper Agent.

According to Figure 3, as the number of mines increase, the winning rate of the Minesweeper Agent decreases. The average steps and average invalid steps also increase with the increase of the number of mines. The results show that the Minesweeper Agent still needs to improve its reasoning ability. The Minesweeper Agent can win the game with a high winning rate when there is only one mine, but it is difficult to win the game when there are more mines. The Minesweeper Agent needs to improve its reasoning ability to win the game with a high winning rate when there are more mines.

5.4 Ablation Study

We conduct an ablation study to evaluate the effectiveness of the Minesweeper Agent. We compare the Minesweeper Agent with different LLMs, including Qwen, Spark, ChatGLM, etc. The results are shown in Figure 4. According to Figure 4, different LLMs have different performance in Minesweeper games. The winning rate of the Minesweeper Agent with ChatGLM is the best, while the winning rate of the Minesweeper Agent with Spark is the worst, always 0. On the other hand, ChatGLM takes the most steps to win a game, while the invalid steps of ChatGLM is the least. However, Spark takes the most invalid steps. The results show that different LLMs have different performance in Minesweeper games, and the Minesweeper Agent with ChatGLM has the best performance.

We also conduct an ablation study to evaluate the effectiveness of the prompt designs. We compare the Minesweeper Agent with different prompt designs, including Tips and Tricks, Step Back Prompting, Examples, Memory, etc. The results are shown in Table 1. According to Table 1, when there is only one mine, the Minesweeper Agent with Tips and Tricks has the best performance, with a winning rate of 1.0. The Minesweeper Agent with Memory has the best performance when there are two or three mines, with a winning rate of 0.3 and 0.1, respectively. One the other hand, the Minesweeper Agent with memory has the least invalid steps in all cases when there are more mines. The results demonstrate that the prompt designs can improve the performance of the Minesweeper Agent, and the memory design can help the Minesweeper Agent not to take invalid steps.

6 Disscussion

In this section, we discuss the results of the Minesweeper Agent, analyze the performance of the Minesweeper Agent, and point out the limitations of the Minesweeper Agent. We also propose some directions for future work.

6.1 Bad Cases Analysis

The Minesweeper Agent has some bad cases where it fails to win the game. We analyze some bad cases and provide explanations for the mistakes made by the Minesweeper Agent. As shown below, the Minesweeper Agent fails to win the game in some cases due to incorrect reasoning or decision-making. There are several reasons for the bad cases:

- **Incorrect Decision-making**: The Minesweeper Agent may make ineffective decisions based on the information it has observed. For example, in Bad Case 1, the Minesweeper Agent correctly identifies the cell at (7,5) as a mine but fails to flag the cell at (7,5), which leads to a mistake in the game.
- **Incorrect Reasoning**: The Minesweeper Agent may make incorrect reasoning based on the information it has observed. For example, in Bad Case 2, the Minesweeper Agent chooses to reveal the cell at (1,3), which is surrounded by ’0’ and ’1’ cells. This leads to a mistake in the game.
- **Invalid Steps**: The Minesweeper Agent may take invalid steps, such as repeatedly clicking on the same cell or making ineffective operations. For example, in Bad Case 3, the Minesweeper Agent repeatedly flags the cell at (7,2) as a mine, which is an invalid step.

The bad cases analysis provides insights into the performance of the Minesweeper Agent and helps identify the weaknesses of the model. By analyzing the bad cases, we can improve the reasoning ability of the Minesweeper Agent and enhance its performance in Minesweeper games.

6.2 Limitations and Future Work

The Minesweeper Agent has several limitations that need to be addressed in future work:

- **Limited Reasoning Ability**: The Minesweeper Agent has limited reasoning ability in complex Minesweeper puzzles. The model may fail to win the game in challenging scenarios that require advanced reasoning and decision-making.
- **Limited Generalization**: The Minesweeper Agent may have limited generalization ability in unseen Minesweeper puzzles. The model may not perform well in new environments or scenarios that are different from the training data.
- **Limited Efficiency**: The Minesweeper Agent may not be efficient in completing Minesweeper puzzles. The model may take more steps or make more invalid steps to win the game, which affects its performance.

To address these limitations, we propose several directions for future work:

- **Advanced Reasoning Models**: Develop advanced reasoning models that can handle complex Minesweeper puzzles with multiple mines and challenging scenarios. The models should be able to reason and make decisions based on the information they have observed.
- **Generalization Techniques**: Explore generalization techniques to improve the generalization ability of the Minesweeper Agent in unseen Minesweeper puzzles. The models should be able to adapt to new environments and scenarios. For example, the better prompt design can help the model generalize better.
- **Efficiency Optimization**: Optimize the efficiency of the Minesweeper Agent in completing Minesweeper puzzles. The models should take fewer steps and make fewer invalid steps to win the game, which improves their performance. For example, elaborate multi-step reasoning can be generalized to improve the efficiency of the model.

Bad Case 1:

```
    0 1 2 3 4 5 6 7 8
  + - - - - - - - - -
0 | 0 0 0 0 0 0 0 0 0
1 | 0 0 0 0 1 1 1 0 0
2 | 0 0 0 0 1 ? 1 0 0
3 | 0 0 0 0 1 1 1 0 0
4 | 0 0 0 0 0 0 0 0 0
5 | 0 0 0 0 0 0 0 0 0
6 | 0 0 0 0 1 1 1 0 0
7 | 0 0 0 0 1 ? 1 0 0
8 | 0 0 0 0 1 ? 1 0 0
```

Answer: Answer: r,7,5

Reason: The cell at row 7, column 5 is surrounded by two cells with the number ’1’. According to the Minesweeper rules, if two adjacent cells with numbers have only one common adjacent unrevealed cell between them, that cell must be a mine. In this case, the cells at (7,4) and (7,6) both have the number ’1’, and the only common adjacent unrevealed cell between them is at (7,5). Therefore, we can safely flag the cell at (7,5) as a mine.

Bad Case 2:

```
    0 1 2 3 4 5 6 7 8
  + - - - - - - - - -
0 | 0 0 1 1 1 0 0 0 0
1 | 0 0 1 ? 1 0 0 0 0
2 | 0 0 1 1 1 0 0 0 0
3 | 0 0 0 0 0 0 0 0 0
4 | 0 0 0 0 0 1 1 1 0
5 | 0 0 0 0 0 1 ? 1 0
6 | 0 0 0 0 0 1 1 1 0
7 | 0 0 0 0 0 0 0 0 0
8 | 0 0 0 0 0 0 0 0 0
```

Answer: The unrevealed areas are (1,3) and (5,6). The cell at (5,6) is adjacent to two ’1’ cells, which means there is a possibility that it could be a mine. However, the cell at (1,3) is surrounded by ’0’ and ’1’ cells, indicating that it is safer to reveal. To minimize the risk, I will choose to reveal the cell at (1,3). r,1,3

Bad Case 3:

Answer: The cell at (6,2) has a ’2’, which means there are two mines adjacent to it. The only two unrevealed cells adjacent to (6,2) are (7,1) and (7,2). Therefore, both of these cells must be mines. We should flag them to avoid clicking on them.
Answer: f,7,2

Answer: The cell at (6,2) has a ’2’, which means there are two mines adjacent to it. The only two unrevealed cells adjacent to (6,2) are (7,1) and (7,2). Therefore, both of these cells must be mines. We should flag them to avoid clicking on them.
Answer: f,7,2
...

7 Conclusion

In this paper, we propose a new task to evaluate the reasoning ability of Large Language Models: Minesweeper Agent. With elaborate design of the environment and prompts, we evaluate the reasoning ability of several popular LLMs on the Minesweeper Agent task. The results show that the reasoning ability of LLMs in Minesweeper games still needs to be improved. Our work provides a new perspective to evaluate the reasoning ability of LLMs and provides some feasible directions for the development of agent systems in the future.

References

[Bai *et al.*, 2023] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[Brown *et al.*, 2020] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[Chen *et al.*, 2023] Long Chen, Oleg Sinavski, Jan Hünermann, Alice Karnsund, Andrew James Willmott, Danny Birch, Daniel Maund, and Jamie Shotton. Driving with llms: Fusing object-level vector modality for explainable autonomous driving. *arXiv preprint arXiv:2310.01957*, 2023.

[Devlin *et al.*, 2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[Du *et al.*, 2022] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. Glm: General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335, 2022.

[Johnson *et al.*, 2017] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2901–2910, 2017.

[Kannan *et al.*, 2023] Shyam Sundar Kannan, Vishnunandan LN Venkatesh, and Byung-Cheol Min. Smart-llm: Smart multi-agent robot task planning using large language models. *arXiv preprint arXiv:2309.10062*, 2023.

[Lin *et al.*, 2024] Mengying Lin, Yaran Chen, Dongbin Zhao, and Zhaoran Wang. Advancing object goal navigation through llm-enhanced object affinities transfer. *arXiv preprint arXiv:2403.09971*, 2024.

[Mirowski *et al.*, 2018] Piotr Mirowski, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, Raia Hadsell, et al. Learning to navigate in cities without a map. *Advances in neural information processing systems*, 31, 2018.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *International conference on machine learning*, pages 1928–1937, 2016.

[Narasimhan *et al.*, 2015] Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. *arXiv preprint arXiv:1506.08941*, 2015.

[Radford *et al.*, 2018] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

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

[Schick and Schütze, 2020a] Timo Schick and Hinrich Schütze. Exploiting cloze questions for few shot text classification and natural language inference. *arXiv preprint arXiv:2001.07676*, 2020.

[Schick and Schütze, 2020b] Timo Schick and Hinrich Schütze. It's not just size that matters: Small language models are also few-shot learners. *arXiv preprint arXiv:2009.07118*, 2020.

[Schick and Schütze, 2021] Timo Schick and Hinrich Schütze. Few-shot text generation with natural language instructions. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 390–402, 2021.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[Song *et al.*, 2023] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2998–3009, 2023.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[Wei *et al.*, 2022] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[Weston *et al.*, 2015] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart Van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

[Yang *et al.*, 2019] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

[Zheng *et al.*, 2023] Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H Chi, Quoc V Le, and Denny Zhou. Take a step back: Evoking reasoning via abstraction in large language models. *arXiv preprint arXiv:2310.06117*, 2023.

[Zhou *et al.*, 2022] Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. Learning to prompt for vision-language models. *International Journal of Computer Vision*, 130(9):2337–2348, 2022.

A Details of Feedback

The feedback in the Minesweeper Agent is designed to provide information to the model about the correctness of its operations. The feedback consists of several types, including valid operation, invalid operation, and end. The feedback is triggered by specific conditions, such as revealing an area without a mine, flagging an area as a mine, removing a flag from an area, winning the game, revealing a mine, etc. The feedback is used to guide the model to reason and make decisions based on the information it has observed. The details of the feedback in the Minesweeper Agent are shown in Table 2.

Table 2: Details of the feedback in the Minesweeper Agent.

Feedback	Type	Trigger Condition
safe.	Valid Operation	The agent reveals an area without a mine.
add flag.	Valid Operation	The agent flags an area as a mine.
remove flag.	Valid Operation	The agent removes a flag from an area.
win.	End	The agent flags all mines and reveals all safe areas.
game over.	End	The agent reveals a mine.
invalid answer, answer should be in the form of triples (action,x,y).	Invalid Operation	The agent's answer cannot be parsed into a triple.
invalid opsition, out of range.	Invalid Operation	The position of the answer is off the map.
invalid action, action shoule be r or f.	Invalid Operation	The action of the answer is not 'r' or 'f'.
invalid action, the area is already revealed.	Invalid Operation	The agent tries to reveal or flag an area that has been revealed.
invalid operation, the number of flags exceeds the number of mines.	Invalid Operation	The agent tries to flag more areas than the number of mines.

B Prompt Design

Prompt Engineering is a crucial part of fine-tuning large language models. With carefully designed prompts, we can guide the model to perform specific tasks. In the Minesweeper Agent task, we design a set of prompts to guide the model to play Minesweeper games. The prompt consists of 3 parts: prompt before, environemnt information and prompt after. Here are the introdution of the 3 parts of prompt.

Prompt Before:

```
In Minesweeper , hidden mines are scattered throughout a board, which is divided
   into areas.
The rows are seperated by newlines, and columns by space.
The board is structured as a table, with the first row and column labeled using
   numbers in double quotation marks to indicate row and column indices.
Areas have multiple possible states:
- Unrevealed areas (represented by '?', which cover the board at the start of the
   game, can also be made by removing flags)
- Numbered areas (represented by '0' to '8', which indicate the number of mines in
   the eight neighboring cells, including those diagonally adjacent)
- Flagged areas (represented by 'F', which are marked by the player to indicate a
   potential mine location)

A player selects a cell to open it. If a player opens a cell containing a mine, the
   game ends in a loss.
Otherwise, the opened cell displays either a number, indicating the number of mines
   diagonally and/or adjacent to it,
or a blank tile, and all adjacent cells will automatically be opened.
To win a game of Minesweeper, all non-mine cells must be opened without opening a
   mine.

Here are some tips and tricks for playing Minesweeper:
- Start at the corners: Begin by clicking on one of the four corner cells. There's a
    1/3 chance of not hitting a mine, and it helps open up more of the board.
```

```
- Use the numbers: Each number tells you how many mines are adjacent (including
   diagonally) to that cell. Use this information to logically deduce where mines
   are or aren't.
- The 1-2-3 Rule: When you see a sequence like 1-2-3, the '3' must be surrounded by
   all three mines. This can help you clear the surrounding cells safely.
- Chord Method: If two adjacent cells with numbers have only one common adjacent
   unrevealed cell between them, that cell must be a mine. This is because the
   numbers account for all the mines around them, and the overlap indicates a mine.
- Flagging: As soon as you identify a mine, flag it immediately. This helps keep
   track and avoids accidental clicks.
- Safely open empty areas: When you find an area surrounded by numbered cells that
   account for all adjacent mines, you can safely open all cells around it without
   clicking. This is called "chaining" or "bracketing".
- Patterns: Learn common patterns like "2-1-1", where a '2' is next to a '1',
   indicating the third cell in line must be a mine, as the '2' accounts for the
   '1' and the mine.
- Practice and Patience: Minesweeper is a game of logic and patience. The more you
   play, the better you'll get at recognizing patterns and making quick decisions.
- Take calculated risks: Sometimes, especially in advanced levels, you might need to
    make an educated guess. Weigh the probabilities before clicking, and remember,
   every game is learnable from mistakes.

Below is a Minesweeper board:
```

A example of Environment Information:

```
   0 1 2 3 4 5 6 7 8
  + - - - - - - - - -
0 | ? ? ? ? ? ? ? ? ?
1 | ? ? ? ? ? ? ? ? ?
2 | ? ? ? ? ? ? ? ? ?
3 | ? ? ? ? ? ? ? ? ?
4 | ? ? ? ? ? ? ? ? ?
5 | ? ? ? ? ? ? ? ? ?
6 | ? ? ? ? ? ? ? ? ?
7 | ? ? ? ? ? ? ? ? ?
8 | ? ? ? ? ? ? ? ? ?
Mines rest: 1
Revealed areas:
Unrevealed areas: (0,0), (0,1), (0,2), (0,3), (0,4), (0,5), (0,6), (0,7), (0,8),
    (1,0), (1,1), (1,2), (1,3), (1,4), (1,5), (1,6), (1,7), (1,8), (2,0), (2,1),
    (2,2), (2,3), (2,4), (2,5), (2,6), (2,7), (2,8), (3,0), (3,1), (3,2), (3,3),
    (3,4), (3,5), (3,6), (3,7), (3,8), (4,0), (4,1), (4,2), (4,3), (4,4), (4,5),
    (4,6), (4,7), (4,8), (5,0), (5,1), (5,2), (5,3), (5,4), (5,5), (5,6), (5,7),
    (5,8), (6,0), (6,1), (6,2), (6,3), (6,4), (6,5), (6,6), (6,7), (6,8), (7,0),
    (7,1), (7,2), (7,3), (7,4), (7,5), (7,6), (7,7), (7,8), (8,0), (8,1), (8,2),
    (8,3), (8,4), (8,5), (8,6), (8,7), (8,8),
Flagged areas:
```

Prompt After:

```
Let's step back and think, then give your answer.
Please explain your reason first, and then answer (action,row,col) in the form of
   triples.
- "action" can be "r" (reveal) or "f" (flag or unflag),
- "row" and "col" are the row and column indices of the area, the upper left corner
   is (0,0).

For example, "r,1,2" means to reveal the area at row 1 and column 2,
```

```
and "f,3,4" means to flag the area at row 3 and column 4, if the area is flagged, it
    will be unflagged.
Now, please enter the action, choose an unrevealed area to reveal or flag.
Only one action is allowed at a time. The answer must be in the form of triples,
    separated by commas, and do not add spaces.
```