LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

CENTRUM FÜR INFORMATIONS- UND SPRACHVERARBEITUNG
STUDIENGANG COMPUTERLINGUISTIK

Bachelor's Thesis

in the field of Computational Linguistics

at Ludwig Maximilian University of Munich

Faculty of Language and Literature

Spatial Reasoning in LLMs

submitted by
Arsenii Kvachan

Supervisor:	PD Dr. Helmut Schmid
Examiner:	PD Dr. Helmut Schmid
Processing period:	April 2 - June 4, 2024

Declaration of Authorship

I hereby declare that I have independently authored this work, marked all quotes as such, and have indicated all sources and aids used.

Munich, June 4, 2024

. . . .
Arsenii Kvachan

Abstract [EN]

With language models becoming powerful assistants in various real-world scenarios, their ability to understand and follow spatial instructions has become crucial. This thesis investigates the spatial reasoning capabilities of current state-of-the-art language models through a series of controlled experiments and proposes a theoretical foundation for enhancing these capabilities through various prompt engineering techniques and multi-modality. Evaluation will be approached systematically by first intelligently prompting the models and assessing the quality of the response as well as its success in the environment. The first experiment is conducted on an LLM (GPT-4o). Subsequently, a different, smaller language model (Mistral-7B-Instruct) is fine-tuned. Finally, performance with the vision modality of GPT-4o is evaluated on a series of spatial tasks, such as path planning and instruction following. Experiments showed that just by deploying prompt engineering, the LLMs behave surprisingly well. This aligns with past research on the reasoning capabilities of language models. However, the thesis can not confirm that a few-shot prompting necessarily constitutes a better performance. Fine-tuning the model did not show ideal results, but a significant improvement over a non-fine-tuned model could be observed. Language models showed promising results when dealing with complex tasks that require spatial understanding and reasoning.

Abstract [DE]

Sprachmodelle werden zu leistungsfähigen Assistenten in verschiedenen realen Szenarien. Ihre Fähigkeit, räumliche Anweisungen zu verstehen und zu befolgen ist entscheidend. Diese Arbeit untersucht die räumlichen Fähigkeiten aktueller Sprachmodelle durch eine Reihe von Experimenten und schlägt eine theoretische Grundlage für die Verbesserung dieser Fähigkeiten durch verschiedene Prompting-Techniken und Fine-Tuning vor. Die Evaluierung wird systematisch angegangen, indem die Modelle zunächst auf intelligente Weise geprompted werden und die Qualität der Antwort sowie ihr Erfolg in der Umgebung bewertet wird. Das erste Experiment wird mit einem LLM (GPT-4o) durchgeführt. Anschließend wird ein anderes, kleineres Sprachmodell (Mistral-7B-Instruct) fine-tuned. Schließlich wird die Leistung mit der Sehmodalität von GPT-4o bei einer Reihe von räumlichen Aufgaben, wie z. B. der Wegplanung und dem Befolgen von Anweisungen, bewertet. Die Experimente haben gezeigt, dass sich die LLMs allein durch den Einsatz von Prompt-Engineering erstaunlich gut verhalten. Dies deckt sich mit früheren Forschungen zu den Schlussfolgerungsfähigkeiten von Sprachmodellen. Die Arbeit kann jedoch nicht bestätigen, dass ein Prompting mit nur wenigen Schüssen zwangsläufig eine bessere Leistung bedeutet. Das Fine-Tuning des Modells führte zwar nicht zu idealen Ergebnissen, aber es konnte eine deutliche Verbesserung gegenüber einem nicht fine-tuned Modell festgestellt werden. Sprachmodelle zeigten vielversprechende Ergebnisse bei der Bewältigung komplexer Aufgaben, die räumliches Verständnis und logisches Denken erfordern.

Contents

Abstract [EN]

Abstract [DE]

1 Introduction 2

2 Related work 4

3 Tasks 8
3.1 Short-term Path Planning (STPP) 8
3.2 Spatial Actions Planning (SAP) 8

4 Environments 9
4.1 FrozenLake 9
4.1.1 LLM-FrozenLake Interaction 10
4.2 PutNext 11
4.2.1 LLM-PutNext Interaction 12

5 Language Models 14
5.1 GPT-4o 14
5.2 Mistral-7b-Instruct 14
5.2.1 Fine-tuning 15
5.2.2 Data Generation 15

6 Prompting Strategies 17
6.1 Chain-of-Thought (CoT) 17
6.2 Tree-of-Thought (ToT) 18
6.3 Prompt with Actor-Critic Editing (PACE) 19
6.4 Prompt Templates 20

7 Experiments 21
7.1 Overview 21
7.2 Structure 22
7.2.1 Libraries and Dependencies 22
7.2.2 Experiment Class 22
7.2.3 Execution Flow 22
7.2.4 Running Experiments 23

8 Results 24

9 Discussion 27

10 Conclusions 29

11 Materials 30

References 34

1 Introduction

Language Models
Language models (LMs) are probabilistic models of natural language. LMs can be used for various tasks such as machine translation, natural language generation, grammar induction, and information retrieval. Among the most advanced forms of LMs are large language models (LLMs), which are trained on vast datasets ranging from gigabytes to petabytes and contain billions or even parameters. Currently, the GPT-4 LLM from OpenAI leads in several benchmarks and human preference assessments. One of LLMs' most important and interesting features is that they can resemble human-like reasoning.

Spatial Reasoning
In this thesis, we will evaluate the spatial reasoning abilities of LLMs. Spatial reasoning is the capacity to understand, reason, and remember the visual and spatial relations among objects or spaces. Spatial reasoning has evolutionary and adaptive importance. Cognitive and neural evidence suggests that humans can learn spatial structure solely from sequences of observations. This raises a question: Can LLMs learn spatial reasoning from textual inputs?

State of Research
The current state of research in spatial reasoning of LLMs is primarily focused on relatively simple environments. e.g. simple 2D or 3D Cartesian spaces, and relatively simple tasks, e.g. path planning, shape identification, and spatial relationship identification. These tasks can be decomposed in the step-by-step procedures using few-shot Chain-of-Thought prompting. Most research suggests that LLMs implicitly learn to represent aspects of spatial structure, but their performance on the before-mentioned tasks is not ideal. Multiple benchmarks for assessing the spatial reasoning abilities of LLMs were adapted or proposed, e.g. StepGame and Path Planning from Natural Language (PPNL). These benchmarks are yet to be integrated with other common benchmarks. Overall, the domain of spatial reasoning remains largely an under-explored realm ripe for more research, especially regarding multi-modal LLMs and spatial reasoning datasets.

Thesis Focus
In this thesis, we focus on summarizing the current research in the area of spatial reasoning and confirming some of the findings through a series of controlled experiments. First, GPT-4o is evaluated on a path-planning task in a grid environment using Naïve Prompting, Chain-of-Thought, Tree-of-Thought, and PACE. Second, Mistral-7b-Instruct is fine-tuned and evaluated for the same path-planning task. Third, GPT-4o (with vision) is evaluated on the spatial actions planning task in a 3D world environment. A logical question rises, does fine-tuning enhance spatial reasoning abilities of LLMs?
The thesis consists of four parts: Related Work - provides detailed insights into current research and summarizes all up-to-date findings; Tasks - explains the types of tasks covered by experiments; Environments - delves into the FrozenLake and PutNext environments and their interaction with the LLM; Prompting Strategies - explains Chain-of-Thought, Tree-of-Thought, and REACT Prompting. Experiments - provides an overview of the combinations of Tasks, Environments, Language Models, and Prompting Strategies evaluated, and defines the exact structure and procedures in the API; Results - presents final results; Discussion - addresses problems, expectations, and findings in the related research; Conclusions - summarizes the thesis and presents final conclusions.

2 Related work

LLMs and Trajectory Labeling
One of the most prominent works on spatial reasoning in LLMs is "Exploring and Improving the Spatial Reasoning Abilities of Large Language Models" by M. Sharma (December 2023) [20]. This paper focuses on evaluating the performance of LLMs on 3D robotic trajectory data and associated tasks like 2D directional and shape labeling. To significantly enhance the performance of the LLMs, a new prefix-based prompting mechanism, Spatial Prefix-Prompting (SPP), was proposed. The key idea is to present a simple, tangentially related spatial problem followed by the main query. For instance, if the task is to label the overall tendency of a trajectory as lifting, rotating, or sliding, the model is first prompted to identify the direction of the movement, such as left, up, right, or down. Then, using the directional information, the model completes the adjacent task—trajectory labeling. This prompting technique shares some similarities with the Tree of Thought prompting [30], which will be discussed later in the thesis. This technique improves the trajectory labeling accuracy on the CALVIN-Cleaned dataset by 33% compared to the naïve zero-shot prompting of GPT-4. Sharma concludes that LLMs are capable of general spatial pattern matching but face challenges with more complex 3D trajectory data. The author emphasizes the need to explore larger datasets, other spatial tasks, and additional LLMs, besides GPT-3.5 and GPT-4, to further understand and improve these capabilities.

LLMs and Path Planning
Another intriguing paper titled "Can Large Language Models Be Good Path Planners? A Benchmark and Investigation on Spatial-Temporal Reasoning" by M. Aghzal et al. (2023) [2], investigates the performance of LLMs when navigating grid environments. The authors introduce a new benchmark called Path Planning from Natural Language (PPNL) for assessing the path-planning abilities of LLMs. PPNL includes several important metrics that will be used in this thesis, such as success rate, optimal rate, and survival rate. These metrics will be explained in chapter 8 (Results). Aghzal et al. explore several prompting approaches for GPT-4, including naïve few-shot prompting, action-and-effect prompting, Chain-of-Thought [28] prompting, and ReAct prompting [31]. They compare these prompting strategies using various numbers of shots and conclude that the ReAct approach yields the highest success rate — 96.1%. Lastly, the authors fine-tuned BART and T5 models on the (long-term) path-planning data and found that the models perform well on in-distribution tasks but have difficulty generalizing to larger or more complex environments, a finding that will be confirmed by this thesis. Recent investigations into the effects of fine-tuning on LLMs also suggest that fine-tuning increases the likelihood of hallucinations and overfitting. Moreover, excessive fine-tuning causes the forgetting of the original training, which also negatively impacts the performance and flexibility of the model, as the paper "Investigating Catastrophic Forgetting in Multimodal Large Language Models" by Y. Zhai et al. (2023) [32] found.

LLMs and Spatial Relation Identification
The paper "Advancing Spatial Reasoning in Large Language Models: An In-Depth Evaluation and Enhancement Using the StepGame Benchmark" by F. Li et al. (2024) [12] evaluates the spatial reasoning capabilities of LLMs using the StepGame benchmark [22]. The authors investigate why LLMs have previously shown unsatisfactory performance on the StepGame benchmark and identify that template errors affect evaluation results. The StepGame benchmark evaluates reasoning on 1-Hop and Multi-Hop relations. A hop is a relation between two objects; for example, "J is diagonally above B to the right at a 45-degree angle" is a 1-Hop relation where J and B are objects. A model is given multiple such relations and should answer questions regarding one or more objects in these relations. The authors create an error-free benchmark by employing a semantic parser called ASP, that was initially presented in the paper "Neurasp: Embracing neural networks into answer set programming" by Yang et al. (2023) [29]. They then proceed to customize Chain-of-Thought and Tree-of-Thought prompting for the StepGame benchmark. Generally, an exponential decay of accuracy can be observed with a higher number of Hops, indicating that even the most powerful GPT model (GPT-4) struggles to maintain accuracy as task complexity rises.

LLMs and Prompting Strategies
In this thesis, three prompting strategies will be used: Chain-of-Thought (CoT), Tree-of-Thought (ToT), and Prompt with Actor-Critic Editing (PACE). Chain-of-Thought (CoT) was first proposed in the paper "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models" by J. Wei et al. from the Google Research Brain Team (August 2023) [28]. The main idea is to decompose a task into a series of intermediate reasoning steps. This simple prompting approach significantly enhances the ability of LLMs to perform complex tasks. The key finding is that by providing demonstrations that include the reasoning steps, the reasoning capabilities of LLMs emerge naturally. Regarding arithmetic reasoning, which is closely related to spatial reasoning, CoT outperformed naïve prompting on the GSM8K, SVAMP, and MAWPS benchmarks (PaLM 540B LLM). Since then, the approach has been modified to include trigger phrases such as "Let's think step-by-step" or "Let's solve this problem by splitting it into steps" [11], which also significantly boosted performance.
Tree-of-Thought (ToT) is a relatively novel approach that takes Chain-of-Thought (CoT) even further. The paper "Large Language Model Guided Tree-of-Thought" by J. Long (May 2023) [15] introduced this prompting technique as an alternative to the linear CoT. ToT prompts the model multiple times sequentially; at each step, the model considers a subset of all possible solutions, creating a tree. If the leaf solution does not yield the expected results, backtracking and potentially new generation are performed. ToT is a tree-search algorithm using an LLM as a heuristic for generating the search steps. The LLM is engaged in a multi-round conversation with the ToT system, which guides the LLM to a local optimum. If the local optimum is unsatisfactory, the ToT system may prompt the model again by backtracking and starting from an upper node in the tree. The ToT system consists of interconnected modules, including a checker module, memory module, ToT controller, and prompter agent. It is used to automatically control the flow of prompts and responses. The exact architecture of the system will be omitted in this thesis; instead, a much simpler, multi-turn conversation with some checks and tricks (like ranking solutions) will be used. The author deployed ToT on the Sudoku puzzle-solving task. The evaluation showed a remarkable success rate of 80% on the 5x5 puzzles, which is 30% higher than using few-shot CoT prompting on the same set of puzzles. One disadvantage of this approach is that it requires a significant amount of computation and context length.

Prompt with Actor-Critic Editing (PACE), presented in the paper ”PACE: Improving Prompt with Actor-Critic Editing for Large Language Model” by Dong et al. (2023) [6], utilizes two types of LLMs: Actors and Critics. Actor generates actions, while Critic evaluates the actions to yield a critique. Action is defined as a function of $a = f([p; X], M)$, where p is the prompt, X is the input, M is the LLM, while p and X will be inserted into the correponding template. Critique is a function $c = f([p; X; a; Y], M)$, where a is the actions of Actor and Y is the desired output. The prompting technique is tested on the Instruction Induction benchmark, where model is prompted on a series of simple tasks, for example to generate antonyms or convert singular to plural. In many cases, the LLMs using the PACE-refined prompts achieved performance levels comparable to, and in some cases even surpassing, those using high-quality human-written prompts, i.e., B-HWP or W-HWP. The best average score is 80%.

LLM Agents

LLM Agents can create and execute plans using reasoning and external tools. Paradigms from the survey ”Understanding the Planning of LLM Agents: A Survey” by X. Huang et al. (2024) [9] will be utilized to maximize the success rate of the LLM agent. The survey covers recent works that improve the planning abilities of LLM-based agents. The existing work is categorized into five approaches: Task Decomposition, Multi-Plan Selection, External Module-Aided Planning, Reflection and Refinement, and Memory-Augmented Planning.

Task Decomposition involves the creation of sub-tasks and plans for each sub-task. Example methods include Chain-of-Thought, ReAct, and HuggingGPT [21]. Multi-Plan Selection generates many plans and selects the best one using search algorithms. Examples are Tree-of-Thought and Self-consistency. External Module-Aided Planning, such as LLM+P [14] or LLM-DP [5], will not be utilized in this thesis. For instance, the main idea of LLM+P is to use an LLM to translate a natural language task into a computer-readable planning problem format and then leverage classical planners to quickly find a solution.

Reflection and Refinement encourages LLMs to reflect on failures and refine plans. Methods include PACE (discussed previously), Reflexion [23], Self-refine [16], and CRITIC [7]. Memory-Augmented Planning utilizes memory modules to store valuable information. Examples are Generative Agents [17], MemoryBank [34], and SwiftSage [13]. Some form of memory was initially planned for use in the project to efficiently remember sequences of previous actions but was never implemented. Memory could solve some issues regarding repetitive moves and remembering what solutions worked under similar circumstances. More on these issues will be discussed later.

VLMs and Spatial Reasoning

Vision Language Models (VLMs) are an extension of LLMs. They leverage vision alongside natural language to reason on visual data, such as images. Understanding and reasoning about spatial relationships is a fundamental capability for Visual Question Answering (VQA). However, VLMs are limited in their ability to estimate size, width, depth, orientation, and positioning. The paper ”SpatialVLM: Endowing Vision-Language Models with Spatial Reasoning Capabilities” by B. Chen et al. from Google DeepMind (2024) [3] hypothesizes that VLMs’ limited spatial reasoning capability is due to the lack of 3D spatial knowledge in training data. The authors aim to solve this problem by training VLMs with Internet-scale spatial reasoning data. They investigate various factors in the training recipe, including data quality, training pipeline, and VLM architecture. The main focus lies on infusing spatial reasoning capabilities into VLMs by pretraining with synthetic data. VLMs pretrained via SpatialVLM are demonstrated to be useful for robotics tasks, where they can be used as reward annotators.

VLMs can significantly advance the field of Embodied AI, where vision-and-language navigation requires agents to navigate in diverse and unseen environments following free-form linguistic instructions. The paper "NaVid: Video-based VLM Plans the Next Step for Vision-and-Language Navigation" by J. Zhang et al. (2024) [33] proposes a video-based VLM, NaVid, to accomplish this task. The VLM was trained on approximately 510k navigation samples from continuous environments (such as rooms) and 736k web data. The architecture consists of a Vision Encoder, Query Generator, LLM, and Cross-Modality Projectors. The Vision Encoder takes a video frame and converts it into visual embeddings. The Query Generator generates task-specific queries to extract visual features from the embeddings. The LLM processes both the visual and textual tokens to interpret the query and visual embeddings and predict the next action. The Cross-Modality Projector embeds visual and textual information into a shared space for effective interaction. NaVid achieved state-of-the-art performance on the VLN-CE R2R dataset and demonstrated robustness in real-world indoor environments, achieving a 66% success rate across diverse scenarios.

Spatial Environments
A library of choice for the realization of spatial environments is Gymnasium [26], with alternatives being HabitatSim [18, 24, 19] and PettingZoo [25]. Game environments in Gymnasium are modeled in a standardized manner, which allows for easier compatibility of experiments and algorithms between the environments. In this thesis, a 2D grid environment, FrozenLake, and a 3D continuous environment, PutNext, will be used for experiments. These environments are ideal for evaluating spatial reasoning tasks since they provide real-time multi-modal feedback based on the agent's actions, including textual and visual feedback.
In gaming, the most widely used environment is currently Minecraft, where the agent needs to gather materials to create tools for obtaining more rewards. The quantity of tools created by the agent is often used as an evaluation metric. Another popular category is text-based interactive environments, such as ALFWorld [4] and ScienceWorld [27], where the agent operates in an environment described in natural language, with limited actions and locations. Compared to Minecraft, these text-based interactive environments are often simpler, with straightforward feedback and fewer feasible actions, as is the case in this thesis. Unity-based environments are also commonly used. The success rate or the rewards obtained are commonly used as evaluation metrics.

3 Tasks

Two types of tasks will be evaluated: Short-term Path Planning (STPP) and Actions Planning (SAP).

3.1 Short-term Path Planning (STPP)

A planning is considered short-term if, for each observation, the number of predicted actions $|\mathcal{A}|$ equals 1. Short-term planning is a subset of long-term planning and can be useful in scenarios where the topology of the environment is not too complex and the action space is relatively small. A task will be approached step-by-step without considering long-range dependencies. In path planning, only navigational actions, such as "left," "right," "up," and "down," are considered. This approach, while being naïve, simplifies the planning for an agent. SAP, on the other hand, allows non-navigational actions, such as "take," "put," "destroy," "consume," and so on, alongside navigational actions. Metrics for STPP can be seen below. They are primarily motivated by the metrics used in the PPNL benchmark.

Metrics:

- Success Rate - % of successful runs
- Optimal Rate - % of optimal runs (according to some conditions)
- Survival Rate - % of failed runs where the agent did not collide with an obstacle
- Avg. Last Distance - Mean last distance across all runs
- Avg. Escapes - Mean number of times an agent escaped a so-called trap. A trap is considered a position out of which there is only one exit. Escaping the trap means previously being in this position

3.2 Spatial Actions Planning (SAP)

Tasks are defined for specific environments. A task is considered successful if the agent completes all goals under certain constraints, if they exist. A task is considered partially successful if only some goals are completed or some constraints are fulfilled. For instance, in the FrozenLake environment, there is only one goal: find a path from the origin tile to the goal tile without touching the obstacles. In the PutNext environment, there can be multiple goals, such as fetching an object and putting it next to another object while doing it in the least number of steps possible. Metrics for SAP are similar to STPP and can be seen below.
Metrics:

- Success Rate - % of successful runs
- Optimal Rate - % of optimal runs (according to some conditions)
- Survival Rate - % collision rate in case of failure (always 100
- Partial Success Rate - % of runs where only some goals were completed

4 Environments

This chapter describes each environment in detail.

	FrozenLake	**PutNext**
Coordinate System	2D	3D
Discrete Space	✓	✗
Discrete Actions	✓	✓
Modifiable	✗	✓
Dynamic	✗	✓
Obstacles	✓	✗
Observation Shape	(1)	(80, 60, 3)
# Goals	1	≥ 1
# Actions	4	8

Table 4.1: Short comparison of FrozenLake and PutNext

4.1 FrozenLake

The FrozenLake environment is a 2D grid. It is very simple and is meant to evaluate the fundamental ability of an LLM to perform short-term path planning (STPP). FrozenLake is generated in three sizes: 4x4, 8x8, and 12x12. An agent can execute four navigational actions: (0) "left," (1) "right," (2) "up," and (3) "down." There is only one type of obstacle, which is a hole in the lake. The environment can be formally and completely described by the size k, goal tile G, start tile S, and a set of obstacles H. The agent's position is defined by coordinates (r, c), where r is the row, c is the column, and $(0, 0)$ is in the upper left corner. The observation is the number of the tile on which the agent is currently located. We will use coordinates instead of tile numbers as an observation for more convenient work with the grid. If the agent collides with an obstacle tile h from H, the run terminates immediately. There is always at least one unobstructed path p from P from S to G. Moreover, there can be multiple possible paths. If a path starts at S and ends at G and takes the minimal number of actions to complete, it is considered optimal. An example of the environment can be seen below. It is worth noting that a large number of maps were generated via Gymnasium and then they were automatically solved with A* algorithm to find the optimal path. This is how we know that agent behaved optimally.

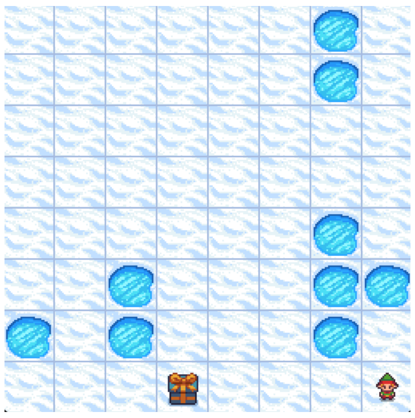

Figure 4.1: An 8x8 FrozenLake env. with the goal at (7,3) and origin at (7,7)

4.1.1 LLM-FrozenLake Interaction

The Fig. 4.2 illustrates the interaction between an LLM agent and the FrozenLake environment. The process flow is as follows:

1. The FrozenLake environment provides the current position.
2. The termination condition is checked. If the task is not terminated, the process continues.
3. The Observation Prompt component updates the agent with the current position.
4. The LLM agent receives the observation prompt and generates a response.
5. The Response Parser translates the LLM agent's response into an action.
6. The action is executed in the FrozenLake environment, updating the agent's position and continuing the cycle until termination.

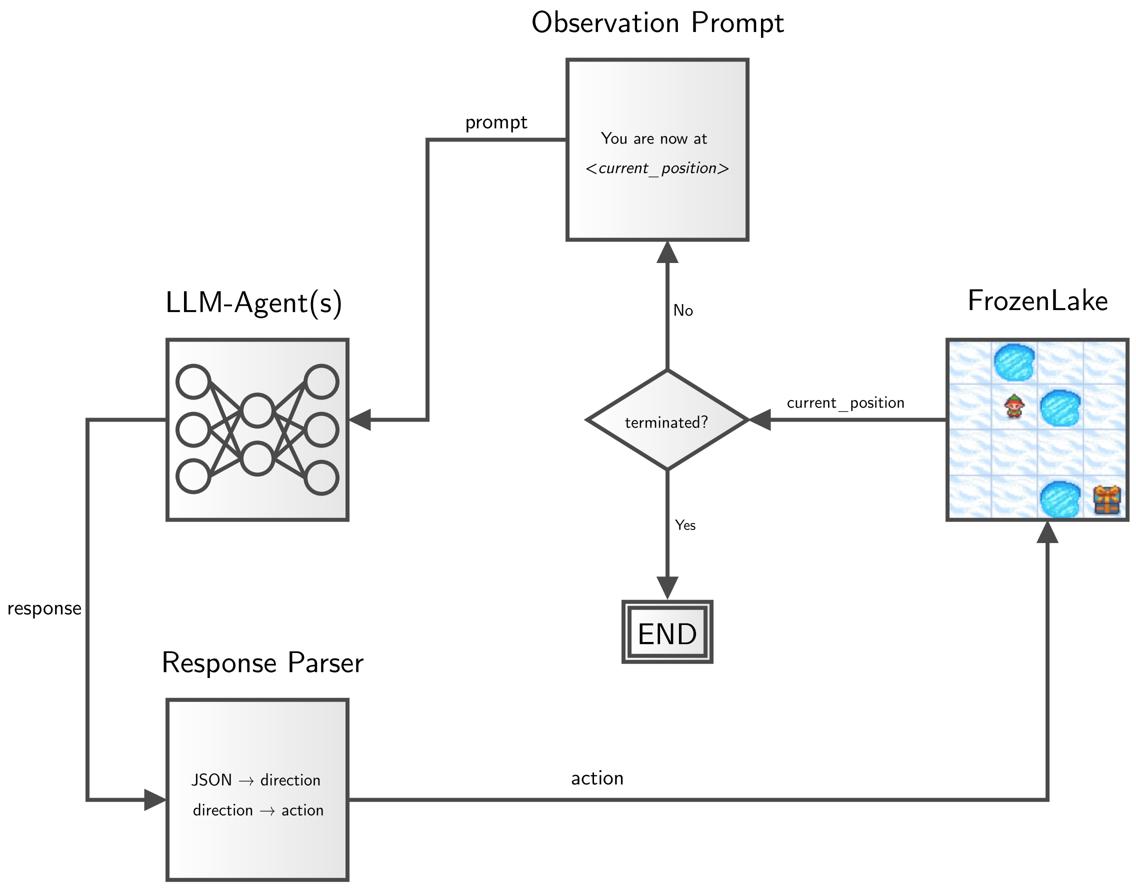

Figure 4.2: LLM-FrozenLake Interaction

4.2 PutNext

MiniWorld's PutNext environment will be used for the experiments. MiniWorld is a minimalistic 3D interior environment simulator for reinforcement learning robotics research. The world is made of static elements (rooms and hallways), as well as objects which may be dynamic. MiniWorld uses OpenGL's right-handed coordinate system. Agent's position is defined per coordinates (x, y, z, d), where x and z form ground plane, y axis points up and d is a angle of view. The observation that environment gives us at each step is an 80 by 60 pixel RGB image; each pixel is a value between 0 and 255. A top-down fully observable view of the environment can be produced as well, then our environment becomes a continous version of the grid world. PutNext does not have any dynamic objects (except the agent itself), nor any obstacles. The environment cannot be described as simply as FrozenLake, since the agent has only local view (initial image at the start). It is a single-room environment where a red box must be placed next to a yellow box. The goal is to perform this task in as few steps as possible. The avaliable actions in PutNext are (0) ”turn left”, (1) ”turn right”, (2) ”move forward”, (3) ”move back”, (4) ”pick up”, (5) ”drop”, (6) ”toggle” and (7) ”complete task”. An example observation image of PutNext can be viewed below.

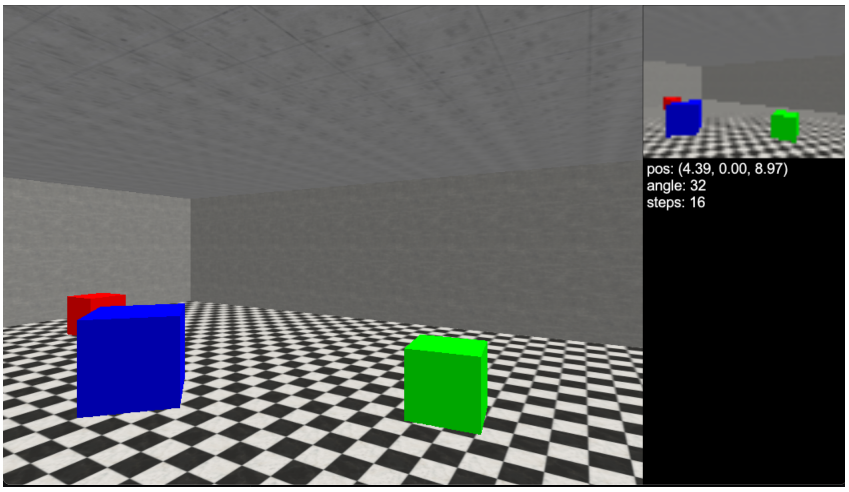

Figure 4.3: A 12 blocks large PutNext env. with the view at pos. (4.39, 0.00, 8,97, 32) on some colorful blocks that an agent can interact with

4.2.1 LLM-PutNext Interaction

The Fig. 4.2 illustrates the interaction between an LLM agent and the PutNext environment. The process flow is as follows:

1. The PutNext environment provides the current view.
2. The termination condition is checked. If the task is not terminated, the process continues.
3. The Observation Prompt component updates the agent with the current view and textual instructions.
4. The LLM agent receives the observation prompt and generates a response.
5. The Response Parser translates the LLM agent's response into an action.
6. The action is executed in the PutNext environment, updating the agent's view and continuing the cycle until termination.

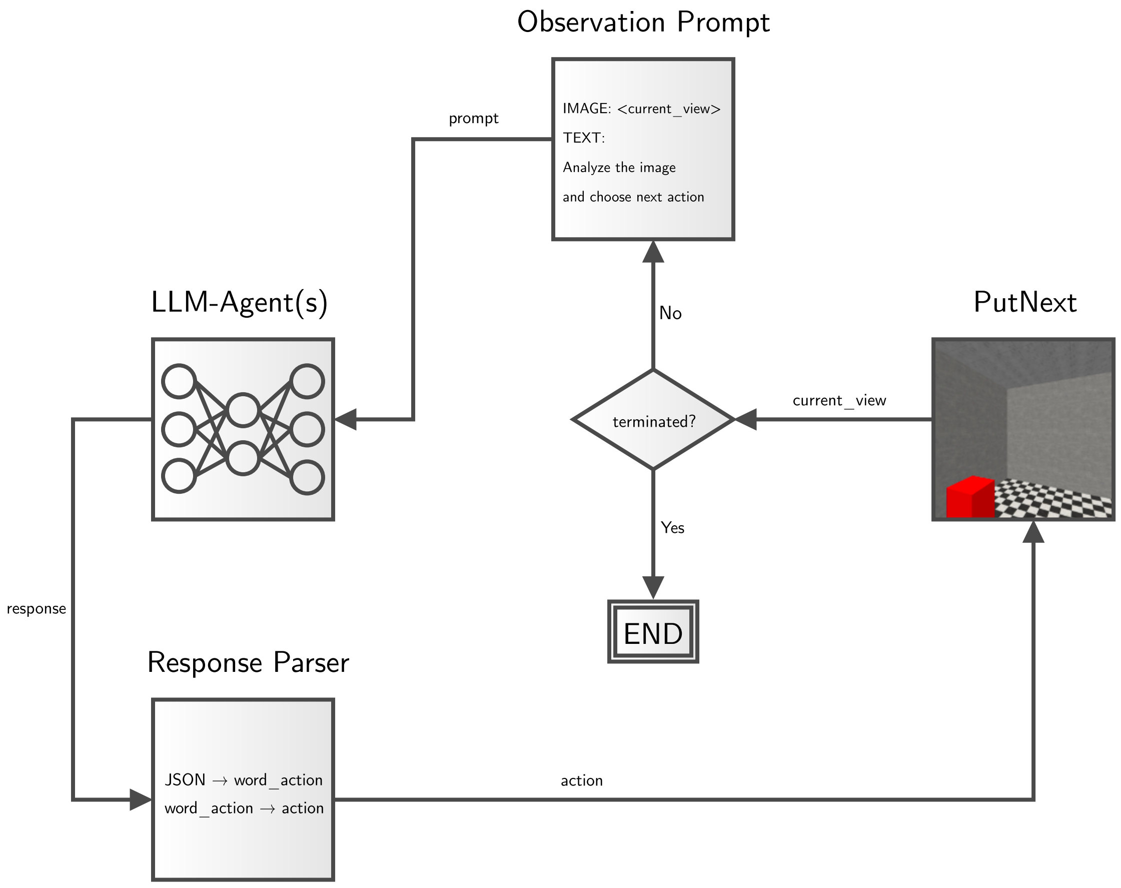

Figure 4.4: LLM-PutNext Interaction

5 Language Models

5.1 GPT-4o

Description and Performance
On May 13, 2024, OpenAI introduced GPT-4o, a new multi-modal, GPT-based LLM capable of faster and cheaper inference than its predecessor, GPT-4 Turbo. GPT-4o scored 88.7 on the 0-shot COT Massive Multitask Language Understanding (MMLU) benchmark, which is an improvement compared to the 86.5 scored by its predecessor, GPT-4 Turbo, according to the official website [1]. It has a context length of 128k tokens with an output limit of 2048 tokens. It does not support fine-tuning yet.

API
Inference is done via the OpenAI API. The model's response is limited to 2000 tokens and the temperature is set to 0.6. In cases where chat context is needed, the Assistants API is used to automatically handle the chaining of the messages. In other cases, the Chat Completions API is sufficient. Vector Stores are not used.

5.2 Mistral-7b-Instruct

Description and Performance
Mistral-7b-Instruct is an instruct fine-tuned version of Mistral-7b [10]. It is a 7.3B parameter language model using the transformers architecture. Mistral-7b outperforms Llama 2 13B across all evaluated benchmarks, and Llama 1 34B in reasoning, mathematics, and code generation. It is also highly efficient. It is easily and cheaply fine-tunable due to its size and open-source nature (Apache 2.0 license).

Inference
Inference is done locally, on a quantized and fine-tuned version of Mistral-7b-Instruct. The LLM is used via an API server running on localhost. Requests and responses follow OpenAI's API format. The temperature is set to 0.6 and tokens are limited to 2000. A short overview of the fine-tuning process can be viewed below.

5.2.1 Fine-tuning

Fine-tuning the model in full 16-bit precision is not achievable with the VRAM available for this thesis, so the base model is loaded in 4-bit precision and only then fine-tuned via Low-Rank Adaptation (LoRA) [8]. The fine-tuning process consists of the following steps:

1. Install and import all dependencies
2. Build the dataset from JSON files
3. Load the base model from HuggingFace
4. Load the tokenizer
5. Prepare the LoRA configuration with the hyperparameters
6. Train the adapter via LoRA (PEFT) as described in the definition
7. Save the adapter
8. Merge the adapter with the base model in full precision
9. Quantize the model to 4- and 5-bit precisions in GGUF format via `llama.cpp`
10. Push the quantized models to HuggingFace
11. Set up a local server via LM Studio to interact with the model

The exact fine-tuning process can be seen in the notebooks given in chapter 11. Inference was done in 5-bit precision. The model was fine-tuned for choosing the best next action in the FrozenLake environment. LoRA, as it was presented in the paper "Lora: Low-rank adaptation of large language models" by Hu et al. (2021) [8], approximates the adapter matrix of the transformer modules by training two low-rank matrices, A and B. The adapter matrix is then added to the pre-training matrix. Some of the hyperparameters are displayed below. N is the number of records or fine-tuning samples, α is the scaling factor for the weight matrices, which decides how much influence the matrices of the fine-tuned transformer modules have in comparison to the matrices of pre-trained modules, and r is the rank of the fine-tuned matrices before multiplication.

- $N = 410$
- $\alpha = 16$
- $r = 64$

5.2.2 Data Generation

One fine-tuning sample consists of a system message, observation prompt, and JSON response. The system message explains the exact situation to the agent, including map size, obstacle positions, goal position, origin position, and available actions. The observation prompt states the current position of the agent. The JSON response consists of an explanation of why the action is executed and the action itself. An example can be seen below.

```
sample =
'<s>'
+ '[INST]' + system_message + observation_prompt + '[/INST]' + response +
'</s>'
```

A new set of maps were generated for the fine-tuning, different from those that we will conduct an evaluation on. These **training** maps will be solved by GPT-4o using ToT prompting strategy. It means, that JSON-responses were generated solely by GPT-4o. Afterwards, only successful runs were filtered and used to construct the dataset in the format as above.

6 Prompting Strategies

A prompt in LLMs is the textual input provided by users to guide the model's output. Correctly prompting the LLMs has proven to enhance the performance on a variety of benchmarks. In this chapter prompting strategies used in the experiments will be explained.

6.1 Chain-of-Thought (CoT)

Chain-of-Thought (CoT) Prompting is a strategy which decomposes a task into a digestible, step-by-step executable format. Initially described in the paper "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models" by J. Wei et al. (2022) [28], CoT is based on making the implicit reasoning process of LLMs explicit. By outlining the steps required for reasoning, the model is directed closer to a logical and reasoned output.

CoT is programmed as a part of the System Message. This version of CoT consists of the trigger phrase, step-by-step reasoning guidance (decision rationale) and k examples. These k examples of question-answer pairs are added at the end of the System Message. Fig. 6.1 shows what information is added to the system message.

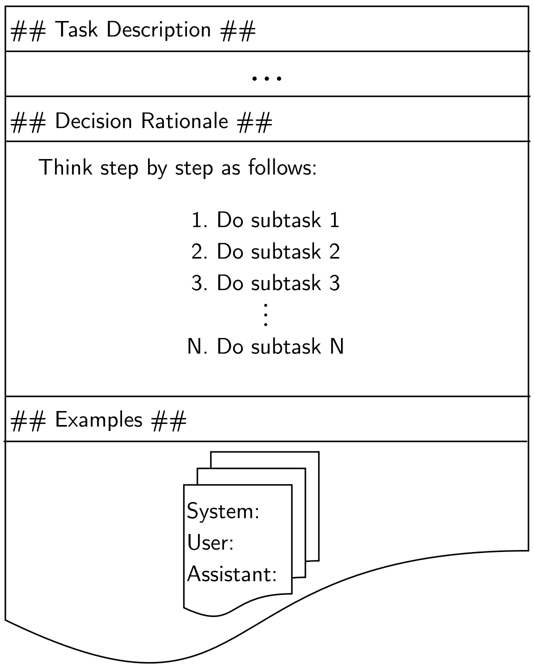

Figure 6.1: System Message with CoT, main feature is *## Decision Rationale ##*

For each step in the environment, the model is prompted with CoT. It receives an Observation Prompt that may contain current position, acquired items, images or other reasoning-relevant information. System Message, while remaining constant throughout the run, can be sent at each step if the model lacks support for the System Message. Mistral-7b-Instruct does not support System Message, so it should be sent each time with the Observation Prompt. Model reasons according to the CoT and gives a response, which may be parsed into a valid action in the environment. The detailed flow can be seen in Fig. 6.2.

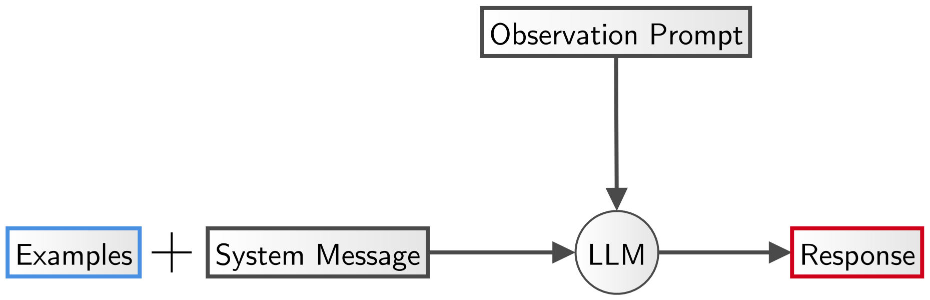

Figure 6.2: Flow, visualizing the interaction of components in CoT

6.2 Tree-of-Thought (ToT)

Tree-of-Thought (ToT) Prompting involves generating and developing thoughts. A thought is a coherent language sequence that serves as an intermediate step toward problem-solving. The creation of ToT consists of three parts:

1. Generation of potential thoughts for each state
2. Development of these thoughts for each state
3. Evaluation of the states

An important note is that, in comparison to the original ToT presented in the paper "Tree of Thoughts: Deliberate Problem Solving with Large Language Models" by S. Yao et al. (2022) [30], this version does not include any form of a search algorithm or backtracking. It is a simplified version of ToT.
Each level of the tree is a single prompt that develops the response from the previous level. The LLM is prompted by the first-level prompt, and after generating the response, the model is prompted by the second-level prompt, and so on, until it reaches the last level. At the last level, the model is prompted to rank each possible action in order of promise. The action with the highest ranking is chosen as the final response. In a few-shot setting, some examples can be added for each level.

Algorithm 1 Custom ToT

```
Require: $LLM(p)$, Set of prompts $P$, Set of available actions $A$; $a \in A, p \in P$
1: for each $prompt$ in $P$ do
2:     $response \leftarrow LLM(prompt)$
3: end for
4: return $\arg\max_a(response.ranking)$
```

The visualization of the flow can be seen in Fig. 6.3

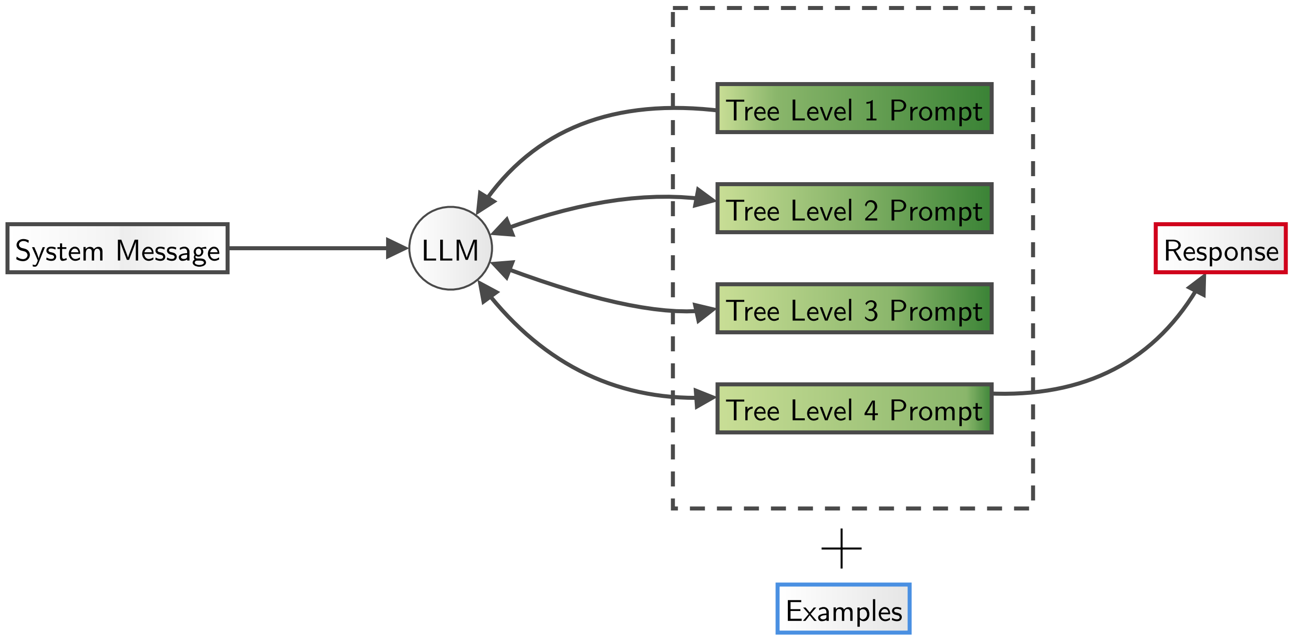

Figure 6.3: Flow, visualizing the interaction of components in ToT

6.3 Prompt with Actor-Critic Editing (PACE)

Prompt with Actor-Critic Editing, initially proposed in the paper ”PACE: Improving Prompt with Actor-Critic Editing for Large Language Models” by Y. Dong et al. (2023) [6], is a prompting strategy that utilizes two LLMs that interact with each other. One LLM is called the Actor, and the other is called the Critic. The Actor makes decisions, while the Critic evaluates these decisions and gives feedback to the Actor.

Algorithm 2 PACE

```
Require: Actor, Critic, Retries $\mu$, Best rating-action $\psi$, Observation $o$, Threshold $\epsilon$
 1: feedback $\leftarrow o$
 2: pace(feedback, retries)
 3: if retries $< \mu$ then
 4:     action $\leftarrow$ Actor(feedback)
 5:     critique $\leftarrow$ Critic(action)
 6:     if critique.rating $\geq \epsilon$ then
 7:         return action
 8:     else
 9:         if $\psi$.rating $<$ critique.rating then
10:             $\psi \leftarrow$ (critique.rating, action)
11:         end if
12:         pace(critique, retries - 1)
13:     end if
14: else
15:     return $\psi$.action
16: end if
```

The Actor always provides an explanation in addition to the action. The Critic evaluates both the explanation and the action. The Critic is given the same amount of information regarding the world as the Actor. The evaluation happens in two phases: 1. The Critic reads the explanation of the Actor and criticizes it, 2. The Critic gives the action a rating. The rating used in this thesis ranges from 1 (worst) to 5 (best). The flow in Fig. 6.4 describes the Actor-Critic interaction.

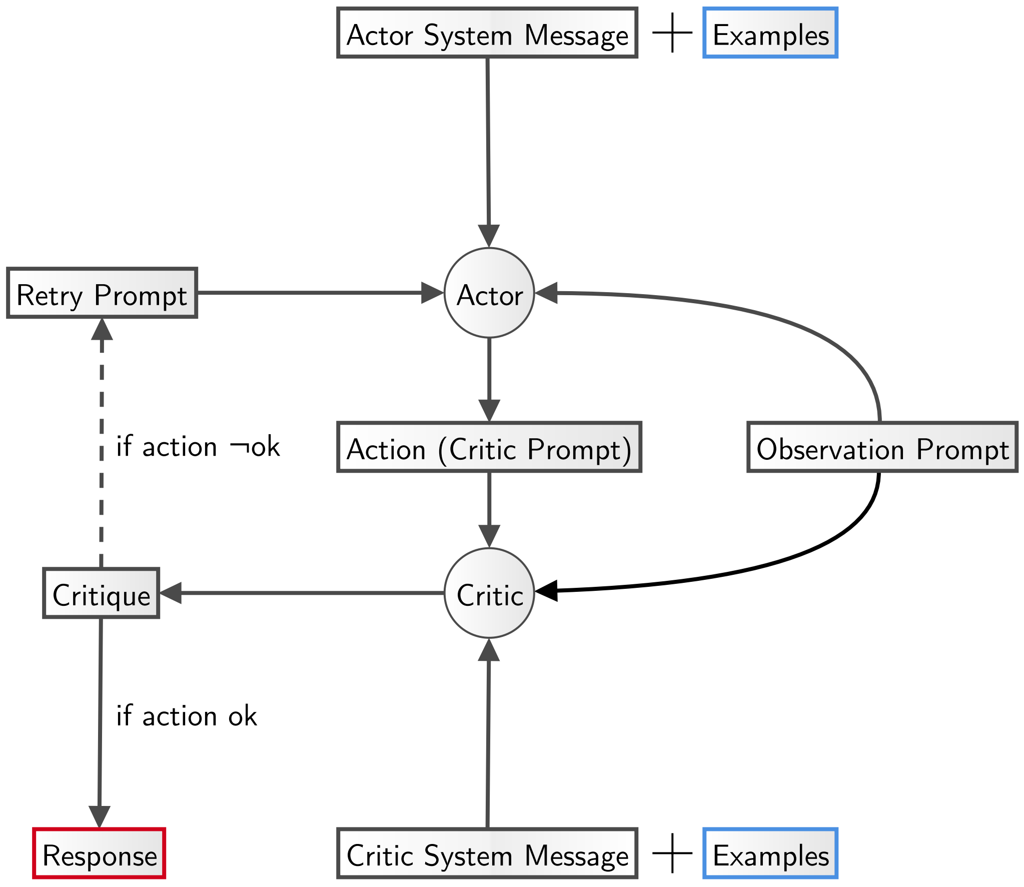

Figure 6.4: Process flow and components interaction in the PACE prompting

6.4 Prompt Templates

The corresponding prompt templates with the exact wording can be found at the end of the paper in Materials (chapter 11).

7 Experiments

This chapter dives deeper into the technical implementation and setup of the experiments. Custom libraries and various components will be discussed. Structure (section 7.2) describes SR-LLM project.

7.1 Overview

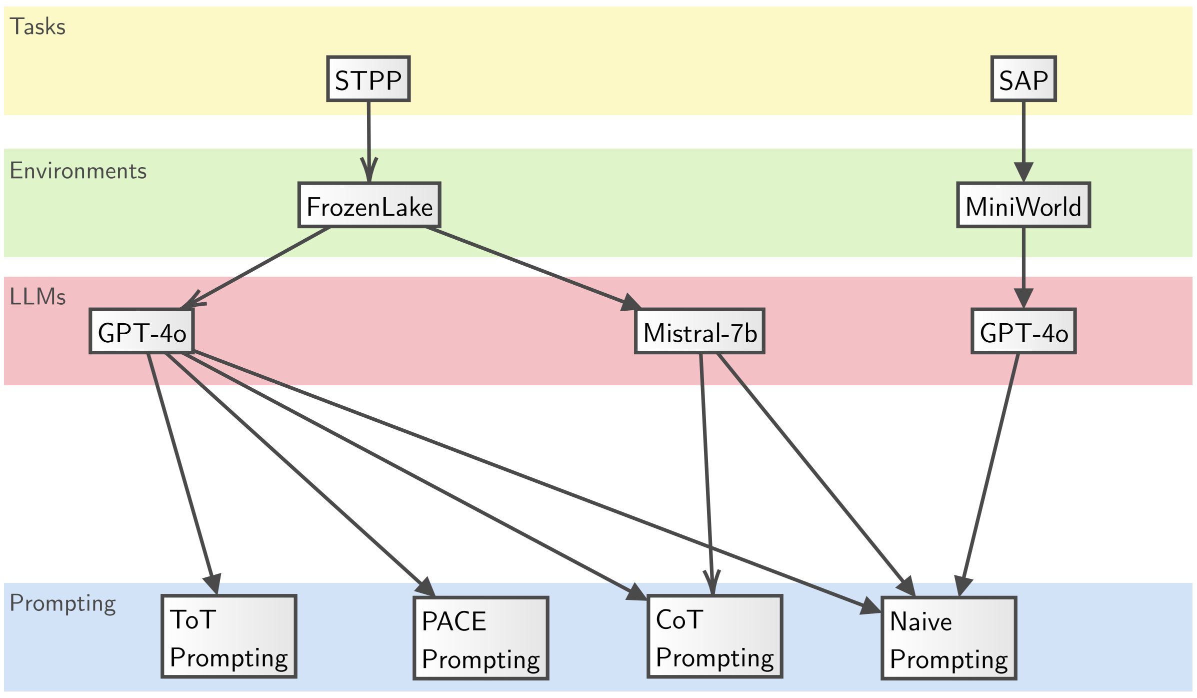

Figure 7.1: Overview of all the experiments conducted. Each path represents an experiment. Number of examples (shots) is not considered in the diagram.

7.2 Structure

The experiments are automated through scripts that utilize various components. The experiments are structured as follows:

7.2.1 Libraries and Dependencies

The implementation utilizes several libraries and modules, including:

- `oshandler`: Manages file operations.
- `frozenlake.prompt`: Contains various prompting strategies.
- `frozenlake.map`: Handles the FrozenLake map generation and manipulation.
- `frozenlake.intel`: Collects and processes experimental data.
- `frozenlake.logger`: Logs interactions and responses.
- `language_models.gpt4` and `language_models.mistral_7b_instruct`: Interfaces with GPT-4 and Mistral-7b-Instruct language models.
- `gymnasium`: Simulates the FrozenLake environment.
- Additional standard libraries: `os`, `math`, `numpy`, `argparse`.

7.2.2 Experiment Class

The `Experiment` class orchestrates the experiments, initializing the environment, prompting the LLM, logging interactions, and collecting data. Key attributes and methods include:

- **Initialization**: Loads the map, sets up the logger, defines variables, and prepares the system message.
- **Run Methods**:
  - `task_decomposition_run()`: Implements the CoT strategy as described in section 6.1.
  - `multi_plan_selection_run()`: Implements the ToT strategy as described in section 6.2.
  - `reflection_refinement_run()`: Implements the reflection and refinement strategy as described in section 6.3.
  - `mistral_naive_run()`: Runs naive strategy experiments with Mistral-7b-Instruct.

7.2.3 Execution Flow

Each experiment follows a structured flow:

1. **Initialization**: Set up the environment and LLM agents.
2. **Iterative Interaction**: Agents interact with the environment, receive feedback, generate responses, and take actions.
3. **Logging and Evaluation**: All interactions are logged, and the state of the agent and environment is continuously evaluated.
4. **Termination**: Upon reaching termination conditions, results are saved, and the environment is closed.

7.2.4 Running Experiments

The main script allows running experiments via command-line arguments, specifying the prompting strategy, number of examples (shots), and iterations. It supports the following commands:

- `task_decomposition`
- `reflection_refinement`
- `multi_plan_selection`
- `naive`
- `mistral_naive`
- `mistral_cot`

8 Results

The table 8.1 presents the performance metrics of various prompting strategies used with GPT-4o in the FrozenLake environment. # shows the number of maps the evaluation was conducted on. The table is organized as follows:

Naive Prompting: The success rate ranges from 58% (0 shots) to 64% (1 shot). The optimal rate is 58% for 0 shots. No survival runs were recorded. The average last distance is highest at 2.24 with 0 shots and reduces with examples. No escapes were recorded. 4x4 maps have a success rate of 46%, 8x8 maps 26% and 12x12 maps 0%

Chain-of-Thought (CoT): The success rate improves with the number of shots, peaking at 69% with 1 shot. The optimal rate is highest with 1 shot at 61%. Survival runs are only recorded with 1 shot at 0.03%. The average last distance decreases with more shots, indicating better performance. Escapes are recorded at 0.06 for 0 and 4 shots. For 12x12 maps in the 1-shot setting the success rate of just 6% is observed.

Tree-of-Thought (ToT): This strategy shows the highest success rate among all strategies, reaching 81% with 4 shots. The optimal rate is also highest at 69% with 4 shots. Survival runs peak at 30% with 0 shots, reducing with more examples. The average last distance significantly reduces with more shots. Escapes are recorded at 0.06 for 0 shots. Looking at 12x12 maps in the 4-s0.6hot setting, the success rate is 25%.

Prompt with Actor-Critic Editing (PACE): The success rate is relatively low, peaking at 55% with 1 shot. The optimal rate is highest at 49% with 0 shots. Survival runs reach 48% with 4 shots, indicating better performance in avoiding obstacles. The average last distance increases with more shots. Escapes are recorded at 0.03 with 0 shots. Again, looking at 12x12 maps in the 1-shot setting, the success rate is 5%.

	Shots	#	Success %	Optimal %	Survival %	Avg. Last Distance	Escapes
Naive	0	32	58	58	0	2.24	0
	0	32	64	64	0	1.77	**0.06**
CoT	1	32	69	61	0	1.09	0.03
	4	32	66	56	0	2.0	**0.06**
	0	32	70	60	30	0.70	0
ToT	1	32	71	59	24	0.76	0
	4	32	**81**	**69**	19	**0.44**	0
	0	32	54	49	5	2.41	0.03
PACE	1	32	55	45	0	1.86	0
	4	32	36	36	**48**	2.64	0

Table 8.1: STPP with GPT-4o in the FrozenLake environment

On Fig. 8.1 the success rates of the best combinations of prompting strategies are plotted against the map sizes.

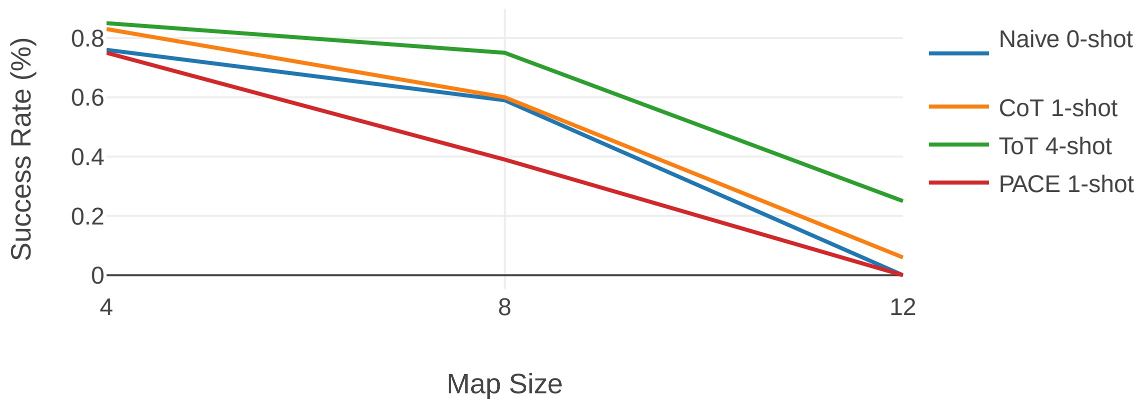

Figure 8.1: Success rate of the best prompting strategies combinations vs map size

The table 8.2 shows the performance metrics of a non-fine-tuned (also 5-bit quantized) Mistral-7b-Instruct model to the fine-tuned model. Non-fine-tuned model showeed very poor performance regarding the success rate (3%), but has a relatively high survival rate (47%). On the fine-tuned version two prompting strategies were test, Naive and CoT (Chain-of-Thought).

Naive Prompting (fine-tuned): The success rate for the Naive strategy is 18%. This is 500% (6 times) improvement compared to the non-fine-tuned model. The optimal rate is 13%, indicating that only a small fraction of runs achieved the optimal solution, but still surpassing the non-fine-tuned Mistral and achieving 333% better results. The survival rate is relatively high at 54%, suggesting that over half of the runs avoided collisions with obstacles. The average last distance to the goal is 3.67, and the number of escapes is 0.23, indicating some ability to escape traps. The success rate of 12x12 maps is 0%.

Chain-of-Thought (CoT) (fine-tuned): The CoT strategy shows a slightly higher success rate of 21% compared to Naive prompting. The optimal rate is also slightly higher at 17%. The survival rate drops to 21%, indicating fewer runs avoided collisions with obstacles compared to the Naive strategy. The average last distance to the goal is 4.05, higher than that of the Naive strategy. The number of escapes is lower at 0.12, indicating fewer instances of escaping traps. The success rate of 12x12 maps is 2%.

	Shots	#	Success %	Optimal %	Survival %	Avg. Last Distance	Escapes
Naive (no-ft)	0	32	3	3	47	5.13	0.19
Naive (ft)	0	32	18	13	**54**	3.67	**0.23**
CoT (ft)	0	32	**21**	17	21	4.05	0.12

Table 8.2: STPP with Mistral-7b-Instruct in the FrozenLake environment

Finally, the table 8.3 is about the performance metrics of the **Naive prompting strategy** using the GPT-4o-Vision model in the PutNext environment. This table evaluates the model's performance without providing any example shots (0 shots).
In terms of success, the model achieved a 35% success rate, while the optimal rate is slightly lower at 30%. The partial success rate, is 40%.

Only 12 blocks large environments were tested with 32 random initial positions.

	Shots	#	Success %	Optimal %	Survival %	Partial Success %
Naive	0	32	35	30	100	40

Table 8.3: SAP with GPT-4o-Vision in the PutNext environment

9 Discussion

Regarding the prompting strategies, the overall expectation was that PACE would show the best success rate, but this was not the case. The GPT-4o model, when not equipped with memory, often performed repetitive moves or went in circles, as it is described on the Fig. 9.1. In this situation the LLM would have first moved to the right, then cycled left and right. Obviously, such behavior had a negative impacting on both the success rate and the optimal rate. These repetitive moves are difficult to combat and require a much larger step window or some form of context attached to each feedback/observation prompt of the model.

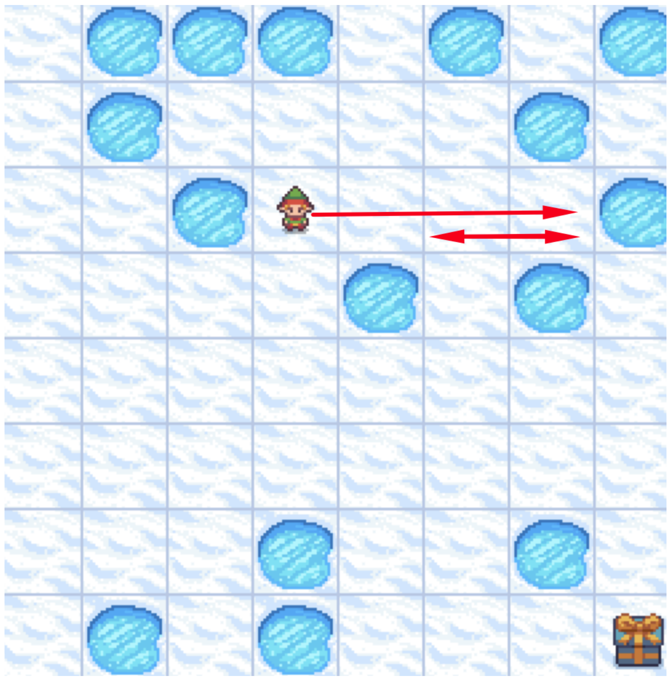
Figure 9.1: Caption

During testing, it was observed that GPT-4o could not plan many steps ahead correctly (although it was not explicitly prompted to do so in any prompting strategy except the ToT) and similar issues were noted with Mistral-7b-Instruct. In the paper "Can large language models be good path planners? A benchmark and investigation on spatial-temporal reasoning," the authors achieved much higher performance with ReAct Prompting (96% success rate), while task fine-tuned T5-base achieved a success rate of 98%. However, the paper focused on long-term path planning, which differs from STPP. On Fig. 8.1 one can observe that with the increasing map size, the overall success rate decreases, which indicates that GPT-4o struggles with bigger and more complex environments. Moreover, this is a characteristic trend with every kind of prompting and fine-tuning, the overall trend is that when given a larger map, the LLMs struggle and cannot perform as well as on smaller maps.

Regarding the fine-tuning, there could be observed a significant improvement in performance on the fine-tuned model vs non-fine-tuned model. This is a good indicator and a step in the right direction. Nevertheless, the high survival rate of the fine-tuned Mistral-7b-Instruct model indicates that the model probably learned how to avoid obstacles but not so much how to choose the best next action. Overall, the fine-tuning could justify the cost and time investment if the dataset size increased from just 410 records to let's say couple of thousands of records.

Returning to the prompting strategies, Tree-of-Thought (ToT) with 4 shots showed the best overall performance (81% success rate) in terms of success and optimal rates, which was a pleasant surprise. Increase in number of examples does not necessarily constitute in an improved performance, as it was observed in CoT and PACE.

The performance of GPT-4o-Vision on the SAP task was comparable to those discovered in research.

10 Conclusions

This thesis explored the spatial reasoning capabilities of current state-of-the-art language models (LLMs) through a series of controlled experiments. The primary focus was on evaluating various prompting strategies vs explicit fine-tuning to enhance the performance of LLMs in spatial tasks.

The experiments demonstrated that different prompting strategies significantly impact the performance of LLMs. Contrary to initial expectations, the PACE strategy did not yield the highest success rate. Instead, the Tree-of-Thought (ToT) strategy with 4 shots showed the best overall performance in terms of success and optimal rates. This finding was a pleasant surprise, indicating that more complex, structured prompting can significantly enhance the spatial reasoning abilities of LLMs.

However, the experiments also revealed limitations. The GPT-4o model, when not equipped with memory, often performed repetitive moves or went in circles, negatively affecting both success and optimal rates. This issue underscores the importance of incorporating memory or context into the feedback/observation prompts to improve performance. Both GPT-4o and Mistral-7b-Instruct struggled on bigger maps and planning multiple steps ahead accurately, indicating a broader challenge in current LLMs' ability to manage long-range dependencies in spatial tasks.

Fine-tuning the Mistral-7b-Instruct model showed mixed results. While the model did improve through dine-tuning of the FrozenLake data, the ability to choose the best next action is far from perfect. The high survival rate achieved indicates an understanding of obstacle avoidance but highlights the need for more effective fine-tuning strategies to enhance overall task performance. The fine-tuning process may have justified the fine-tuning more, if the dataset was larger.

The addition of a vision component to the GPT-4o model (GPT-4o-Vision) improved performance in the Spatial Actions Planning (SAP) tasks, aligning with findings in recent research. This enhancement was particularly beneficial in scenarios where textual descriptions alone were insufficient.

In conclusion, while LLMs demonstrated promising results in handling complex tasks requiring spatial understanding and reasoning, there is still significant room for improvement. The challenges observed in planning accuracy and the mixed results from fine-tuning highlight the need for further research and development. Future work should focus on integrating memory and context mechanisms, exploring more sophisticated fine-tuning approaches, and expanding the scope of multi-modal capabilities to advance the spatial reasoning abilities of LLMs.

Overall, this thesis contributes valuable insights into the strengths and limitations of current LLMs in spatial reasoning tasks, providing a foundation for future advancements in this area.

11 Materials

- GitHub Project
- FineTuning Notebook (Colab)
- Merging Adapter Notebook (Kaggle)
- GGUF Notebook (Kaggle)

11.1 Naive Prompt Template (GPT-4o)

System Message

Task
You are a navigational agent in a 2D {SIZE}x{SIZE} grid world filled with specific obstacles and limited by boundaries.
Your objective is to navigate to {GOAL}.
If given, always use the output format template.
Always generate response shorter than MAX_TOKENS = {MAX_TOKENS}.

Environment
Each cell in the grid is defined through coordinates (r, c).
Your starting position is at {ORIGIN}.
The grid has obstacles located at {OBSTACLES}.
Collision with boundaries happens when:
r < {LOWER_BOUND} OR c < {LOWER_BOUND} OR r > {UPPER_BOUND} OR c > {UPPER_BOUND}.
If you collide with obstacles the game ends immediately.

Move commands
You can move in four directions:
"up" decreases r by one
"down" increases r by one
"left" decreases c by one
"right" increases c by one

Objective
Given current position (r, c), navigate to {GOAL} using move commands.

Avoid collisions with obstacles and boundaries of the grid.

11.2 CoT Prompt Template (GPT-4o)

System Message

Task
You are a navigational agent in a 2D {SIZE}x{SIZE} grid world filled with specific obstacles and limited by boundaries.
Your objective is to navigate to {GOAL}.
If given, always use the output format template.
Always generate response shorter than MAX_TOKENS = {MAX_TOKENS}.

Environment
Each cell in the grid is defined through coordinates (r, c).
Your starting position is at {ORIGIN}.
The grid has obstacles located at {OBSTACLES}.
Collision with boundaries happens when:
r < {LOWER_BOUND} OR c < {LOWER_BOUND} OR r > {UPPER_BOUND} OR c > {UPPER_BOUND}.
If you collide with obstacles the game ends immediately.

Move commands
You can move in four directions:
"up" decreases r by one
"down" increases r by one
"left" decreases c by one
"right" increases c by one

Objective
Given current position (r, c), navigate to {GOAL} using move commands.
Avoid collisions with obstacles and boundaries of the grid.

Decision Rationale
Think step by step as follows:

1. Determine direction of the goal: From your current location (r, c) determine direction of the goal {GOAL}.
2. Choose a safe command: Based on the direction determined in step 1, think about the needed increase/decrease of r/c and choose the next move command. Ensure that this move does not lead you into an obstacle or into a boundary.
3. Execute the move: Give a single command to move in the chosen direction.

Continue this process until the goal is reached.

11.3 ToT Prompt Templates (GPT-4o)

System Message

Same as Naive prompt

Tree Level 1 Prompt

Prompt
{FEEDBACK}. For each possible move: "up", "down", "left", and "right" brainstorm and analyze its outcome in one (1) sentence.

Tree Level 2 Prompt

Prompt
For each direction, consider in one (1) sentence:
1. What will be the new position after making this move?
2. Is this position in the list of obstacles?
3. Is it in the boundaries of the grid?
4. What is the distance to the goal?

Assign a probability of success of reaching the goal to each direction based on these factors

Tree Level 3 Prompt

```
## Prompt ##
Prompt For each direction, plan your path further in one (1) sentence.
Are those scenarios generally safe or not?
```

Tree Level 4 Prompt

```
## Prompt ##
Based on the evaluations and scenarios, rank the moves in order of promise from most promising to the
least promising. Give one (1) sentence explanation per rank.
```

11.4 PACE Prompt Template (GPT-4o)

Actor System Message

Same as Naive Prompt

Critic System Message

```
## Task ##
You are a critic of a navigational agent.
Navigational agent is in a 2D {SIZE}x{SIZE} grid world filled with specific obstacles and limited by
boundaries.
Objective of navigational agent is to navigate to {GOAL}.
Your objective is to criticize and evaluate actions of a navigational agent.
If given, always use the output format template.
Always generate response shorter than MAX_TOKENS = {MAX_TOKENS}.

## Environment ##
Each cell in the grid is defined through coordinates (r, c).
Your starting position is at {ORIGIN}.
The grid has obstacles located at {OBSTACLES}.
Collision with boundaries happens when:
r < {LOWER_BOUND} OR c < {LOWER_BOUND} OR r > {UPPER_BOUND} OR c >
{UPPER_BOUND}.
If you collide with obstacles the game ends immediately.

## Move commands ##
You can move in four directions:
"up" decreases r by one
"down" increases r by one
"left" decreases c by one
"right" increases c by one

## Objective of navigational agent ##
Given current position (r, c), navigational agent should navigate to {GOAL} using move commands. Nav-
igational agent should avoid collisions with obstacles and boundaries of the grid.
```

Critic Prompt

```
## Prompt ##
Navigational agent decided to move {DIRECTION} with following explanation: {EXPLANATION}
Please criticize and evaluate decision of navigational agent as follows:
  1. Tell whether this move is good or bad and shortly explain why. Consider following in your evaluation:
       • Does this move result in a collision with obstacles or grid boundaries?
       • Does this move put navigational agent in a dangerous or unsafe situation?
       • Does this move have a potential to be beneficial in the long term?
  2. Rate the decision of navigational agent from 0 to 5 (0 being worst and 5 being best).
```

11.5 Naive Prompt Template (GPT-4o-Vision)

System Message

```
## Task ##
You are an agent in a 3D {SIZE}x{SIZE} world filled with specific objects and limited by boundaries.
Your objective is to find a red box and place it next to a yellow box.
If given, always use the output format template.
Always generate response shorter than MAX_TOKENS = {MAX_TOKENS}.

## Environment ##
You will be given an image, that represents your current view in some specific position in the 3D world.

Available actions
At each step you can perform one action. Here is a list of all possible actions:
"turn left"
"turn right"
"move forward"
"move back"
"pick up"
"drop"
"toggle / activate an object"
"complete task"

## Objective ##
The red box must be placed next to a yellow box. The goal is to perform this task in as few steps as possible.
```

References

[1] `https://openai.com/index/hello-gpt-4o/`. [Accessed 03-06-2024].

[2] Mohamed Aghzal, Erion Plaku, and Ziyu Yao. Can large language models be good path planners? a benchmark and investigation on spatial-temporal reasoning. *arXiv preprint arXiv:2310.03249*, 2023.

[3] Boyuan Chen, Zhuo Xu, Sean Kirmani, Brian Ichter, Danny Driess, Pete Florence, Dorsa Sadigh, Leonidas Guibas, and Fei Xia. Spatialvlm: Endowing vision-language models with spatial reasoning capabilities. *arXiv preprint arXiv:2401.12168*, 2024.

[4] Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. Textworld: A learning environment for text-based games. In *Workshop on Computer Games*, pages 41–75. Springer, 2018.

[5] Gautier Dagan, Frank Keller, and Alex Lascarides. Dynamic planning with a llm. *arXiv preprint arXiv:2308.06391*, 2023.

[6] Yihong Dong, Kangcheng Luo, Xue Jiang, Zhi Jin, and Ge Li. Pace: Improving prompt with actor-critic editing for large language model. *arXiv preprint arXiv:2308.10088*, 2023.

[7] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.

[8] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[9] Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*, 2024.

[10] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[11] Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213. Curran Associates, Inc., 2022.

[12] Fangjun Li, David C Hogg, and Anthony G Cohn. Advancing spatial reasoning in large language models: An in-depth evaluation and enhancement using the stepgame benchmark. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18500–18507, 2024.

[13] Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. *Advances in Neural Information Processing Systems*, 36, 2024.

[14] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

[15] Jieyi Long. Large language model guided tree-of-thought. *arXiv preprint arXiv:2305.08291*, 2023.

[16] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

[17] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22, 2023.

[18] Xavi Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Ruslan Partsey, Jimmy Yang, Ruta Desai, Alexander William Clegg, Michal Hlavac, Tiffany Min, Theo Gervet, Vladimir Vondrus, Vincent-Pierre Berges, John Turner, Oleksandr Maksymets, Zsolt Kira, Mrinal Kalakrishnan, Jitendra Malik, Devendra Singh Chaplot, Unnat Jain, Dhruv Batra, Akshara Rai, and Roozbeh Mottaghi. Habitat 3.0: A co-habitat for humans, avatars and robots, 2023.

[19] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[20] Manasi Sharma. Exploring and improving the spatial reasoning abilities of large language models. *arXiv preprint arXiv:2312.01054*, 2023.

[21] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36, 2024.

[22] Zhengxiang Shi, Qiang Zhang, and Aldo Lipani. Stepgame: A new benchmark for robust multi-hop spatial reasoning in texts. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 11321–11329, 2022.

[23] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[24] Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to rearrange their habitat. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[25] J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:15032–15043, 2021.

[26] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel,

Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023.

[27] Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader?, 2022.

[28] 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.

[29] Zhun Yang, Adam Ishay, and Joohyung Lee. Neurasp: Embracing neural networks into answer set programming. *arXiv preprint arXiv:2307.07700*, 2023.

[30] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[31] 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.

[32] Yuexiang Zhai, Shengbang Tong, Xiao Li, Mu Cai, Qing Qu, Yong Jae Lee, and Yi Ma. Investigating the catastrophic forgetting in multimodal large language model fine-tuning. In Yuejie Chi, Gintare Karolina Dziugaite, Qing Qu, Atlas Wang Wang, and Zhihui Zhu, editors, *Conference on Parsimony and Learning*, volume 234 of *Proceedings of Machine Learning Research*, pages 202–227. PMLR, 03–06 Jan 2024.

[33] Jiazhao Zhang, Kunyu Wang, Rongtao Xu, Gengze Zhou, Yicong Hong, Xiaomeng Fang, Qi Wu, Zhizheng Zhang, and Wang He. Navid: Video-based vlm plans the next step for vision-and-language navigation. *arXiv preprint arXiv:2402.15852*, 2024.

[34] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19724–19731, 2024.