LLM-A*: Large Language Model Enhanced Incremental Heuristic Search on Path Planning

Anonymous ACL submission

Abstract

Path planning is a fundamental scientific problem in robotics and autonomous navigation, requiring the derivation of efficient routes from starting to destination points while avoiding obstacles. Traditional algorithms like A* and its variants are capable of ensuring path validity but suffer from significant computational and memory inefficiencies as the state space grows. Conversely, large language models (LLMs) excel in broader environmental analysis through contextual understanding, providing global insights into environments. However, they fall short in detailed spatial and temporal reasoning, often leading to invalid or inefficient routes. In this work, we propose **LLM-A***, an new LLM based route planning method that synergistically combines the precise pathfinding capabilities of A* with the global reasoning capability of LLMs. This hybrid approach aims to enhance pathfinding efficiency in terms of time and space complexity while maintaining the integrity of path validity, especially in large-scale scenarios. By integrating the strengths of both methodologies, **LLM-A*** addresses the computational and memory limitations of conventional algorithms without compromising on the validity required for effective pathfinding.

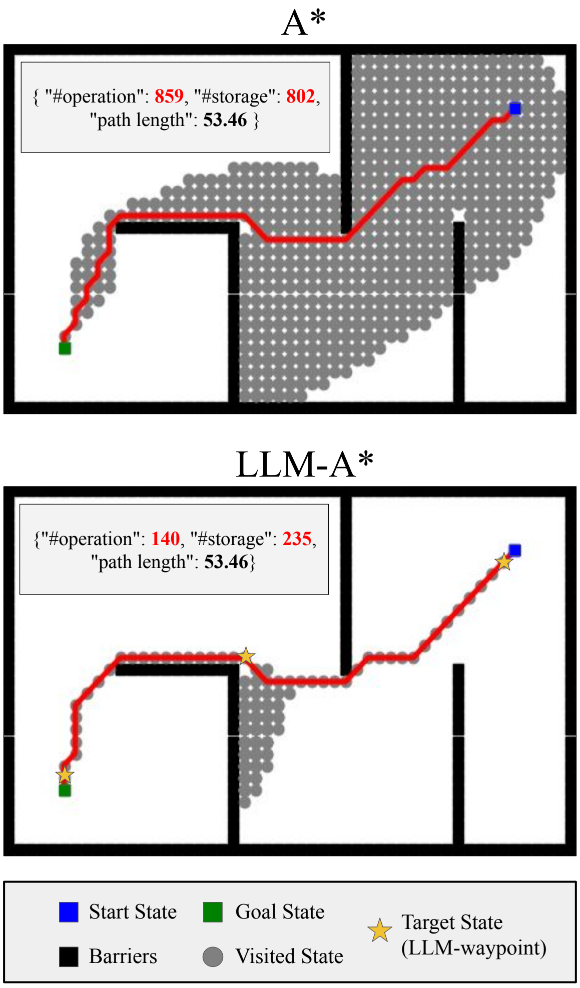


Figure 1: An comparison between **LLM-A*** and A* in computation and memory efficiency during pathfinding process. **LLM-A*** leverages target states generated by LLMs as waypoints to guide the searching process, significantly reducing the number of visited states, which leads to fewer operations and storage usage than A*.

1 Introduction

Path planning is the computational process of determining a path from an initial point to a destination point that adheres to specific criteria, such as avoiding obstacles, minimizing travel distance or time, and satisfying other constraints (LaValle, 2006; Hart et al., 1968b; Karaman and Frazzoli, 2011). This problem is crucial across several fields, such as robotics, autonomous vehicle navigation, industrial automation, and virtual environment navigation due to its direct impact on the efficiency, safety, and feasibility of operational systems (Thrun et al., 2005; Karaman and Frazzoli, 2011; Fiorini and Shiller, 1998; Fox et al., 1997).

Existing path planning algorithms are capable of effectively completing planning tasks and ensuring the validity of their paths. However, as the environment and map scale up, algorithms like A* and its variants (Hart et al., 1968b; Korf et al., 2001; Harabor and Grastien, 2011; Jansen and Buro, 2007) encounter an exponential increase in computational and memory demands. This occurs because the pathfinding process can become sub-optimal (see Figure 1), where the algorithm might spend unnecessary effort exploring less relevant areas, leading to exponential increases in time complexity as the map size enlarges.

Meanwhile, Large Language Models (LLMs) have achieved notable milestones in various planning tasks (Naveed et al., 2023; Yin et al., 2023; Chen et al., 2023a; Shinn et al., 2024; Yang et al., 2023). These models demonstrate capabilities in processing and reasoning over long-context input to provide valuable global insights that reflect their understanding of the environment, such as identifying the relative positions of barriers, agents, and goals. However, they struggle with complex, long-term planning and complex spatial reasoning tasks such as grid-based path planning. LLMs often generate paths that are either invalid or ungrounded, resulting in incomplete or colliding paths, indicating a gap in their capability to handle detailed spatial intricacies (Aghzal et al., 2023).

In this work, we propose **LLM-A***, a new LLM based route planning method that synergizes the traditional A* algorithm with the global insights from Large Language Models. As illustrated in Fig. 1, this hybrid approach leverages LLM-generated waypoints to guide the path searching process, significantly reducing computational and memory costs. In addition, by integrating the standard L2 distance-based heuristic of A* with new heuristic values derived from these waypoints, **LLM-A*** addresses the granularity issues in LLM-generated solutions, ensuring the validity of the output paths.

We conducted extensive experiments across various environment to compare the performance of A* and **LLM-A*** (integrating LLAMA3 with few-shot prompting). As illustrated in Figure 3, A* exhibits exponential growth in both computational operations and storage requirements with linearly increasing environment scale. In contrast, **LLM-A*** shows a nearly linear growth pattern, indicating superior scalability. This suggests that **LLM-A*** is significantly more efficient in terms of both computation and memory, making it better suited for larger environments. Furthermore, our primary experimental results, summarized in Table 1, reveal that **LLM-A*** not only excels in scalability but also outperforms A* in baseline computational and memory efficiency. **LLM-A*** achieves significantly lower operation and storage ratios compared to A*, requiring less than about half the operations and storage needed by A* on average for the pathfinding process, thereby offering a robust and efficient solution for large-scale path planning.

2 Related Work

Traditional Algorithms in Path Planning. Pathfinding has been pivotal in artificial intelligence, robotics, and computer graphics, with numerous algorithms developed to address various challenges. Among the foundational methods, the A* algorithm, introduced by Hart, Nilsson, and Raphael in 1968, stands out for its use of a heuristic to estimate the cost from the current to the goal node, balancing greedy best-first search with uniform-cost search for efficient pathfinding (Hart et al., 1968a). Similarly, Pearl's Best First Search (BFS), proposed in 1984, prioritizes nodes based on heuristic values but can lead to longer paths due to its focus on the most promising nodes (Pearl, 1984).

Extensions of A* have aimed to enhance its efficiency and adaptability. Korf's Iterative Deepening A* (IDA*), from 1985, combines depth-first search with A*'s heuristic to create a memory-efficient approach (Korf, 1985). Korf also introduced Learning Real-time A* (LRTA*) in 1990, incorporating real-time learning to dynamically update heuristic values, improving performance in changing environments (Korf, 1990). Russell's Simplified Memory Bounded A* (SMA*), from 1992, addresses memory constraints by selectively forgetting less promising paths, making it suitable for resource-limited applications (Russell, 1992).

Further enhancements include Stentz's Dynamic A* (D*) from 1994, which recalculates paths as the environment changes, proving effective for navigation in unknown or evolving terrains (Stentz, 1994). Koenig et al.'s Lifelong Planning A* (LPA*), introduced in 2004, incrementally updates paths in dynamic and large-scale environments (Koenig et al., 2004). Harabor and Grastien's Jump Point Search (JPS), proposed in 2011, optimizes A* for only grid-based maps by identifying key "jump points", reducing the number of expanded nodes (Harabor and Grastien, 2011). Nash et al.'s Theta*, from 2007, allows line-of-sight checks between nodes, resulting in more direct paths (Nash et al., 2007).

Hierarchical approaches, such as Holte et al.'s Hierarchical A* (HA*) from 1996, decompose large pathfinding problems into smaller subproblems through a hierarchy of abstractions, reducing computational complexity (Holte et al., 1996). Botea et al.'s Hierarchical Path-finding A* (HPA*), introduced in 2004, improves transitions between abstraction levels for efficient large-map pathfinding (Botea et al., 2004).

Specialized methods also contribute significantly. Demyen and Buro's Triangulation-Based Pathfinding (TRA*), proposed in 2006, navigates polygonal environments using triangulation, suited for non-grid-based settings (Demyen and Buro, 2006). Koch's Grid-specific Hierarchical Path-finding (GHPA*), introduced in 2011, optimizes grid maps pathfinding by integrating hierarchical and grid-specific optimizations (Koch, 2011).

Large Language Models in Path Planning. Large Language Models (LLMs) have recently achieved remarkable success in natural language processing tasks and other domains (Naveed et al., 2023). Studies such as (Shridhar et al., 2020b; Song et al., 2023; Shah et al., 2023) explore LLMs in high-level planning, highlighting challenges in long-term planning and spatial reasoning (Aghzal et al., 2023). Our research shifts focus to continuous environments, offering a more realistic setting compared to grid-based maps. Continuous spaces align better with real-world conditions, providing a more natural interface for human interaction and allowing higher precision in spatial reasoning.

LLMs show varying proficiency in spatial reasoning (Ilharco et al., 2020; Patel and Pavlick, 2021; Bubeck et al., 2023; Abdou et al., 2021), yet face limitations in spatial reasoning and planning (Agrawal, 2023; Xie et al., 2023; Wu et al., 2023). We introduce a benchmark for path planning in continuous environments, integrating spatial and temporal reasoning. Prior benchmarks (Côté et al., 2019; Shridhar et al., 2020a; Ruis et al., 2020; Wu et al., 2021) often neglect temporal planning aspects. Our study further evaluates LLMs in robot motion and path planning contexts, addressing limitations in end-to-end planning (Liu et al., 2023; Chen et al., 2023b; Xie et al., 2023; Silver et al., 2022).

Understanding the interplay between high-level and low-level planning is crucial (Latif, 2024; Ding et al., 2024; Zhou et al., 2024). High-level planning involves strategic goals, while low-level focuses on detailed task execution. Our research explores LLMs' adaptability in correcting low-level planning errors, ensuring resilience in dynamic conditions.

3 Methodology

3.1 A* Algorithm

The A* algorithm is a widely used pathfinding and graph traversal algorithm. It seeks to find the shortest path from a start node s_0 to a goal node s_g by combining the strengths of Dijkstra's Algorithm and Greedy Best-First Search.

A* employs a heuristic function $h(s)$ to estimate the cost from a node s to the goal, and a cost function $g(s)$ to track the exact cost from the start to s. The total cost function $f(s)$, defined as $f(s) = g(s) + h(s)$, guides the search towards the goal. The algorithm operates as follows:

1. **Initialization:** Place the start node s_0 in the OPEN list with $f(s_0) = g(s_0) + h(s_0)$, and initialize the CLOSED list as empty.

2. **Search:** Continuously select the node s from the OPEN list with the lowest f-cost, expand its neighbors, and update their costs. If a neighbor s_n offers a cheaper path than previously recorded, update its cost and parent node. Repeat until the goal node s_g is reached or the OPEN list is empty.

3. **Path Reconstruction:** Once s_g is reached, reconstruct the path by tracing back from s_g to s_0 via parent nodes.

The heuristic $h(s)$ should be admissible, meaning it does not overestimate the true cost to reach the goal. This ensures the path optimality of A*.

3.2 LLM-A* Algorithm

LLM-A* integrates the global insights provided by Large Language Models (LLMs) with the A* algorithm's optimal local search mechanism, where achieves a balance between the efficiency of the pathfinding process and optimality. The pseudocode for **LLM-A*** is shown in Figure 2, and it closely resembles the original A* algorithm.

LLM-A* accepts the same inputs as A*, with the addition of an obstacle state variable, denoted

Algorithm 1 LLM-A* Algorithm for Path Planning

1: **Input:** START state s_0, GOAL state s_g, OBSTACLE state obs, heuristic function h, cost function g, Large Language Model llm
2: **Output:** Path P from s_0 to s_g
3: Initialize the OPEN list $O = \{s_0\}$, CLOSE list $C = \{\}$, TARGET list $T = llm(s_0, s_g, obs)$, TARGET state $t = T.start$, $g(s_0) = 0$, $f(s_0) = h(s_0)$, $P = \{\}$
4: **while** $O \neq \emptyset$ **do**
5:     $s_a \leftarrow$ state in O with the lowest f-cost
6:     **if** $s_a = s_g$ **then**
7:         **return** reconstruct_path(s_a)
8:     Remove s_a from O
9:     Add s_a to C
10:     **for all** neighbors s_n of s_a **do**
11:         **if** $s_n \in C$ **then**
12:             **continue**
13:         **if** $s_n = t$ **and** $s_g \neq t$ **then**
14:             $t = T.next$
15:             update f-cost of s in O
16:         Tentative cost $g_{tent} = g(s_a) + cost(s_a, s_n)$
17:         **if** $s_n \notin O$ or $g_{tent} < g(s_n)$ **then**
18:             Update path to s_n to go through s_a
19:             $g(s_n) = g_{tent}$
20:             $f(s_n) = g(s_n) + h(s_n) + cost(t, s_n)$
21:             **if** $s_n \notin O$ **then**
22:                 Add s_n to O
23: **return failure**

Figure 2: LLM-A* Algorithm Pseudocode

as obs. This obstacle state is utilized to compute a TARGET list T, which comprises a sequence of path nodes from the start state s_0 to the goal state s_g. This list is generated through a prompt to a large language model, reflecting the model's understanding and global perspective of the current environment. The returned T must meet two critical constraints in the following:

1. **Containment of Start and Goal Points:** T must include the start point and goal point that match the inputs s_0 and s_g. If the returned T does not satisfy this requirement, s_0 and s_g must be inserted by algorithm.

2. **Obstacle Avoidance:** Every target node t in T must not be located within any obstacle obs. If any node t is found within an obstacle, it is removed from T by algorithm.

The pathfinding process of **LLM-A*** is similar to that of A*. It uses a heuristic function h, a cost function g, an OPEN list O, and a CLOSED list C. The algorithm searches through each state in O until the goal state s_g is reached. Each explored state s_a is saved into C to avoid redundant searches. The distinction that encapsulates the main differences between **LLM-A*** and A* happens during the expansion of the neighbor state s_n (see in Figure 2:13-15). For each s_n, we check if it matches the current target t from T. If the current t is reached, t is updated to the next target in T. Subsequently, the f-cost of every state in the current O is re-computed, where the f-cost in **LLM-A*** is computed as the sum of the state's cost, the heuristic value, and the cost from the state to current t (see in Figure 2:20), defined as $f(s) = g(s) + h(s) + cost(t, s)$. This step introduces an additional computational amount to the pathfinding process, and the time complexity scales linearly with both the length of T and the increasing size of O. However, it is important that this re-computation process ensures that the f-cost of visited states in O remains accurate and updated with the new target t.

General Applicability. **LLM-A*** retains the versatility of the original A*, making it suitable for a wide range of pathfinding tasks across various environments, where specialized A* variants such as JPS and GHPA* (Harabor and Grastien, 2011; Koch, 2011), which are tailored to grid maps and specific scenarios, and the mechanism of **LLM-A*** is able to handle diverse and large-scale environments effectively. This generality positions **LLM-A*** as a robust alternative to A*.

3.3 Prompt Techniques

Few shot Learning. In the methodology we termed "Few Shot Learning" or "Vanilla Prompting," our initial approach involves directly presenting the Large Language Model (LLM) with ground-truth sequences of actions as prompts. This method is informed by previous studies which have demonstrated that the performance of such models can vary significantly based on the volume of task-specific examples provided (Cao et al., 2019; Razeghi et al., 2022). To investigate this further, we employed a few-shot learning technique, wherein we provides five demonstrations (See Table 2 in Appendix) presented to the LLM. This approach aimed to determine the optimal number of examples that would enhance the model's accuracy and learning efficiency.

Chain of Thought. The Chain-of-Thought (CoT) methodology, as proposed by (Wei et al., 2022), introduces a technique that encourages a Large Language Model (LLM) to engage in a sequential, step-by-step reasoning process. This approach has demonstrated substantial efficacy in tasks necessitating multiple layers of reasoning and decision-making. In light of its proven effectiveness, we have adapted the CoT strategy (See Table 3 in Appendix) to the specific requirements of path planning.

Recursive Path Evaluation. The Recursive Path Evaluation (RePE) methodology (See Table 4 in Appendix) is designed to guide Large Language Models (LLMs) in generating paths incrementally, with a particular emphasis on evaluating each step in the process. This approach gains its effectiveness from deconstructing the path planning problem into three distinct sub-problems: environment analysis, path generation, and path evaluation. By following these sub-problems in a recursive manner, the model systematically navigates towards the goal, ensuring compliance with predefined constraints at each stage. This concept bears a resemblance to the ReAct approach, Step Back QA, and Self Reflection (Yao et al., 2022; Zheng et al., 2023; Renze and Guven, 2024) in its processing step by step foundational principles. Meanwhile, RePE receives no feedback or observation from environment, and it distinctively focuses on a step-by-step progression and only intrinsic reasoning, where the path is constructed one point at a time with environment analysis and path evaluation. This methodical approach not only facilitates more precise navigation by the LLM but also allows for continuous assessment and adjustment at each juncture, thereby may enhancing the overall accuracy of the path planning process.

4 Experiments

4.1 Dataset

Our dataset consists of 100 manually selected 50×30 maps from a randomly generated collection, each with 10 different start and goal positions. Therefore, there are 1000 samples in total (see Figure 1 for sample visualization). Our data conform to the standard of search-based algorithm environments in a continuous space. Each map includes the following parameters:

- x_range: The minimum and maximum x-coordinates of the environment boundary range as $[x\_min, x\_max]$.
- y_range: The minimum and maximum y-coordinates of the environment boundary range as $[y\_min, y\_max]$.
- $horizontal\_barriers$: List of horizontal barriers, each represented as $[y, x\_start, x\_end]$.
- $vertical\_barriers$: List of vertical barriers, each represented as $[x, y\_start, y\_end]$.
- $start\_goal$: List of 10 unique start and goal positions for each map.

These parameters define the structure and constraints of each map, ensuring consistency and relevance to the standard experimental environment conditions for search-based algorithms. Meanwhile, the map environment is able to scale properly for scalability experiment.

Methodology	Base Model	Prompt Approach	Operation Ratio ↓ (%)	Storage Ratio ↓ (%)	Relative Path Length ↓ (%)	Valid Path Ratio ↑ (%)
A*	-	-	100	100	100	100
LLM	GPT-3.5	Few-Shot	-	-	119.38	12.80
		CoT	-	-	151.73	15.20
		RePE	-	-	183.87	7.80
	LLAMA3	Few-Shot	-	-	111.05	12.60
		CoT	-	-	114.89	12.00
		RePE	-	-	138.32	16.40
LLM-A* (Ours)	GPT-3.5	Few-Shot	57.39	74.96	102.44	**100**
		CoT	69.50	83.65	102.54	**100**
		RePE	85.47	96.53	**102.41**	**100**
	LLAMA3	Few-Shot	**44.59**	**64.02**	102.47	**100**
		CoT	47.60	66.27	102.46	**100**
		RePE	64.08	80.19	102.54	**100**

Table 1: Quantitative analysis of three pathfinding methodologies: the classical A* algorithm, an LLM-only approach, and our proposed **LLM-A*** approach. The methodologies are evaluated on the map size (50×30) of original samples. The LLM-only approaches explore the path without explicitly searching the space grid by grid, so we do not report the operation and storage ratio. The table includes the results from **GPT-3.5** and **LLAMA3** models with three prompting approaches: Few-Shot, Chain of Thought (CoT), and Recursive Path Evaluation (RePE) for both LLM-only and **LLM-A*** approaches (see Section 4.4 for details).

4.2 Experimental Setup

Large Language Model. We employ **GPT-3.5-TURBO** and **LLAMA3-8B-16bit** for their balance of robustness and cost-effectiveness in validating the **LLM-A*** algorithm. Prompts include simple instructions, standard 5-shot examples, chain of thought with 3-shot, and recursive path evaluation with 3-shot for in-context learning (see Section 3.3).

Experiment Environment. Our system allows search-based pathfinding in a continuous environment with modules for environment management, agent control, and visualization (see Section 4.1).

- **Environment Management:** Configures the environment and provides feedback.
- **Agent Control:** Customizes the agent's operations using the algorithm and model.
- **Visualization:** Offers real-time and final visual outputs for analysis.

Experiment Subject. Our experiments focus on two critical aspects: efficiency and scalability. For **efficiency**, we assess the number of operations and the storage required for the pathfinding process, defined as time and space complexity, respectively. Additionally, we evaluate the generated path length to assess path efficiency. These metrics are used to compute a composite efficiency score, as presented in Table 1. Larger environments and maps are employed to better illustrate algorithm efficiency, as they offer a more comprehensive reflection of the algorithm's performance under increased complexity. Specifically, we conducted efficiency experiments on a 50×30 map of the original sample size. This size was selected as it provides a substantial basis for evaluating efficiency while keeping the computational demands within a manageable range. Beyond this scale, the experiment run times become excessively long. For **scalability**, we tested both A* and LLM-A* algorithms across 10 different scales, from 1 to 10, to examine how they adapt to progressively larger environments, as depicted in Figure 3.

4.3 Evaluation Metrics

We assess **LLM-A*** against A* using metrics for operation efficiency, storage efficiency, and path quality. Performance is summarized by the geometric mean of performance ratios between **LLM-A*** and A* for operation, storage, path length, offering a balanced view less affected by outliers.

Operation and Storage Ratios. We compute the geometric mean of the ratios of operations and storage used by **LLM-A*** relative to A* ($\frac{\text{LLM-A*}}{\text{A*}}$). A lower score indicates better efficiency, e.g., a 50% score means **LLM-A*** uses 50% of the resources compared to A*.

Relative Path Length. Path quality is evaluated by comparing the path lengths from **LLM-A***, A* and LLM-only approach to the optimal path. The geometric mean of these ratios indicates how close **LLM-A*** paths are to optimal.

Valid Path Ratio. This metric measures the proportion of successful pathfinding attempts, often indicating that the generated path is collision-free and completable. A higher ratio indicates better reliability, showing the algorithm's effectiveness in generating valid paths consistently.

Growth Factor. We assess how performance scales from a 50×30 environment to larger sizes by calculating the arithmetic mean of the growth factors for operations and storage. This normalizes efficiency and scalability across different environment sizes.

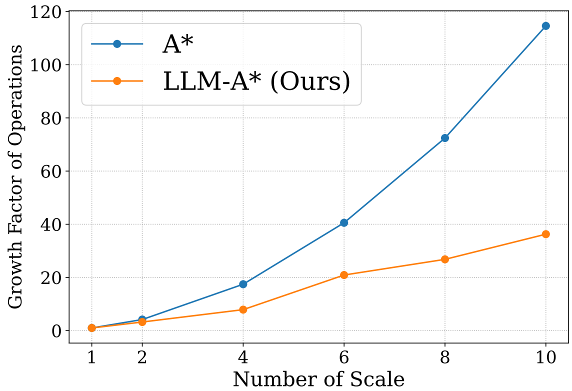
4.4 Quantitative Analysis

Table 1 presents a comparative analysis of three pathfinding methodologies: the classical A* algorithm, an LLM-only approach, and our proposed **LLM-A*** approach. The A* algorithm serves as the baseline, with an index value of 100 indicating performance equivalent to A*, as outlined in Section 4.3. The methodologies are evaluated on maps 50×30 of original map sizes.

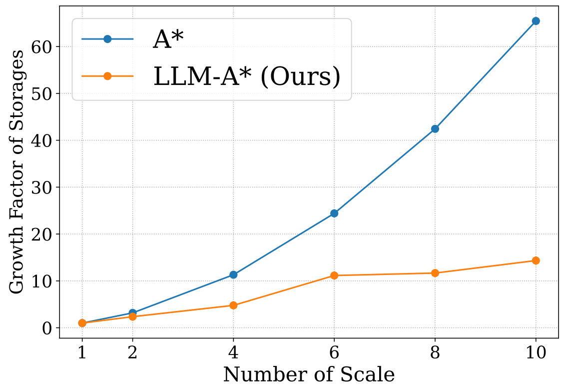
The results demonstrate that **LLM-A*** significantly enhances both operation and storage efficiencies compared to A*. Specifically, when utilizing the **LLM-A*** model, **GPT-3.5** achieves a 57.39% score in operations and a 74.96% score in storage, with a modest 2.44% increase in relative path length. Superior, with the **LLAMA3** model, **LLM-A*** reduces operations by 44.59% and storage by 64.02%, accompanied by a slight 2.47% increase in relative path length. These results highlight that **LLM-A*** not only reduces resource consumption but also maintains path validity, consistently achieving a valid path ratio of 100% across all scenarios. The observed increase in path length remains relatively low compared to the optimal path.

Meanwhile, the LLM-only approach underperforms compared to **LLM-A*** and A* algorithms in terms of both path efficiency and validity. When used in isolation, LLMs may struggle with comprehensive path planning due to their lack of heuristic guidance, which is provided by **LLM-A***, or the deterministic guarantees inherent in A*. The integration of LLM insights in **LLM-A*** significantly enhances its operational and storage efficiencies, surpassing the performance of A*.

Ablation Analysis. Notably, the Recursive Path Evaluation (RePE) prompting method achieves the smallest increases in relative path length in **LLM-A***, with increments of 2.41% for the **GPT-3.5** models, respectively. This suggests that RePE's step-by-step progression and intrinsic reasoning capabilities improve the models' ability to generate more optimal waypoints, resulting in more efficient paths. However, RePE underperforms compared to Chain of Thought (CoT) and few-shot prompting when used in the LLM-only approach. This indicates limitations in LLMs' ability to execute end-to-end path planning and spatial-temporal reasoning, which not only affects their proficiency in sequentially reasoning out detailed path sequences but also leads to issues such as hallucinations and misunderstandings. These limitations can cause the model to generate incorrect or implausible paths, undermining the effectiveness of LLMs in isolated path planning tasks.



(a) OPEARTION



(b) STORAGE

Figure 3: The comparative analysis examines the computational and memory efficiency between A* and **LLM-A*** (incorporating **LLAMA3** with few-shot prompting) across scaled environments ranging from 1 to 10 times enlargement, based on the means of 10 trials of random sampling. A* exhibits exponential growth in both (a) OPERATION and (b) STORAGE with linear increasing, environment scale, in contrast, **LLM-A*** achieves a near linear scalability.

Scalability Analysis. Figure 3 provides a comparative analysis of the computational and memory efficiency of the A* and LLM-A* algorithms

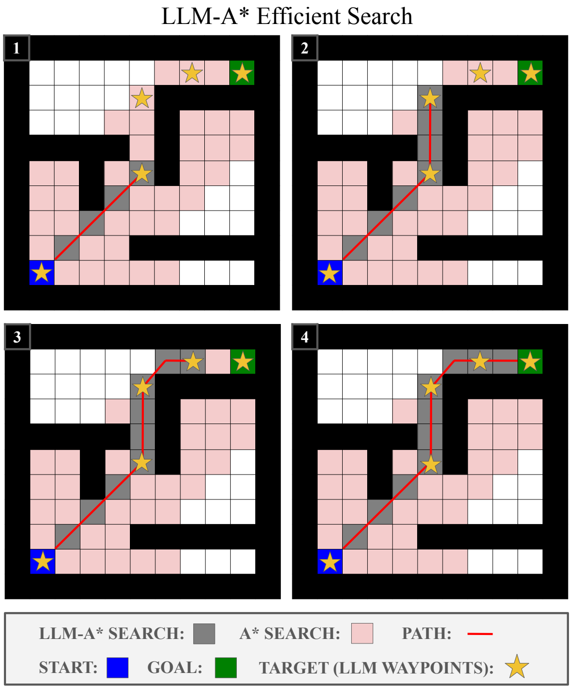


Figure 4: Visualization of pathfinding process with **LLM-A*** algorithms (under **chebyshev heuristic** setting in 11×11 grid environment) utilizing each LLM-generated waypoint, as well as comparison with A* in number of explored states. The **blue** and **green rectangles** denote the start and goal states, respectively. **Grey rectangles** indicate the states explored by the **LLM-A*** algorithms, while **pink rectangles** represent states explored by A*. **Red line** illustrate the generated paths. **Stars** indicate LLM-generated waypoints. (See Section 4.5 for more)

across environments of different scales. The analysis is presented through two metrics: the growth factor of operations and the growth factor of storage, with respect to different environment scales.

The results from Fig. 3 indicate that **LLM-A*** significantly outperforms **A*** in both computational and memory efficiency across various environment scales. While A* grows exponentially in operations and storage, LLM-A* achieves near-linear scalability relative to the environment size. This performance advantage arises from the learning-based enhanced heuristic values incorporated into LLM-A*, which allow it to avoid unnecessary node exploration and facilitate a more direct search towards the goal. This adaptation proves especially effective in larger and more complex environments. The efficiency gains of LLM-A* are particularly noteworthy in environments scaled up to 10 times, where the inefficiencies of A* become increasingly pronounced.

4.5 Qualitative Analysis

From the visualization in Figure 1, **LLM-A*** identifies the optimal path with only 140 operations, less than one-fifth the 859 operations required by A*, as well as the storage reduction. Both algorithms utilize a priority queue that stores the f-cost of each reached state, with the state having the lowest f-cost selected for exploration. The fundamental distinction between the two algorithms lies in their calculation of the f-cost or heuristic values.

As illustrated in Figure 4, **LLM-A*** leverages heuristic values derived from LLM-generated waypoints in addition to standard heuristic from A*, resulting in a dynamic heuristic that changes as the algorithm progresses. This dynamic adjustment is achieved through switching to the next target state during search when the current target state is reached. Each time the target state changes, the heuristic values for all previously reached states are recalculated. This allows **LLM-A*** to steer the search direction towards areas deemed more favorable by the large model at various stages of the search.

In contrast, A* employs a static heuristic for each state, which remains unchanged throughout the search. This static approach can lead to extensive exploration of non-optimal paths, including dead-end areas in the environment.

5 Conclusion

In this work, we propose a novel path planning algorithm, **LLM-A***, which outperforms traditional algorithms like A* in terms of both computational and memory efficiency, as well as LLM-only approach in path robustness and optimality. **LLM-A*** integrates heuristic values derived from LLM-generated waypoints (serves as global insight), with the deterministic guarantees in the A* algorithm. This hybrid approach addresses the shortcomings of both LLM-only approach and the A* algorithm by combining their respective strengths. Furthermore, the methodology of **LLM-A*** retains the general applicability of A*, making it suitable for pathfinding tasks in a wide range of environments. Thus, **LLM-A*** serves as an effective alternative to A* algorithm for path planning, especially in large-scale scenarios.

Limitations

Although around 90% of the paths generated by **LLM-A*** are optimal, our algorithm does not guarantee optimal path. While these cases are relatively few, they indicate that the algorithm may sometimes yield paths that are not the shortest or most efficient. Future improvements could focus on enhancing the optimality of the generated paths to ensure more consistent performance. Our experiments mainly utilized **GPT-3.5-TURBO** and **LLAMA3-8B-16bit** with basic prompt techniques. Although these models and prompts were adequate to validate the robustness of the **LLM-A*** algorithm, we did not explore a wider array of models or advanced prompt engineering strategies. Further testing with additional models and varied prompting methods could provide more comprehensive insights into the algorithm's performance across different scenarios.

References

Mostafa Abdou, Artur Kulmizev, Daniel Hershcovich, Stella Frank, Ellie Pavlick, and Anders Søgaard. 2021. Can language models encode perceptual structure without grounding? a case study in color. *arXiv preprint arXiv:2109.06129*.

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

Shrivats Agrawal. 2023. Are llms the master of all trades?: Exploring domain-agnostic reasoning skills of llms. *arXiv preprint arXiv:2303.12810*.

Adi Botea, Martin Müller, and Jonathan Schaeffer. 2004. Near optimal hierarchical path-finding. *Journal of Game Development*, 1(1):7–28.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Tianshi Cao, Marc Law, and Sanja Fidler. 2019. A theoretical analysis of the number of shots in few-shot learning. *arXiv preprint arXiv:1909.11722*.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023a. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*.

Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas Roy, and Chuchu Fan. 2023b. Autotamp: Autoregressive task and motion planning with llms as translators and checkers. *arXiv preprint arXiv:2306.06531*.

Marc-Alexandre Côté, Akos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. 2019. Textworld: A learning environment for text-based games. In *Computer Games: 7th Workshop, CGW 2018, Held in Conjunction with the 27th International Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, July 13, 2018, Revised Selected Papers 7*, pages 41–75. Springer.

Douglas Demyen and Michael Buro. 2006. Efficient triangulation-based pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 942–947.

Peng Ding, Jiading Fang, Peng Li, Kangrui Wang, Xiaochen Zhou, Mo Yu, Jing Li, Matthew R Walter, and Hongyuan Mei. 2024. Mango: A benchmark for evaluating mapping and navigation abilities of large language models. *arXiv preprint arXiv:2403.19913*.

Paolo Fiorini and Zvi Shiller. 1998. Motion planning in dynamic environments using velocity obstacles. In *IEEE International Conference on Robotics and Automation*, pages 760–765. IEEE.

Dieter Fox, Wolfram Burgard, and Sebastian Thrun. 1997. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, 4(1):23–33.

Daniel Harabor and Alban Grastien. 2011. Online graph pruning for pathfinding on grid maps. In *Proceedings of the AAAI conference on artificial intelligence*, volume 25, pages 1114–1119.

Peter Hart, Nils Nilsson, and Bertram Raphael. 1968a. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. 1968b. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107.

Robert Holte, M Perez, R Zimmer, and A MacDonald. 1996. Hierarchical a. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 530–535.

Gabriel Ilharco, Rowan Zellers, Ali Farhadi, and Hannaneh Hajishirzi. 2020. Probing contextual language models for common ground with visual representations. *arXiv preprint arXiv:2005.00619*.

M Jansen and Michael Buro. 2007. Hpa* enhancements. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 3, pages 84–87.

Sertac Karaman and Emilio Frazzoli. 2011. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894.

Uwe Koch. 2011. Grid-specific feature of hpa*. In *Proceedings of the International Conference on Artificial Intelligence*, pages 135–142.

Sven Koenig, Maxim Likhachev, and David Furcy. 2004. Lifelong planning a. *Artificial Intelligence*, 155(1-2):93–146.

Richard E Korf. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109.

Richard E Korf. 1990. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211.

Richard E Korf, Michael Reid, and Stefan Edelkamp. 2001. Time complexity of iterative-deepening-a. *Artificial Intelligence*, 129(1-2):199–218.

Ehsan Latif. 2024. 3p-llm: Probabilistic path planning using large language model for autonomous robot navigation. *arXiv preprint arXiv:2403.18778*.

Steven M LaValle. 2006. *Planning Algorithms*. Cambridge University Press.

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

Alex Nash, Kenny Daniel, Sven Koenig, and Ariel Felner. 2007. Theta: Any-angle path planning on grids. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1177–1183.

Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Nick Barnes, and Ajmal Mian. 2023. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*.

Roma Patel and Ellie Pavlick. 2021. Mapping language models to grounded conceptual spaces. In *International Conference on Learning Representations*.

Judea Pearl. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Yasaman Razeghi, Robert L Logan IV, Matt Gardner, and Sameer Singh. 2022. Impact of pretraining term frequencies on few-shot numerical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 840–854.

Matthew Renze and Erhan Guven. 2024. Self-reflection in llm agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682*.

Laura Ruis, Jacob Andreas, Marco Baroni, Diane Bouchacourt, and Brenden M Lake. 2020. A benchmark for systematic generalization in grounded language understanding. *Advances in neural information processing systems*, 33:19861–19872.

Stuart J Russell. 1992. Memory-bounded heuristic search. *Artificial Intelligence*, 49(1-3):5–27.

Dhruv Shah, Michael Robert Equi, Błażej Osiński, Fei Xia, Brian Ichter, and Sergey Levine. 2023. Navigation with large language models: Semantic guesswork as a heuristic for planning. In *Conference on Robot Learning*, pages 2683–2699. PMLR.

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

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020a. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020b. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Tom Silver, Varun Hariprasad, Reece S Shuttleworth, Nishanth Kumar, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2022. Pddl planning with pretrained large language models. In *NeurIPS 2022 foundation models for decision making workshop*.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. 2023. 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.

Anthony Stentz. 1994. Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3310–3317.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. 2005. *Probabilistic Robotics*. MIT press.

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

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. 2023. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. *arXiv preprint arXiv:2307.02477*.

Zhengxuan Wu, Elisa Kreiss, Desmond C Ong, and Christopher Potts. 2021. Reascan: Compositional reasoning in language grounding. *arXiv preprint arXiv:2109.08994*.

Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. 2023. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*.

Cheng-Fu Yang, Yen-Chun Chen, Jianwei Yang, Xiyang Dai, Lu Yuan, Yu-Chiang Frank Wang, and Kai-Wei Chang. 2023. Lacma: Language-aligning contrastive learning with meta-actions for embodied instruction following. *arXiv preprint arXiv:2310.12344*.

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

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. 2023. Lumos: Learning agents with unified data, modular design, and open-source llms. *arXiv preprint arXiv:2311.05657*.

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

Gengze Zhou, Yicong Hong, and Qi Wu. 2024. Navgpt: Explicit reasoning in vision-and-language navigation with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 7641–7649.

A Admissible Heuristic and Optimality

In path planning algorithms such as A*, a heuristic function $h(n)$ is deemed admissible if it never overestimates the cost to reach the goal from any given node n. This ensures that the estimated cost from n to the goal does not exceed the actual lowest possible cost, thereby providing a lower bound on the true cost. An admissible heuristic guarantees that the A* algorithm will find an optimal solution, as it always explores the least costly path first.

The standard A* heuristic is often the Euclidean distance or straight-line distance between the current node and the goal, which is both admissible and consistent. This heuristic function accurately reflects the minimum possible cost in scenarios where there are no obstacles or other constraints that might alter the cost path.

However, the **LLM-A*** algorithm integrates an additional heuristic component, influenced by insights from large language models (LLMs), into the traditional A* heuristic function. Specifically, **LLM-A*** incorporates a modified heuristic $h_{LLMA*}(n)$ which includes an additional cost term that estimates the difficulty of transitioning from the current state to the target state, based on the learned patterns from the LLM. This adjustment effectively amplifies the traditional heuristic by adding a factor derived from the LLM's assessment of the state-space complexity and the likely transitions required.

Let $h_{A*}(n)$ represent the conventional heuristic, and $c_{LLM}(n)$ represent the cost component derived from the LLM insights. The modified heuristic can be expressed as:

$$h_{LLMA*}(n) = h_{A*}(n) + c_{LLM}(n)$$

The term $c_{LLM}(n)$ may include factors such as predicted transition costs, obstacle avoidance strategies, or other environmental complexities inferred by the LLM, through selected target states in target list. Consequently, the heuristic function $h_{LLMA*}(n)$ provides a more nuanced estimate of the cost to reach the goal, potentially guiding the search more effectively by leveraging the LLM's understanding of the domain.

While this enhanced heuristic expedites the search process by prioritizing paths that the LLM identifies as promising, it introduces a deviation from admissibility. By incorporating the additional cost $c_{LLM}(n)$, the heuristic may overestimate the true cost to the goal, particularly if the LLM-derived costs are overly conservative or based on non-optimal path predictions. This overestimation violates the admissibility condition because the total estimated cost $g(n) + h_{LLMA*}(n)$ could exceed the actual optimal path cost, where $g(n)$ is the cost from the start to the current node.

The implications of this non-admissibility are significant: while the LLM-A* heuristic can potentially lead to faster convergence towards the goal by focusing the search in promising regions of the state space, it compromises the guarantee of finding the optimal path. The trade-off between search efficiency and optimality must be carefully considered in the application of LLM-A*. In scenarios where the heuristic insights from the LLM offer substantial benefits in reducing search time and computational resources, the potential loss of optimality may be justified. However, for applications where finding the absolute optimal path is crucial, relying solely on an admissible heuristic might be preferable.

B Prompts in LLMs

This appendix outlines the prompting techniques used in our LLM-A* algorithm to generate paths between start and goal points while navigating around obstacles. We employed different prompting strategies to evaluate their effectiveness in guiding the model. Below are the details of each technique along with the templates used.

B.1 Standard 5-Shot Demonstration

In the standard 5-shot demonstration in Table 2, the model is provided with five examples (or demonstrations) to guide the generation of the path. Each example includes start and goal points, along with horizontal and vertical barriers. The model is prompted to generate a path by following the pattern observed in the examples.

B.2 Chain of Thought (CoT) Prompting

The chain of thought prompting technique in Table 3 provides a sequence of reasoning steps that the model follows to arrive at the final path. This technique includes a detailed thought process and evaluation for each step, helping the model to understand the rationale behind the path generation.

B.3 Recursive Path Evaluation (RePE)

In the recursive path evaluation technique shown Table 4, the model iteratively evaluates the path

Identify a path between the start and goal points to navigate around obstacles and find the shortest path to the goal. Horizontal barriers are represented as [y, x_start, x_end], and vertical barriers are represented as [x, y_start, y_end]. Conclude your response with the generated path in the format "Generated Path: [[x1, y1], [x2, y2], ...]".

Start Point: [5, 5]
Goal Point: [20, 20]
Horizontal Barriers: [[10, 0, 25], [15, 30, 50]]
Vertical Barriers: [[25, 10, 22]]
Generated Path: [[5, 5], [26, 9], [25, 23], [20, 20]]

[5 in-context demonstrations abbreviated]

Start Point: {start}
Goal Point: {goal}
Horizontal Barriers: {horizontal_barriers}
Vertical Barriers: {vertical_barriers}
Generated Path: **Model Generated Answer Goes Here**

Table 2: The template of the prompt we used for LLM-A* using standard 5-shot demonstration.

Identify a path between the start and goal points to navigate around obstacles and find the shortest path to the goal. Horizontal barriers are represented as [y, x_start, x_end], and vertical barriers are represented as [x, y_start, y_end]. Conclude your response with the generated path in the format "Generated Path: [[x1, y1], [x2, y2], ...]".

Start Point: [5, 5]
Goal Point: [20, 20]
Horizontal Barriers: [[10, 0, 25], [15, 30, 50]]
Vertical Barriers: [[25, 10, 22]]
Thought: Identify a path from [5, 5] to [20, 20] while avoiding the horizontal barrier at y=10 spanning x=0 to x=25 by moving upwards and right, then bypass the vertical barrier at x=25 spanning y=10 to y=22, and finally move directly to [20, 20].
Generated Path: [[5, 5], [26, 9], [25, 23], [20, 20]]

[3 in-context demonstrations abbreviated]

Start Point: {start}
Goal Point: {goal}
Horizontal Barriers: {horizontal_barriers}
Vertical Barriers: {vertical_barriers}
Generated Path: **Model Generated Answer Goes Here**

Table 3: The template of the prompt we used for LLM-A* using standard 3-shot demonstration with chain of thought generation process.

at each step and makes decisions based on previous iterations. This process involves selecting points, evaluating their effectiveness, and adjusting the path as necessary to avoid obstacles and reach the goal.

C Details of Dataset Construction

The dataset for A* path planning is generated using a custom Python script, leveraging several key packages for randomization, geometric manipulation, visualization, and data management. The process involves the following steps:

1. **Initialization**: The script initializes with specified map dimensions (x and y boundaries) and parameters (number of barriers and obstacles) for the number of unique environments and start-goal pairs.

2. **Environment Creation**: For each map configuration, do the following:

   - Random obstacles, horizontal barriers, and vertical barriers are generated within defined x and y ranges using the `shapely.geometry.LineString` for line segments.
   - Start and goal points are randomly

Identify a path between the start and goal points to navigate around obstacles and find the shortest path to the goal. Horizontal barriers are represented as [y, x_start, x_end], and vertical barriers are represented as [x, y_start, y_end]. Conclude your response with the generated path in the format "Generated Path: [[x1, y1], [x2, y2], ...]".

Start Point: [5, 5]
Goal Point: [20, 20]
Horizontal Barriers: [[10, 0, 25], [15, 30, 50]]
Vertical Barriers: [[25, 10, 22]]
- First Iteration on [5, 5]
Thought: The horizontal barrier at y=10 spanning x=0 to x=25 blocks the direct path to the goal. To navigate around it, we should move to the upper-right corner of the barrier.
Selected Point: [26, 9]
Evaluation: The selected point [26, 9] effectively bypasses the horizontal barrier, positioning us at its corner and maintaining progress toward the goal without encountering additional obstacles.
- Second Iteration on [26, 9]
Thought: Now that we have bypassed the horizontal barrier, the path to the goal seems clear.
Selected Point: [20, 20]
Evaluation: The path is obstructed by the vertical barrier, leading to a collision. A more effective route involves moving around this vertical barrier.
Thought: To bypass the vertical barrier at x=25, we should move along its length and then turn around it to continue toward the goal.
Selected Point: [25, 23]
Evaluation: The selected point [25, 23] successfully avoids the vertical barrier and brings us closer to the goal without encountering further obstacles.
- Third Iteration on [25, 23]
Thought: From this position, there are no barriers directly obstructing the path to the goal.
Selected Point: [20, 20]
Evaluation: The path to the goal is clear from here, allowing a direct move to the goal.
Generated Path: [[5, 5], [26, 9], [25, 23], [20, 20]]

[3 in-context demonstrations abbreviated]

Start Point: {start}
Goal Point: {goal}
Horizontal Barriers: {horizontal_barriers}
Vertical Barriers: {vertical_barriers}
Generated Path: <u>**Model Generated Answer Goes Here**</u>

Table 4: The template of the prompt we used for LLM-A* using standard 3-shot demonstration with recursive path evaluation generation process.

placed on the map, ensuring they do not intersect with any obstacles. Valid pairs form non-intersecting line segments.

3. **Data Storage**: The generated environments, including the obstacles and start-goal pairs, are stored in JSON format.

4. **Query Generation**: Natural language queries are appended to each start-goal pair. These queries describe the task of finding a path that avoids the obstacles, which is supported as text input for LLMs.

5. **Visualization**: The environments are visualized using `matplotlib`, displaying the grid, obstacles, and paths. The plots are supported to be saved as image files for reference and stream in a show..

The Python packages utilized include:

- `random`: For generating random coordinates.
- `shapely`: For geometric operations, specifically creating and validating the positions of obstacles and points.
- `matplotlib`: For plotting and saving visual representations of the environments.
- `inquirer`: For command-line prompts to

make user decisions during dataset generation.

- `json` and `os`: For managing the reading and writing of dataset files.

- `search_env`: A custom package for environment setup and plotting specific to the search based path planning task.

This process ensures a comprehensive dataset with varied environments and queries, suitable for training and testing A* path planning algorithms.

D Evaluation Metric

In this study, we evaluate the performance of our algorithm using the geometric mean of ratios. This metric provides a robust measure for comparing the efficiency and effectiveness of different path planning algorithms. Below, we outline the rationale for choosing this metric, the calculation procedure, and its advantages.

D.1 Rationale

The geometric mean of ratios is used in this study to assess the relative performance of different path planning algorithms or approaches. It provides a balanced evaluation by aggregating multiple performance ratios, ensuring that no single extreme value disproportionately affects the overall metric. This is particularly useful in scenarios where the distribution of ratios can be skewed, and a simple arithmetic mean might be misleading.

D.2 Calculation Procedure

Let R_i represent the ratio of performance measures (such as path length, computation time, or any other relevant metric) between the proposed algorithm and a baseline or reference algorithm for the i-th test case. The geometric mean G of N ratios is calculated as follows:

$$G = \left(\prod_{i=1}^{N} R_i \right)^{\frac{1}{N}} \tag{1}$$

The geometric mean G provides a multiplicative average, effectively normalizing the ratios and providing a single representative value that reflects the overall performance across all test cases.

D.3 Advantages

Using the geometric mean of ratios offers several benefits in the context of evaluating path planning algorithms:

1. **Sensitivity to Relative Changes**: The geometric mean is sensitive to the relative differences between performance measures, making it suitable for comparing ratios.

2. **Mitigation of Outliers**: Unlike the arithmetic mean, the geometric mean minimizes the impact of extreme values or outliers, providing a more stable and representative metric.

3. **Interpretability**: The geometric mean allows for easy interpretation of performance improvements or deteriorations. A geometric mean greater than 1 indicates that, on average, the proposed algorithm performs better than the baseline, while a value less than 1 suggests poorer performance.

4. **Scalability**: The geometric mean naturally scales with multiplicative factors, making it appropriate for comparing algorithms across different scales or units of measurement.