Title

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

Author

Meng, Silin

Publication Date

2025

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

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

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Computer Science

by

Silin Meng

2025

ABSTRACT OF THE THESIS

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

by

Silin Meng

Master of Science in Computer Science
University of California, Los Angeles, 2025
Professor Kai-Wei Chang, Chair

Path planning is a fundamental problem in robotics and autonomous systems, requiring the computation of efficient routes from a start to a goal while avoiding obstacles. Traditional pathfinding algorithms like A* guarantee valid paths but face scalability issues in large environments, incurring high computational and memory costs as the state space grows. Large language models (LLMs), on the other hand, have emerged as powerful AI tools capable of understanding context at a global level and providing high-level environmental insights. However, an LLM alone lacks precise spatial reasoning, often producing impractical or invalid routes in detailed path planning.

This thesis proposes a novel hybrid algorithm, dubbed LLM-A*, that integrates an LLM with the A* algorithm to combine their complementary strengths. The LLM component offers global guidance to narrow the search space and provide heuristic hints, while the A* component ensures local optimality and path validity. By leveraging the LLM's broad environmental awareness to inform the A* search, this approach significantly improves pathfinding efficiency and reduces memory usage without sacrificing the quality of the path. Results indicate that this LLM-assisted strategy can plan routes more efficiently in complex, large-scale maps, enhancing scalability.

The thesis of Silin Meng is approved.

Yiwei Wang

Nanyun Peng

Kai-Wei Chang, Committee Chair

University of California, Los Angeles

2025

To my family and loves

your support

and inspiration

made this journey possible.

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

I am deeply grateful to Professor Kai-Wei Chang and Professor Nanyun Peng and my mentor, Professor Yiwei Wang, for their unwavering support, invaluable mentorship, and insightful guidance throughout my Master's journey. I also extend my appreciation to Cheng-Fu Yang in the UCLA NLP Lab for the stimulating discussions and collaborative spirit that enriched this work. Furthermore, this thesis has been shaped by the collective efforts behind a published paper, made possible through collaboration with an exceptional team of researchers, whose contributions have been truly invaluable.

Chapter 1

Introduction

1.1 Motivation

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 [17, 21, 28]. 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 [13, 14, 21, 45].

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 [15, 17, 20, 26] encounter an exponential increase in computational and memory demands. This occurs because the pathfinding process can become sub-optimal (see Figure 1.1 and 1.2), 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 [7, 12, 31, 39, 54]. These models demonstrate capabilities in processing and reasoning over long-context input to provide valuable global insights that reflect their un-

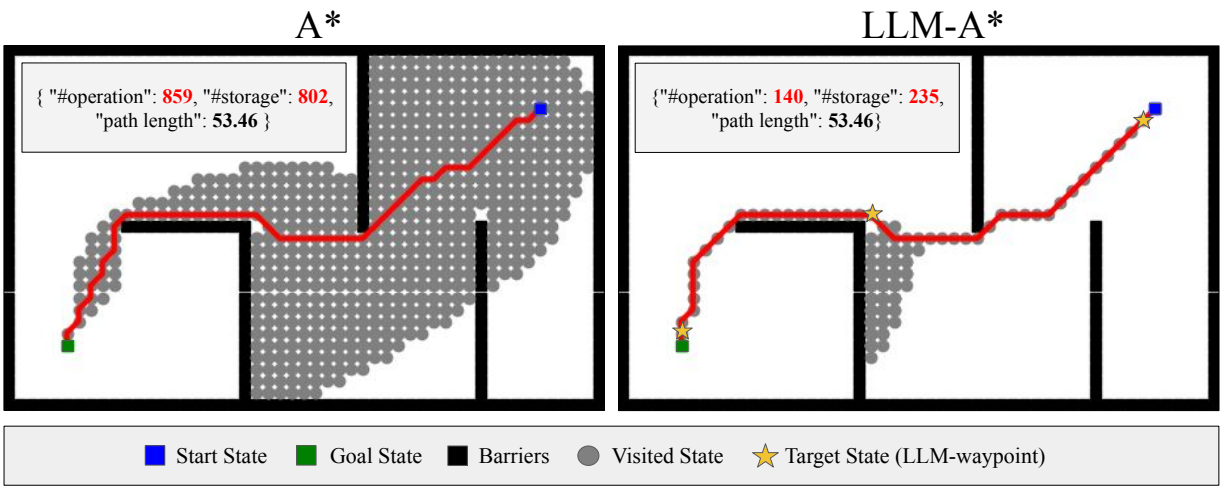

Figure 1.1: An comparison between **LLM-A*** and A* in computation and memory efficiency during pathfinding process. **LLM-A*** leverages target states generated by LLMs as waypoints to guide the searching process, significantly reducing the number of visited states, which leads to fewer operations and storage usage than A*.

derstanding 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 [2].

1.2 Thesis Overview

This thesis introduces **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. 1.1 and 1.2, 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.

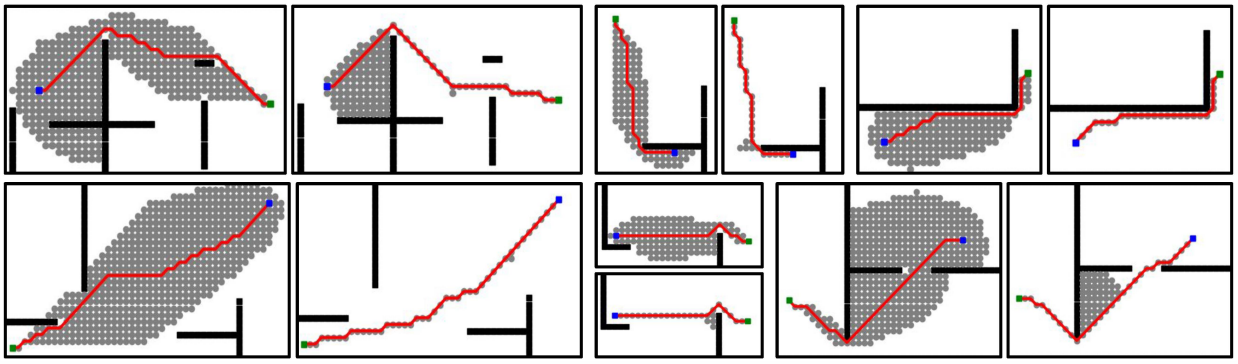

Figure 1.2: Visual comparison of pathfinding efficiency Between A* and **LLM-A***. This figure illustrates the performance differences between the traditional A* algorithm (left upper images) and the **LLM-A*** algorithm (right lower images). Red lines indicate the computed paths, blue dots mark the starting state, green dots indicate the goal state, gray areas represent visited states, and black lines denote obstacles. The **LLM-A*** demonstrates more efficient pathfinding by requiring significantly fewer visited states than A*.

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 4.1, 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 4.1, 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.

Chapter 2

Background

2.1 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 [16]. 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 [33].

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 [24]. Korf also introduced Learning Real-time A* (LRTA*) in 1990, incorporating real-time learning to dynamically update heuristic values, improving performance in changing environments [25]. 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 [37].

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 [44]. Koenig et al.'s Lifelong Planning A* (LPA*), introduced in 2004, incrementally updates paths in dynamic and large-scale environments [23]. 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 [15]. Nash's Theta*, from 2007, allows line-of-sight checks between nodes, resulting in more direct paths [30].

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 [18]. Botea et al.'s Hierarchical Path-finding A* (HPA*), introduced in 2004, improves transitions between abstraction levels for efficient large-map pathfinding [4].

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 [10]. Koch's Grid-specific Hierarchical Path-finding (GHPA*), introduced in 2011, optimizes grid maps pathfinding by integrating hierarchical and grid-specific optimizations [22].

2.2 Large Language Models in Path Planning.

Large Language Models (LLMs) have recently achieved remarkable success in natural language processing tasks and other domains [31]. Studies such as [38, 41, 43] explore LLMs in high-level planning, highlighting challenges in long-term planning and spatial reasoning [2]. 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 [1, 5, 19, 32, 52], yet face limitations in spatial reasoning and planning [3, 48, 50]. We introduce a benchmark for path planning

in continuous environments, integrating spatial and temporal reasoning. Prior benchmarks [9, 36, 40, 49] often neglect temporal planning aspects. Our study further evaluates LLMs in robot motion and path planning contexts, addressing limitations in end-to-end planning [8, 29, 42, 50].

Recent work, such as the LLM3 framework [46] leverages pre-trained LLMs to integrate symbolic task planning with continuous motion generation through motion failure reasoning, where LLM3 iteratively refines both symbolic actions and continuous parameters, significantly improving planning efficiency in dynamic environments, which aligns with our focus on LLMs' adaptability in correcting low-level planning errors and enhancing resilience in dynamic conditions.

Understanding the interplay between high-level and low-level planning is crucial [11, 27, 51, 56]. 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.

Chapter 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, and operates as follows:

1. **Initialization:** Begin by placing the start node s_0 in the OPEN list with $f(s_0) = g(s_0) + h(s_0)$. The OPEN list will store nodes yet to be explored, prioritized by their estimated cost to the goal. Simultaneously, initialize the CLOSED list as empty; this list will store nodes that have already been expanded to prevent redundant processing. Hence, the algorithm can efficiently manages its search space while maintaining optimality.

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. Start from the goal node and iteratively follow the recorded parent nodes until reaching the start node. The final reconstructed path represents the optimal sequence of moves, ensuring the shortest and most efficient route.

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 Algorithm 1, 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 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

Algorithm 1 LLM-A* Algorithm for Path Planning

Require: START state s_0, GOAL state s_g, OBSTACLE set obs, heuristic function $h()$, cost function $g()$, Large Language Model $llm()$
OPEN list $\mathcal{O}_{\text{open}} \leftarrow \{s_0\}$, CLOSE list $\mathcal{C}_{\text{close}} \leftarrow \{\}$, TARGET list $\mathcal{T} \leftarrow llm(s_0, s_g, \mathcal{O})$, TARGET state $t \leftarrow \mathcal{T}_{\text{start}}$, $g(s_0) \leftarrow 0$, $f(s_0) \leftarrow h(s_0)$, $P \leftarrow \{\}$
while $\mathcal{O}_{\text{open}} \neq \emptyset$ **do**
    $s_a \leftarrow \arg\min_{s \in \mathcal{O}_{\text{open}}} f(s)$
    if $s_a = s_g$ **then**
        return reconstruct_path(s_a)
    Remove s_a from $\mathcal{O}_{\text{open}}$
    Add s_a to $\mathcal{C}_{\text{close}}$
    for all neighbors s_n of s_a **do**
        if $s_n = t$ **and** $s_g \neq t$ **then**
            $t \leftarrow \mathcal{T}_{\text{next}}$
            Update f-cost of states in $\mathcal{O}_{\text{open}}$
        if $s_n \in (\mathcal{C}_{\text{close}} \cup \text{obs})$ **then**
            continue
        Tentative cost $g_{\text{tent}} \leftarrow g(s_a) + \text{cost}(s_a, s_n)$
        if $s_n \notin \mathcal{O}_{\text{open}}$ **or** $g_{\text{tent}} < g(s_n)$ **then**
            Update path to s_n to go through s_a
            $g(s_n) \leftarrow g_{\text{tent}}$
            $f(s_n) \leftarrow g(s_n) + h(s_n) + \text{cost}(t, s_n)$
            if $s_n \notin \mathcal{O}_{\text{open}}$ **then**
                Add s_n to $\mathcal{O}_{\text{open}}$
return failure

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 Algorithm 1:13-15). 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 Algorithm 1:20), 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* [15,22], 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 [6,34]. To investigate this further, we employed a few-shot learning technique, wherein we provides five demonstrations (See Table A.1 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 [47], 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 A.2 in Appendix) to the specific requirements of path planning.

Recursive Path Evaluation. The Recursive Path Evaluation (RePE) methodology (See Table A.3 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 [35, 53, 55] 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.

Chapter 4

Experiments

4.1 Dataset

Our dataset comprises 100 manually selected 50×30 maps, each containing 10 distinct start and goal positions, resulting in a total of 1000 samples (see Figure 1.1 for visualization). The dataset adheres to standard search-based algorithm environments in a continuous space, with each map characterized by specific structural parameters: the environment's x and y coordinate ranges, horizontal and vertical barriers, and designated start-goal pairs. The coordinate boundaries are defined as $[x_{\min}, x_{\max}]$ and $[y_{\min}, y_{\max}]$, while barriers are represented as lists of segments—horizontal barriers denoted by $[y, x_{\text{start}}, x_{\text{end}}]$ and vertical barriers by $[x, y_{\text{start}}, y_{\text{end}}]$. Each map contains 10 unique start and goal positions, ensuring diversity.

These structured parameters maintain consistency and relevance in experimental environments for search-based algorithms. Notably, the discretization of points and actions within this continuous framework is a necessary simplification that enables computational tractability and effective evaluation. Additionally, the dataset is designed to scale appropriately, allowing for robust scalability experiments and comparative performance assessments.

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 facilitates search-based pathfinding within a scalable framework designed for environments of varying complexity. It consists of modules for environment management, agent control, and visualization (see Section 4.1).

Environment Management: This module is responsible for defining the environmental parameters, such as obstacle placement, boundary conditions, and permissible agent movements. It dynamically configures the environment to introduce varying levels of difficulty and complexity, ensuring a diverse set of test scenarios. Additionally, it provides real-time feedback on environmental constraints, helping maintain the integrity of the experimental setup and preventing invalid configurations.

Agent Control: This module enables customization of the agent's operations based on the selected algorithm and model. It ensures that agent movements adhere to predefined constraints while maintaining the flexibility required for adaptive path planning. The agent operates within a discretized representation of the environment, executing precise movements based on waypoints or heuristic evaluations. By enforcing discrete action spaces, the system ensures computational tractability while allowing the evaluation of different planning strategies under uniform conditions.

Visualization: The visualization module provides comprehensive insights into the agent's navigation process. It offers real-time monitoring capabilities, enabling dynamic observation of the pathfinding process. Additionally, it generates final graphical representations of the computed paths, facilitating qualitative assessments of algorithmic performance. The visualization component is particularly useful for debugging, performance comparison, and

interpreting the decision-making processes of both traditional and LLM-based methods.

While the environments considered are presented within a continuous framework, both the LLM and A* algorithms operate on discrete points and actions. This necessary simplification allows us to effectively evaluate the proposed LLM-A* algorithm, ensuring the system remains applicable to a wide range of complex environments.

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 4.1. 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 4.1.

4.3 Evaluation Metrics

To comprehensively evaluate the effectiveness of **LLM-A*** in comparison to A*, we employ a set of well-defined performance metrics that assess computational efficiency, memory usage, and path quality. These metrics provide a quantitative basis for understanding how **LLM-A*** improves upon traditional methods while maintaining path validity and optimality. Our evaluation framework ensures that results remain interpretable and robust, enabling direct comparisons across different scales and experimental conditions.

Operation and Storage Ratios. We measure the computational efficiency of **LLM-A*** by calculating the geometric mean of the ratios (operations and storage) usage relative to A*:

$$\text{Efficiency Ratio} = \frac{\text{LLM-A*}}{\text{A*}} \tag{4.1}$$

A lower efficiency ratio indicates that **LLM-A*** requires fewer operations or less memory to achieve the same pathfinding objectives, thereby improving overall resource efficiency. For instance, a score of 50 means **LLM-A*** uses only 50% of the resources compared to A*. This metric is particularly crucial for large-scale environments, where high computational and storage demands can become limiting factors in real-world applications.

Relative Path Length. The quality of the generated paths is assessed by comparing their lengths to an optimal reference. We compute the geometric mean of the path length ratios among **LLM-A***, A*, and the LLM-only approach:

$$\text{Path Length Ratio} = \frac{\text{Path Length of Algorithm}}{\text{Optimal Path Length}} \tag{4.2}$$

A lower ratio signifies paths that are closer to the optimal solution, indicating greater efficiency in path selection. By evaluating relative path lengths, we ensure that **LLM-A*** maintains a balance between computational savings and path optimality.

Valid Path Ratio. Path validity is a fundamental requirement in any navigation system. We define the valid path ratio as the proportion of pathfinding attempts that result in a successful, collision-free path. This metric provides insight into the reliability of the algorithm in different scenarios:

$$\text{Valid Path Ratio} = \frac{\text{Number of Valid Paths}}{\text{Total Paths Tested}} \tag{4.3}$$

A higher valid path ratio indicates a more robust planning approach, ensuring that generated paths are consistently executable in real-world applications.

Growth Factor. Scalability is a key consideration for deploying path planning algorithms in large environments. We analyze how performance scales from a 50×30 environment to larger sizes by computing the arithmetic mean of the growth factors for operations and storage:

$$\text{Growth Factor} = \frac{\text{Performance at Larger Scale}}{\text{Performance at Baseline Scale}} \tag{4.4}$$

This normalization provides a clear view of how computational and memory requirements evolve with increasing problem complexity. A near-linear growth factor suggests that **LLM-A*** adapts efficiently to larger-scale environments, whereas exponential growth may indicate scalability limitations.

By employing these comprehensive metrics, we ensure a rigorous evaluation of **LLM-A***, providing a balanced assessment of its computational efficiency, memory usage, path quality, and scalability. This structured approach allows us to benchmark its performance against traditional A* and LLM-only methods, ultimately demonstrating its advantages in practical path planning applications.

4.4 Quantitative Analysis

Table 4.1 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

Methodology	Base Model	Prompt Approach	Operation Ratio ↓ (%)	Storage Ratio ↓ (%)	Relative Path Length ↓ (%)	Valid Path Ratio ↑ (%)
A*	-	-	100	100	100	100
Dynamic WA* ($w = 2$)	-	-	60.91	78.53	100.24	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 4.1: Quantitative analysis of three pathfinding methodologies: the classical A* algorithm, dynamic weighted A* with initial weight of 2 and 0.99 decay, 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).

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.

When compared to other variants using non-admissible heuristics, LLM-A* demonstrates an superior performance in term of operation and storage efficiency. Dynamic weighted A* (with initial weight of 2 and 0.99 decay) employs a static logic to dynamically update the weight, but it lacks the flexibility inherent to LLM-A*. Consequently, dynamic weighted A* falls short in terms of both operation efficiency and storage efficiency compared to our advanced LLM-A* approach.

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*.

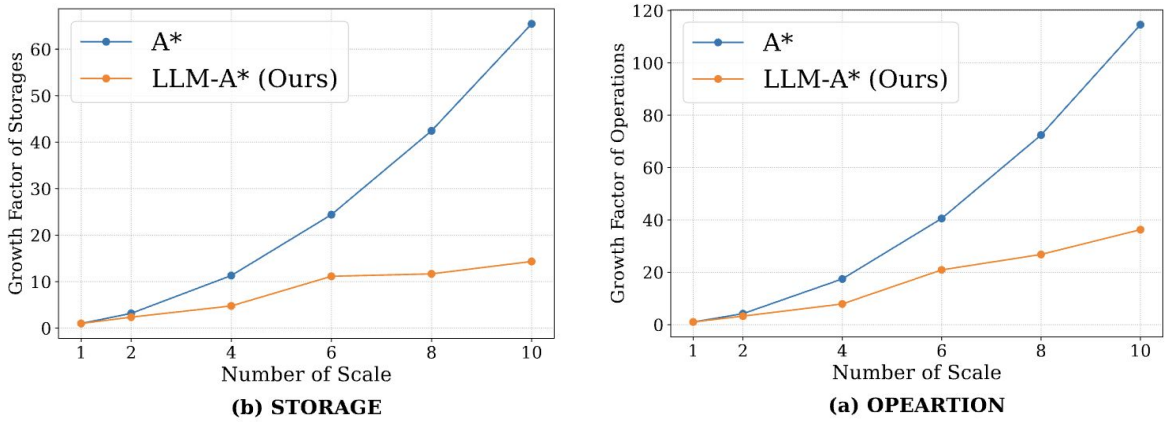

Figure 4.1: The comparative analysis examines the computational and memory efficiency between A* and **LLM-A*** (incorporating **LLAMA3** with few-shot prompting) across scaled environments ranging from 1 to 10 times enlargement, based on the means of 10 trials of random sampling. A* exhibits exponential growth in both (a) OPERATION and (b) STORAGE with linear increasing, environment scale, in contrast, **LLM-A*** achieves a near linear scalability.

Ablation Analysis. The Recursive Path Evaluation (RePE) prompting method achieves marginal improvements in relative path length for the **LLM-A*** approach using **GPT-3.5**, with an increment of 2.41%. This suggests some potential of RePE's step-by-step progression and intrinsic reasoning capabilities in generating more optimal waypoints, resulting in slightly more efficient paths. However, it is important to acknowledge that RePE underperforms compared to Chain of Thought (CoT) and few-shot prompting in a both operation and storage ratio, as well as efficiency in the LLM-only approach. This observation aligns with the limitations of LLMs in executing end-to-end path planning and spatial-temporal reasoning, which can affect their proficiency in sequentially reasoning out detailed path sequences and lead to issues such as hallucinations and misunderstandings, highlighting the diminish of RePE's efficiency for long-horizon tasks, where the intermediate points chosen using this technique are not optimal. Therefore, while RePE shows some promise, its overall effectiveness is limited compared to other methods in both LLM-A* and LLM-only scenarios.

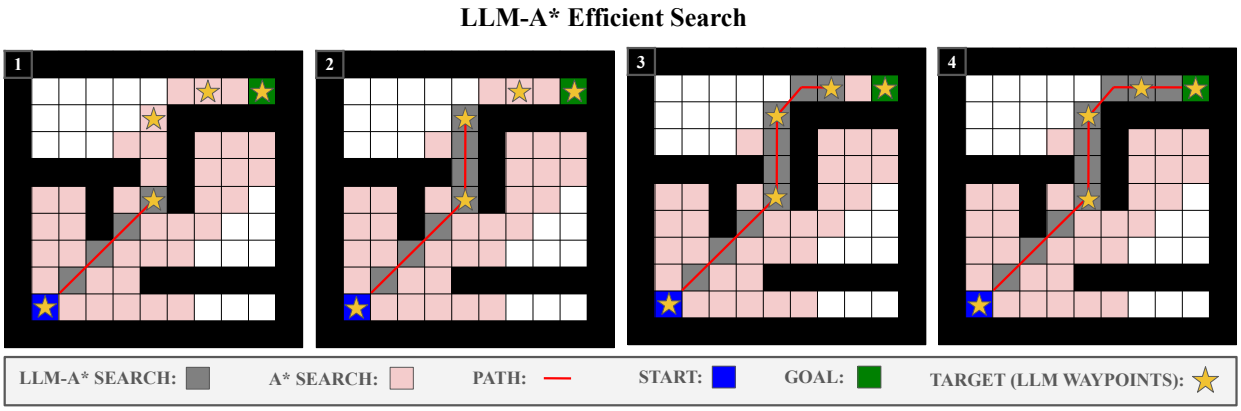

Figure 4.2: Visualization of pathfinding process with **LLM-A*** algorithms (under **chebyshev heuristic** setting in 11×11 grid environment) utilizing each LLM-generated waypoint, as well as comparison with A* in number of explored states. The **blue** and **green rectangles** denote the start and goal 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)

Scalability Analysis. Figure 4.1 provides a comparative analysis of the computational and memory efficiency of the A* and LLM-A* algorithms 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. 4.1 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 1.1, **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. In addition, as the map size increases, this operational efficiency difference could become more pronounced, as further illustrated in Figure 4.1.

As illustrated in Figure 4.2, **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.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This thesis introduced **LLM-A***, a hybrid path planning algorithm that integrates a large language model with A* to improve efficiency and scalability. By leveraging LLM-generated waypoints, LLM-A* significantly reduces computational cost and memory usage while maintaining path validity. Experimental results demonstrated that this approach scales more effectively than traditional A*, making it a promising option for large-scale and complex navigation tasks. The integration of LLMs with classical search algorithms opens new possibilities for autonomous robotics, enabling more efficient and context-aware decision-making.

5.2 Current Challenges

While Large Language Models (LLMs) have demonstrated strong reasoning capabilities, their application to path planning presents several challenges. One major limitation is their difficulty in precise spatial reasoning, leading to incorrect or inefficient paths. LLMs are trained primarily on textual data, lacking an inherent understanding of geometric constraints and obstacle avoidance. As a result, they often generate impractical waypoints or fail to account for environmental changes.

Another challenge is computational efficiency. LLMs require significant processing power, making them impractical for real-time applications where rapid path updates are necessary. Unlike classical algorithms such as A*, which can incrementally adjust paths as new data arrives, LLMs typically provide static responses that do not dynamically adapt to changes in the environment. This limitation makes them less suitable for scenarios that require continuous replanning.

Furthermore, LLMs can produce hallucinations—plausible but incorrect outputs—making their path recommendations unreliable without additional validation. Their lack of interpretability also raises concerns, as it is difficult to verify the reasoning behind their decisions. This contrasts with traditional algorithms, which follow explicit rules and cost functions, ensuring predictable and explainable behavior.

Finally, integrating LLMs with classical path planning methods remains an open challenge. While LLMs can assist in providing high-level waypoints or heuristic improvements, ensuring the validity and optimality of generated paths is non-trivial. Hybrid approaches must carefully balance the benefits of learned reasoning with the rigorous correctness of algorithmic search techniques.

5.3 Future Prospects

Despite these challenges, several promising directions can enhance the effectiveness of LLMs in path planning. One key approach is reinforcement learning on reasoning, which can help LLMs refine their spatial reasoning by interacting with simulated environments. By rewarding efficient and valid path generation, models can improve their ability to account for obstacles and optimize navigation strategies.

Advancements in LLM architectures and training methodologies also offer potential improvements. Future models may integrate geometric reasoning modules or leverage multimodal training that includes spatial data, allowing for better route generation. Additionally,

optimizing inference efficiency—such as through model distillation or quantization—can make LLMs more viable for real-time applications.

Another avenue for improvement is exploring alternative hybrid approaches that extend beyond A*. Integrating LLMs with sampling-based methods like RRT* or adaptive algorithms like D* could enhance their ability to navigate complex and dynamic environments. By combining the strengths of both data-driven reasoning and traditional search strategies, these hybrid methods may provide more robust and scalable solutions.

Practical deployment considerations, such as ensuring robustness in real-world scenarios and addressing safety concerns, will also be crucial. Future research should focus on making LLM-driven path planning more interpretable, reducing hallucinations, and improving real-time adaptability. By addressing these aspects, LLMs can become valuable tools in autonomous navigation, complementing established path planning techniques while expanding the scope of intelligent decision-making in robotics.

Appendices

Appendix A

Additional from Chapter 3

A.1 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:

$$h_{LLMA^*}(n) = h_{A^*}(n) + c_{LLM}(n) \tag{A.1}$$

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.

A.2 Waypoints Generation Prompts

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.

A.2.1 Standard 5-Shot Demonstration

In the standard 5-shot demonstration in Table A.1, 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.

A.2.2 Chain of Thought (CoT) Prompting

The chain of thought prompting technique in Table A.2 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.

A.2.3 Recursive Path Evaluation (RePE)

In the recursive path evaluation technique shown Table A.3, the model iteratively evaluates the path 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.

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 A.1: 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 A.2: The template of the prompt we used for LLM-A* using standard 3-shot demonstration with chain of thought generation process.

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: **Model Generated Answer Goes Here**

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

Appendix B

Additional from Chapter 4

B.1 Dataset Construction Details

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 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..

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

B.2 Evaluation Metric Details

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.

B.2.1 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.

B.2.2 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:

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

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.

B.2.3 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.

Bibliography

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

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

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

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

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

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

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

[9] Marc-Alexandre Côté, Akos 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 *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, 2019.

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

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

[12] Zi-Yi Dou, Cheng-Fu Yang, Xueqing Wu, Kai-Wei Chang, and Nanyun Peng. Reflection-reinforced self-training for language agents. *arXiv preprint arXiv:2406.01495*, 2024.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[28] Steven M LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[29] 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.

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

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

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

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

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

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

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

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

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

[39] 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.

[40] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 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, 2020.

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

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

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

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

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

[46] Shu Wang, Muzhi Han, Ziyuan Jiao, Zeyu Zhang, Ying Nian Wu, Song-Chun Zhu, and Hangxin Liu. Llmˆ 3: Large language model-based task and motion planning with motion failure reasoning. *arXiv preprint arXiv:2403.11552*, 2024.

[47] 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.

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

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

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

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

[52] Cheng-Fu Yang, Haoyang Xu, Te-Lin Wu, Xiaofeng Gao, Kai-Wei Chang, and Feng Gao. Planning as in-painting: A diffusion-based embodied task planning framework for environments under uncertainty. *arXiv preprint arXiv:2312.01097*, 2023.

[53] 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.

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

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

[56] Gengze Zhou, Yicong Hong, and Qi Wu. 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, 2024.