LLM-Enhanced Path Planning For Large-Scale Grid Maps

Anonymous ACL submission

Abstract

Path planning in grid maps is a fundamental problem in abroad range of applications. Existing works fall into traditional algorithms and learning-based algorithms. The traditional algorithms could generally guarantee completeness or optimality, but demand ultra-high computational and memory cost in large-scale maps. The learning-based ones adopt end-to-end learning paradigms that exploit priors to reduce computation, which however rely on task-specific pretraining and face scalability issues. Recently, Large Language Models (LLMs) have shown strong reasoning capacity for planning tasks, but suffer from spatial illusion and unstable performance in terms of long path length, long runtime and large memory occupation. Observing both the advantages and limitations of the SOTA LLM-enhanced algorithm, we propose iLLM-A*, an innovative LLM-enhanced efficient path planning algorithm for large-scale maps. iLLM-A* addresses the limitations of the SOTA LLM-Enhanced algorithm by integrating 4 modules: Optimal Standard A*, Waypoint Generating by Incremental Learning-Based LLM, Appropriate Waypoint Selection, and Circle-based Neighboring Waypoint Connection. iLLM-A* leverages the base LLM without fine-tuning, which bears the advantage of scalability. Finally, comprehensive evaluations are conducted to compare iLLM-A* with various baselines, using a set synthetic maps and a set of widely-adopted benchmarking maps related to 6 scenarios. The results show that iLLM-A* significantly outperforms the baselines.

1 Introduction

In a grid map with obstacles, path planning determines a collision-free path from a source to a destination, adhering to specific criteria such as minimizing distance, time, or energy (Liu et al., 2023a). This is a fundamental problem in a wide range of real-world applications, such as robot navigation (Carvalho and Aguiar, 2025), automated vehicle parking (Jiang et al., 2023), logistics scheduling (Hang et al., 2024), player role planning in game or emulated training environments (Panov et al., 2018), and route optimization in large-scale road networks (Luo et al., 2022; Ferrara et al., 2025; Liu et al., 2021; Farhan et al., 2023). In more scenarios, the need for path planning for large-scale grid maps boosts (Sun et al., 2024), specifically as robots' working space dramatically expands (Tang et al., 2025) and game maps are increasingly complex (Lee and Lawrence, 2013; Kirilenko et al., 2025).

Over the past decades, researchers have developed various path planning algorithms for grid maps, categorized into traditional and learning-based approaches. Traditional algorithms, such as A* and its extensions including JPS (Harabor and Grastien, 2011), Theta* (Nash et al., 2007), D* Lite (Koenig and Likhachev, 2002)), guarantee completeness or optimality in some cases but suffer from prohibitive computational costs on large-scale grids. For instance, the worst-case time complexity of A* can reach $O(N^4 \log N)$ (Carlson et al., 2023), where N is the edge length of the map and N^2 denotes the total number of grid cells, making it challenging to scale to very large environments. The learning-based methods (e.g., Neural A* (Yonetani et al., 2021), RLHA* (Ha et al., 2023), GraphMP (Zang et al., 2023)) leverage neural networks to learn heuristics, reducing computation by exploiting the learned priors. However, these methods typically rely on task-specific training, limiting their generalization to unseen environments and facing scalability issues in large-scale maps where training and inference becomes time-consuming.

In recent years, Large Language Models (LLMs) have showed remarkable reasoning capacities in complex contexts, inspiring their use in path planning(Fan et al., 2025; Aghzal et al., 2024; Xie et al.,

2024; Kwon et al., 2024), which, however, suffer from spatial illusion and therefore achieve unstable and limited planning performance. To achieve robust path planning, a state-of-the-art (SOTA) LLM-enhanced algorithm, LLM-A* (Meng et al., 2024), proposes a promising paradigm by combining the global insight of LLMs with the robust planning capacity of A*. Specifically, LLM-A* uses LLM to generate a series of waypoints from the source to the destination, restricting the A* solver to plan paths only between neighboring waypoints. The overall runtime and memory costs are thus significantly reduced. However, our preliminary experiments on LLM-A* reveal that: when applied for large-scale maps (with $N \geq 200$), LLM-A* also suffers from critical inefficiency in long runtime, large memory occupation and suboptimal path length. We then conduct thorough analysis on the source of inefficiency, and find that both A* and LLM cause inefficiency. Specifically, the A* implementation in LLM-A* needs to frequently maintain, update, and traverse some variable lists, causing high overhead. Meanwhile, the LLM may stochastically generate some inappropriate and redundant waypoints due to spatial illusion, leading to unnecessary node expansions.

Inspired by the promising paradigm of LLM-A* and the analyzed sources of inefficiency, this work proposes an innovative LLM-enhanced algorithm, abbr. as iLLM-A*. Unlike prior neural methods relying on fine-tuning or specialized training, iLLM-A* targets the inefficiency sources directly through a training-free design consisting of 4 modules: (I) **Optimal Standard A*** optimizes the underlying solver to replace linear lists with efficient heap/hash structures and applies a delayed update mechanism; (II) **Waypoint Generating by Incremental Learning-Based LLM** implements an incremental learning-based strategy to dynamically enrich few-shot prompts; (III) **Appropriate Waypoint Selection** leverages experience-driven waypoint selection to remove redundancy; and (IV) **Circle-based Neighboring Waypoint Connection** employs adaptive connection mechanisms to handle spatial variations. Our method are simple and only leverage base LLM for efficient path planning, which bears the advantages of scalability.

The contributions of the work are 3-fold:

• This work investigates iLLM-A*, a novel LLM-enhanced design for efficient path planning in large-scale grid maps. iLLM-A* requires no pre-training or environment-specific fine-tuning, which shows significant advantage on scalability for new tasks.

• We give a detailed design of iLLM-A* consisting of the aforementioned 4 modules that mitigate LLM spatial illusion and enhance A* implementation efficiency.

• Comprehensive evaluations are conducted on synthetic maps and widely-adopted benchmarking maps. Compared with multiple baselines, iLLM-A* achieves significant speedup, memory reduction and short path length. Extensive evaluations on different LLMs show the robustness of iLLM-A*.

2 Related Work

Traditional Path Planning Algorithms. Classical graph-search methods form the basis of path planning. Dijkstra's algorithm (DlJKSTRA, 1959) guarantees optimality but suffers from inefficiency on large grid maps, while A* (Hart et al., 1968) improves efficiency via admissible heuristics. Subsequent variants (Pearl, 1984; Korf, 1985; Koenig and Likhachev, 2002; Koenig et al., 2004) optimize memory usage and dynamic replanning, and hierarchical methods (Harabor and Grastien, 2011; Nash et al., 2007; Harabor and Grastien, 2013; Holte et al., 1996; Botea et al., 2004; Salvetti et al., 2018) introduce pruning strategies and multi-level abstractions. However, these methods still struggle with scalability on large-scale maps.

Learning-Based Path Planning Algorithms. Recent learning-based approaches (Yonetani et al., 2021; Ha et al., 2023; Zang et al., 2023; Pándy et al., 2022; Chen et al., 2024; Kirilenko et al., 2023, 2025; Reijnen et al., 2020; Odense et al., 2022) integrate neural networks into search frameworks to learn heuristics or capture global features, but generally require task-specific training and have limited generalization to unseen environments.

LLM-enhanced Path Planning Algorithm. LLMs have also inspired path planning methods, which leverages LLMs to generate semantic heuristics (Shah et al., 2023; Song et al., 2023; Zhou et al., 2024; Dai et al., 2024) or directly produce waypoints (Xie and Schwertfeger, 2024; Tariq et al., 2025). PPNL (Aghzal et al., 2024) introduced a benchmark exposing LLM's spatial instability. Complementarily, LLM-A* (Meng et al., 2024) integrates LLM's global reasoning with A's local optimality. However, these methods suffer from spatial illusion that even advanced reasoning strategies (Wei et al., 2022) only partially mitigate.

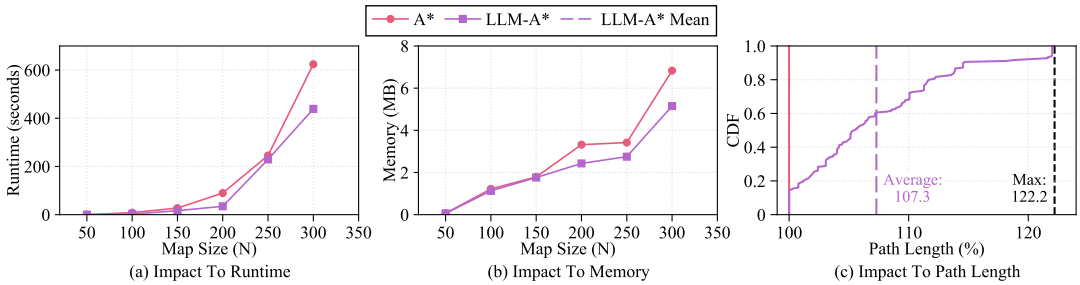

Figure 1: Runtime, Memory and Path Length Distribution of LLM-A* and A* Given Different Map Sizes.

3 Motivation

iLLM-A* is inspired by LLM-A* (Meng et al., 2024) . This section first conducts an evaluation of LLM-A* on different map sizes to show its inefficient performance. Then, the inefficiency source is analyzed.

3.1 Preliminary Evaluation of LLM-A*

We first test the performance of LLM-A* in large-scale grid maps (see detailed settings in Appendix B). Figure 1 depicts the runtime, memory occupation, and planned path length.

As Figure 1 (a) shows, when the map scales from $N = 200$ to $N = 300$, the runtime of LLM-A* exponentially increases from $30s$ to $7min$, which shows the inefficiency of LLM-A* for large-scale maps. Figure 1 (b) illustrates that memory shows a similar increasing trend. When the map scales from $N = 50$ to $N = 300$, the memory of LLM-A* increases by $84.75\times$. Regarding the path quality, the optimality gap is 7.3% on average, and up to 22.2% in extreme cases, which is far from the optimal/shortest path and not feasible in many applications. Figure 1 (c) demonstrates the distribution of the path length (related to the optimal/shortest path by A*) of all the maps with different scales.

3.2 Inefficiency Analysis For LLM-A*

By comprehesive analyzing LLM-A* and reviewing related study on LLMs, we have observed that the inefficiency is caused by both A* implementation and LLM limitations.

3.2.1 Limitation 1: Inefficient Implementation of A*

We find that the A* in LLM-A* needs to maintain, update and traverse some variable lists, which results in long runtime and large memory for large-scale maps.

The A* in LLM-A* uses two linear data structures for the OPEN and CLOSED lists. As detailed in Appendix B.2, the total time complexity is $O(N_{wp}N_{open}\log N_{open} + N_{open}N_{closed}N_{ob})$, where N_{wp} is the number of waypoints, N_{ob} is the obstacle number, and N_{open}/N_{closed} are the lengths of the OPEN/CLOSED lists. In the worst case, N_{open} and N_{closed} may approach N^2. Thereafter, the time complexity of LLM-A* is consistent with that of A*, as verified in Figure 1 (a). Such inefficient transversal operations are time-consuming for large-scale maps.

Global OPEN List Causing Large Memory Occupation: When planning the path between neighboring waypoints, A* needs to maintain the global OPEN list, resulting in memory costs similar to the original A*, as verified in Figure 1 (b).

3.2.2 Limitation 2: Spatial Illusion of LLM

The spatial illusion of LLM generates inappropriate and redundant waypoints, which result in long path length and runtime.

Suboptimal and Redundant Waypoints Due to Spatial Illusion: Some waypoints generated by LLM are stochastic and cannot precisely capture the shortest length objective, known as spatial illusion (Huang et al., 2023). The illusion may be caused by various factors, such as training data, training process, and reasoning process. LLM imitates text syntax instead of exactly learning to find the shortest path. Valmeekam et al. (Valmeekam et al., 2023) verified that the ratio of the advanced LLM successfully planning a mission is only 12%, which confirms the limited capacity of LLM. With redundant waypoints, LLM-A* incurs additional node expansions, resulting in long runtime.

Sparsely Distributed Obstacles Causing Long Path: Our extended experiments (see Appendix B) indicate that LLMs tend to generate worse waypoints within space with sparsely distributed obstacles thereby causing longer paths. Namely, the LLM-generated waypoint quality increases along with the increase of the obstacle density.

4 Detailed Design of iLLM-A*

The proposed iLLM-A* follows the basic paradigm of LLM-A* while addressing the aforementioned limitations. iLLM-A* integrates 4 carefully designed modules to constitute an efficient and scalable framework that reduces computational overhead and enhances stability in large-scale map environments.

4.1 Optimal Standard A*

4.1.1 Efficient Data Structures for A*

iLLM-A* employs the standard optimal A* as the low-level planner. The OPEN list is maintained by a binary heap and the CLOSED list is maintained by a hash set. The hash-based structure reduces the complexity of the search operation on the CLOSED list from $O(N_{closed})$ to $O(1)$ (Sun et al., 2007) (Sun et al., 2009). A hash map is used to store the g-values and parent of each grid. The heuristic is the Euclidean distance, and the search is performed on an 8-connected grid with unit cost for orthogonal moves and $\sqrt{2}$ for diagonal moves. The evaluation function is

$$f(s) = g(s) + h(s) + cost(s, s_{wp}), \qquad (1)$$

where s is a grid, s_{wp} is the active waypoint, $g(s)$ is the accumulated cost from the source, $h(s)$ is the estimated cost to the destination, and cost(s, s_{wp}) is the estimated cost to the waypoint. At each step, the node with the smallest f-value is extracted; for each neighbor, g and parent mappings are updated if a lower cost is found, and the corresponding f-value is recomputed. The final path is obtained by backtracking parent links from the destination.

4.1.2 Delayed Update for Global OPEN List

When $cost(s, s_{wp})$ changes, directly updating $f(s)$ in the global OPEN list demands significant overhead due to its large size. To address this issue, iLLM-A* applies a delayed update strategy (Sun et al., 2009):

- **Case** 1: only the top-k (i.e., $k = 100$) grids with the lowest estimated function in the OPEN list are updated.
- **Case** 2: when we extract a grid from the OPEN list, its heuristic function $f(s)$ is updated if $f(s)$ is outdated, and the $f(s)$ is estimated using the current value.

4.2 Waypoint Generating by Incremental Learning-Based LLM

The traditional static few-shot prompts of LLMs cannot effectively adapt to new environments (Song et al., 2023), often leading to over-fixed planning strategies. To address this limitation, an incremental learning mechanism leveraging the robust few-shot adaptation capacity of LLMs is introduced. Following the prompt engineering practices for Qwen models (Long et al., 2025) and principles from Natural Language Processing (Liu et al., 2023b), the prompts in iLLM-A* consist of three components: a prompt template, incremental learning-based few-shot example augmentation, and task instructions.

4.2.1 Prompt Template

The template first defines the LLM's role as a path planning specialist and specifies its primary destination: to generate an optimal path given the source location, destination locations, and obstacles. It then incorporates key constraints, including obstacle avoidance, a minimum waypoint number (set to 5 as in LLM-A*), and preferences for geometrically optimal paths. Under these constraints, the generated path is required to approximate the geometrically shortest path. The template further standardizes the input/output format in JSON and specifies systematic reasoning steps: (I) applying A* to compute a collision-free shortest path, (II) verifying the waypoint count, and (III) checking constraint satisfaction. A prompt example is presented in Appendix C.1.

4.2.2 Incremental Learning Based Few-shot Example Augmentation

A few-shot example repository (see Appendix C.2) contains validated map–waypoint pairs as in-context learning references. These examples are incrementally updated based on performance validation on a set of maps. The purpose of this incremental learning is to enrich the repository, enabling the LLM to progressively adapt its waypoint generation strategy to diverse environments.

Given a map, the LLM first generates waypoints using the current prompt, and the optimized A* then searches for the path guided by these waypoints, denoted as π_{LLM}. The optimal baseline path from the source to the destination, planned by the optimized A* alone, is denoted as π_{base}. The quality of π_{LLM} is evaluated by comparing its path length, runtime, and memory usage against

π_{base}. Only when π_{LLM} satisfies specific constraints regarding these three metrics, the corresponding map–waypoint example is considered high-quality and added to the repository. The repository maintains a fixed size, replacing the oldest examples with new validated ones. Detailed mathematical definitions of these metrics and the specific update thresholds are provided in Appendix C.4.

4.2.3 Task Instructions

This component provides explicit instructions for the current planning query by integrating specific environmental parameters (the source, the destination, and obstacles) with the template to activate problem-specific reasoning. An example of task instruction is presented in Appendix C.3.

4.3 Appropriate Waypoint Selection

Even with incremental learning, LLM-generated waypoints may contain redundant or deviating ones that compromise the efficiency of the A* algorithm. Inspired by the work (Karaman et al., 2011), we employ an empirically validated method to select the appropriate waypoints.

We compare 4 subset selection methods: Uniform Selection, Source-Prioritized Selection, Destination-Prioritized Selection, and Random Selection. Detailed settings, metrics and results are provided in Appendix D. The source method achieves the highest scores for Memory and Runtime using 1 or 2 waypoints, while maintaining the optimality gap within $6\% - 7\%$. The performance of all selection methods significantly degrades given > 2 waypoints. This is consistent with the findings that LLMs exhibit distance-dependent degradation in spatial validity, making waypoints farther from the source more likely to be infeasible (Farquhar et al., 2024; Valmeekam et al., 2022).

Based on the empirical results, we select the waypoints as follows: if the LLM-generated waypoint number is ≤ 2, all waypoints are used for planning; otherwise, only the first two waypoints closest to the source location are selected.

4.4 Circle-based Neighboring Waypoint Connection

The preliminary experiments above reveal that LLM-generated waypoints approach optimality in obstacle-dense environments but exhibit degraded quality in less constrained ones. To mitigate this spatial performance disparity, we implement an adaptive connection mechanism where the algorithm advances to the next waypoint when the current search node enters a circular region of radius r centered at the waypoint, rather than requiring exact coordinate matching. The connection radius r is defined as

$$r = \beta \cdot D \cdot (1 - \rho_{\text{local}}), \tag{2}$$

where $D = \sqrt{W^2 + H^2}$ is the map diagonal and β is a scaling coefficient empirically set to 0.15. The local obstacle density ρ_{local} is computed over a square window $\mathcal{W}$ of size $\lfloor 0.4D \rfloor \times \lfloor 0.4D \rfloor$ centered at the waypoint as

$$\rho_{\text{local}} = \frac{N_{\text{obs}}(\mathcal{W})}{N_{\text{total}}(\mathcal{W})}. \tag{3}$$

The algorithm transitions when the Euclidean distance between the agent state s and the current waypoint s_{wp} satisfies

$$\|s - s_{\text{wp}}\|_2 < r. \tag{4}$$

The mechanism produces tight connections ($r \to 0$) in highly constrained environments and relaxed connections (larger r) in less constrained areas, thereby minimizing the influence of suboptimal LLM-generated waypoints. We provide the complete pseudocode and a step-by-step description of the iLLM-A* algorithm in Appendix E.

5 Evaluation

5.1 Settings

5.1.1 Scenarios

We evaluate iLLM-A* on both synthetic maps and widely adopted benchmarking maps:

Synthetic Maps:We leverage the LLM to generate synthetic maps with edge lengths $N = 50, 100, \ldots, 450$. For the base map $N = 50$, we generate five test groups with obstacle counts ranging from 1 to 5; for larger maps, the obstacle number scales linearly with the map area. Each configuration includes 20 randomly generated instances, totaling 900 maps. To assess scalability, we also introduce challenging configurations with giant obstacles (cross-shaped and long bar-shaped) absent from the few-shot examples. Detailed generation rules are provided in Appendix F.1.

Benchmarking Maps: We evaluate performance on the MovingAI grid benchmarking maps(Sturtevant, 2012). The dataset includes six scenarios, scaled to $N = 384, 512, \ldots, 1024$. For

each scenario, we generate 100 test instances, totaling 600 tasks. Since most MovingAI maps are ultra-large, LLM-A* is not implemented due to its execution runtime$> 20min$.

5.1.2 Baselines

We compare iLLM-A* with 5 baselines, including 2 traditional algorithms, 2 learning-based algorithm and 1 LLM-Enhanced algorithm. **A*** (Hart et al., 1968): The exact heuristic graph search method, implemented as the standard binary-heap-based A* using the Euclidean distance as the heuristic function. **JPS** (Harabor and Grastien, 2011): A canonical search-acceleration method on uniform grids, implemented as the standard grid-based pruning variant of A*. **Neural A*** (Yonetani et al., 2021): A representative supervised-learning-based method. **RLHA*** (Ha et al., 2023): A representative reinforcement-learning-based method. Detailed training information are provided in Appendix F.2. **LLM-A*** (Meng et al., 2024): LLM-A* is the SOTA LLM-enhanced path planning method.

5.1.3 Metrics

We evaluate all methods using four metrics. **Pre-training Time (PreTime)**: This is the offline model training cost for learning-based methods, which is only applicable to Neural A* and RLHA*. **Runtime**: This is the runtime of an algorithm for planning the path. For iLLM-A* and LLM-A*, runtime includes both LLM query time and A* time. **Memory**: This is the maximum storage occupied by an algorithm when planning the path. **Path Length**: This is the ratio of the obtained path length to the optimal path computed by Dijkstra's algorithm, where 100% indicates optimality.

We implement iLLM-A* and the baselines in Python on a server equipped with Intel Xeon Silver 4216 CPU, NVIDIA V100 GPU, and 128 GB RAM running Ubuntu with CUDA 550. Both LLM-A* and iLLM-A* use Qwen-TURBO as the LLMs component to maintain strict experimental control.

5.2 Evaluation on Synthetic Maps

5.2.1 Overall Performance Comparison With LLM-A*

Table 1 demonstrates the runtime, memory and path length of LLM-A* and iLLM-A* given different map scales. iLLM-A* consistently outperforms LLM-A* across all metrics. On $N = 350$ maps, iLLM-A* achieves $1050\times$ ($0.85s$ vs. $892.63s$) speedups respectively. memory consumption is similarly improved, with iLLM-A* achieving 73.5% savings on $N = 350$ maps. path lengths are on average 102.39% of optimal across maps with $N \leq 300$, compared to 107.26% for LLM-A*, reducing the optimality gap by $\frac{7.26-2.39}{7.26} = 67.1\%$. These gains stem from the strategic waypoint generation that effectively reduces the exploration space of A* while incremental learning improves waypoint quality.

Map Size (N)	Runtime (s) ↓		Memory (MB) ↓		Path Length (%) ↓	
	LLM-A*	iLLM-A*	LLM-A*	iLLM-A*	LLM-A*	iLLM-A*
50	0.58	**0.54**	0.061	**0.059**	110.47	**100.01**
100	0.91	**0.55**	1.13	**0.64**	101.69	**101.03**
150	16.52	**0.55**	1.96	**0.98**	106.86	**104.5**
200	34.65	**0.61**	2.43	**1.02**	112.3	**103.26**
250	285.15	**0.73**	2.75	**1.57**	108.22	**103.63**
300	438.86	**0.82**	5.17	**2.14**	104.04	**101.93**
350	892.63	**0.85**	10.23	**2.71**	107.28	**103.17**
400	—	**0.85**	—	**2.54**	—	**102.37**
450	—	**1.02**	—	**3.62**	—	**104.00**

"—" indicates runtime exceeds 20 minutes.

Table 1: Performance Given Different Map Sizes.

Metrics	Method	Map Size (N)					
		200	250	300	350	400	450
Runtime (s) ↓	iLLM-A*	0.61	**0.73**	**0.82**	**0.85**	**0.85**	**1.02**
	w/o LLM	**0.57**	0.95	1.62	2.83	3.35	5.16
	w/o IncL	0.63	0.79	0.83	0.95	1.07	1.14
	w/o WDS	0.63	0.82	0.85	0.87	1.09	1.37
	w/o CNC	0.61	0.73	0.85	0.88	0.97	1.19
	w/o Opt-A*	27.04	54.17	173.54	278.96	381.96	601.27
Memory (MB) ↓	iLLM-A*	**1.02**	**1.57**	**2.14**	**2.23**	**2.54**	**3.62**
	w/o LLM	3.35	3.61	8.69	13.37	14.10	28.67
	w/o IncL	1.21	1.74	2.41	2.73	2.79	3.98
	w/o WDS	1.34	1.93	5.23	6.52	5.47	6.13
	w/o CNC	1.03	1.62	2.16	2.27	2.54	3.67
	w/o Opt-A*	1.06	1.63	2.14	2.28	2.62	4.13
Path Length (%) ↓	iLLM-A*	103.26	**103.63**	**101.93**	**103.17**	**102.37**	**104.00**
	w/o LLM	100.00	100.00	100.00	100.00	100.00	100.00
	w/o IncL	**102.83**	106.86	106.57	107.40	108.80	112.04
	w/o WDS	109.03	106.10	104.50	105.00	104.25	108.52
	w/o CNC	104.83	107.46	105.29	106.67	107.00	109.40
	w/o Opt-A*	105.74	104.30	103.66	106.35	105.20	106.24

Table 2: Ablation Study on Large-scale Maps.

5.2.2 Ablation Study

Table 2 summarizes the path length, runtime, and memory of iLLM-A* and its ablated variants across map sizes. iLLM-A* outperforms all variants. Removing the LLM (**w/o LLM**) increases runtime and memory usage as the environment scales; at $N = 450$, runtime rises from $1.02\ s$ to $5.16\ s$ ($5.1\times$) and memory from $3.62\ MB$ to $28.67\ MB$ ($7.9\times$). On smaller maps ($N = 200$), the variant without LLM is slightly faster ($0.57\ s$ vs. $0.61\ s$) because the fixed inference latency of LLM calls dominates runtime. Excluding incremental learning (**w/o IncL**) increases path length by an aver-

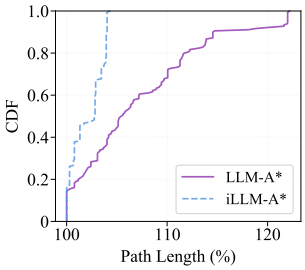

Figure 2: CDF of Path Length.

age of 4.4% and up to 112.04%, indicating its role in maintaining robustness across diverse environments. Disabling the waypoint distribution strategy (**w/o WDS**) raises runtime by $1.34\times$ and memory by $2.9\times$, because WDS helps reduce redundant node expansion. Removing the circle-based neighboring waypoint connection (**w/o CNC**) increases runtime (by $1.17\times$) and path length (by 3.4% on average); this aligns with preliminary results showing that CNC mitigates local detours. Excluding the optimal A* (**w/o Opt-A***) causes pronounced slowdowns (up to $589\times$).

5.2.3 Stability Analysis

In multi-robot coordination scenarios, path planning instability disrupts collaborative operations. To assess iLLM-A*'s consistency, we analyze path length stability. The Cumulative Distribution Function (CDF) of the path length is shown in Figure 2. All path lengths of iLLM-A* are shorter than 105%, while about 54% of the path lengths of LLM-A* are longer than 105%. Furthermore, as detailed in Appendix G.2, the standard deviation (Std) of the path length of iLLM-A* is much smaller than that of LLM-A* for all maps, which shows iLLM-A* is more consistent.

5.2.4 Scalability Analysis

To assess iLLM-A*'s generalization beyond the few-shot training examples, we evaluate performance on maps with giant obstacles absent from the few-shot example. iLLM-A* achieves near-linear scalability and over $1000\times$ speedup compared to LLM-A*, showing robustness even with unseen giant obstacles. Detailed scalability analysis is provided in Appendix G.1.

5.3 Evaluation on MovingAI Maps

We compare iLLM-A* with A* and JPS, Neural A* and RLHA* on the MovingAI maps (Sturtevant, 2012) . iLLM-A* uses one map of each map scenario as the few-shot example, maintaining the same Qwen-TURBO configuration above, without dataset-specific tuning. Unlike Neural-A* and RLHA* which require tens of hours of pre-training, iLLM-A* operates without training.

Figure 3 compares the runtime and memory of different algorithms in the 6 MovingAI scenarios. Overall, iLLM-A* is the most efficient algorithm. For runtime, iLLM-A* runs $16-121\times$ faster than A* and $10-28\times$ faster than JPS. It is slower than Neural-A* but has a similar runtime to RLHA*. This distinction stems from their inference mechanisms. Neural-A* is the fastest because it uses a convolutional encoder that runs once to produce global guidance. Therefore, its remaining search operation is greatly cut down. However, iLLM-A* is limited by the latency of calling the LLM. Its total runtime is close to RLHA*, because RLHA* embeds its neural network inside the node expansion loop and adds a lot of extra computation. iLLM-A* achieves about $2-17\times$ memory reduction compared with all baselines. This comes from two main reasons. First, iLLM-A* reduces the large number of node expansions in the traditional search. Second, it does not need to store the large network parameters required by learning-based methods.

iLLM-A* shows advantages in structured environments such as Baldur's Gate, where rectangular room layouts enable more accurate LLM-based waypoint generation. Whereas, RLHA* achieves lower latency in open terrain scenarios such as StarCraft, where the neural heuristics better accommodate irregular obstacle distributions. Neural-A* achievers $12.23\times-51.23\times$ larger memory due to the resident network parameters. The learning-based algorithms require retraining when deployed in dynamic environments (Hüppi et al., 2022; Wang et al., 2020). Training data further limit their application: each learning-based algorithm requires hundreds of labeled maps for each environment type, whereas iLLM-A* operates with few-shot examples per scenario. Compared to A*, iLLM-A* achieves up to $121.38\times$ speedup on the 1024×1024 StarCraft, alongside $63.6-93.3\%$ memory reduction. Compared to JPS, iLLM-A* achieves $15.4-52.2\%$ lower memory and competitive runtime performance, up to $28.46\times$ faster on the 1024×1024 StarCraft. Path length analysis shows iLLM-A* produces near-optimal solutions with only 2.99% of the optimality gap, which is superior to RLHA* (3.58%). The maximum deviation (5.34%) occurs in the real-world city and street scenarios, where sparse obstacles result in suboptimal waypoints.

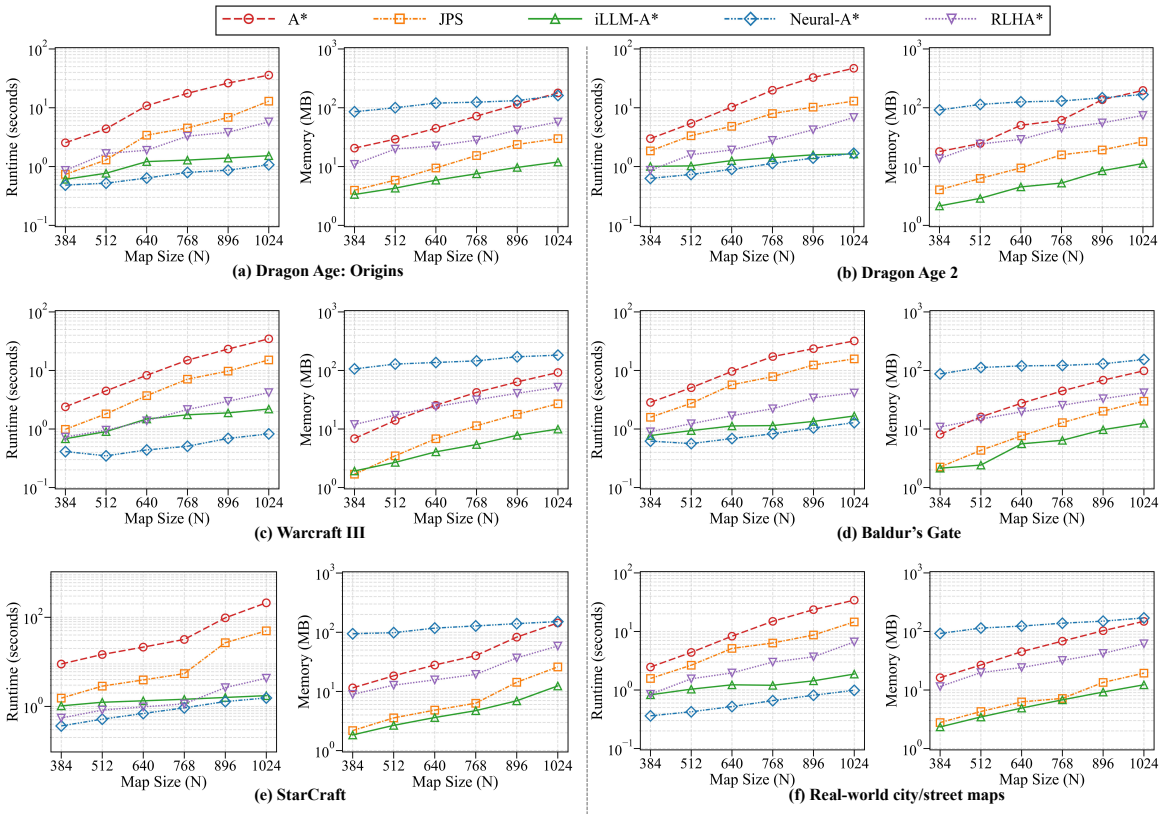

Figure 3: Performance Comparison Across Different Map Sizes on MovingAI Maps.

Map	Path Length (%) ↓				
	A*	JPS	iLLM-A*	Neural-A*	RLHA*
Baldur's Gate	100	100.35	101.57	101.29	103.23
Warcraft III	100	101.82	103.32	100.75	104.51
Dragon Age: Origins	100	102.34	102.46	101.57	103.40
StarCraft	100	100.47	103.27	101.16	105.35
Dragon Age 2	100	100.68	102.03	102.57	102.43
City/Street Maps	100	101.24	105.34	101.84	102.57
Average	100	101.15	102.99	101.53	103.58

Table 3: Path Length Comparison In Different Maps.

5.4 Comparison Over Different LLMs.

This section investigates compatibility of iLLM-A* to different LLMs. Our experiments mainly test **DeepSeek-V**3.2**-Thinking**, **Gemini-**2.5**-Pro**, and **Qwen-TURBO**. All experiments are conducted on the MovingAI maps with a grid size of $N = 512$. Detailed experimental data and analysis are provided in Appendix G.4. Experimental results show that iLLM-A* could achieve efficient path planning using different LLMs, while the specific advantages of different LLMs in reasoning may slightly impact the planning performance in terms of different metrics.

6 Conclusion

This work proposes iLLM-A*, a novel LLM-enhanced algorithm design for efficient path planning in large-scale grid maps. iLLM-A* consists of: (I) Optimal Standard A*, (II) Waypoint Generating by Incremental Learning-Based LLM, (III) Appropriate Waypoint Selection, and (IV) Circle-based Neighboring Waypoint Connection. Due to the remarkable reasoning ability of LLM, iLLM-A* requires no fine-tuning or pre-training. Comprehensive evaluations are conducted on both synthetic maps and MovingAI maps. iLLM-A* achieves competitive short path, the 2nd shortest runtime and the smallest memory, compared with the legacy solutions. Specificlly, iLLM-A* reduces the optimality gap of path length to 2.7%, which is only 37% of the gap of LLM-A* (7.3%) and approaches JPS (1.15%) and Neural-A* (1.53%), and better than RLHA* (3.60%). iLLM-A* achieves $28.46 - 1050\times$ speedup and $1.18 - 14.92\times$ memory cost reduction over most existing traditional, Learning-based and SOTA LLM-enhanced algorithms. Specifically, iLLM-A* achieves up to $1050\times$ speed up over LLM-A*, the SOTA LLM-enhanced algorithm. Besides, iLLM-A* achieves slightly longer runtime and comparable short path length with Neural-A* (The ever-known fastest algorithm with tens of hours of pre-training). Whereas, iLLM-A* requires no pre-training or fine-tuning, and saves up to $8.47\times$ memory cost over Neural-A*. iLLM-A* using different LLMs achieves similar superior performance.

Limitations

In our evaluation of LLM-enhanced path planning for large-scale grids, we acknowledge certain limitations regarding the evaluation scenarios and the inherent properties of LLMs.

Firstly, our study does not cover dynamic or unstructured environments, notably excluding excluding scenarios with moving obstacles. This focus on static grid maps limits the system's real-world application. This suggests that the ability to handle changing environments is an important area for future work.

Secondly, the efficacy of our framework relies on the inference speed of the LLM. For instance, while iLLM-A* significantly reduces the number of search nodes, the time cost of the LLM generation process can still be higher than simple traditional methods.

In addition, though iLLM-A* balances efficiency and path quality effectively, it prioritizes scalability over the optimality of path length. Moreover, for extremely large maps, the model may face a "context window" limit. This means the model cannot read all the map information at once if the input prompt becomes too long.

In summary, these limitations emphasize the need for future work to enhance the robustness and context handling of LLM-based planners.

Acknowledgments

References

Mohamed Aghzal, Erion Plaku, and Ziyu Yao. 2024. Can large language models be good path planners? a benchmark and investigation on spatial-temporal reasoning. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Adi Botea, Martin Müller, and Jonathan Schaeffer. 2004. Near optimal hierarchical path-finding. *J. Game Dev.*, 1(1):1–30.

Mark Carlson, Sajjad K Moghadam, Daniel D Harabor, Peter J Stuckey, and Morteza Ebrahimi. 2023. Optimal pathfinding on weighted grid maps. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 12373–12380.

José Pedro Carvalho and A Pedro Aguiar. 2025. Deep reinforcement learning for zero-shot coverage path planning with mobile robots. *IEEE/CAA Journal of Automatica Sinica*.

Dillon Z Chen, Sylvie Thiébaux, and Felipe Trevizan. 2024. Learning domain-independent heuristics for grounded and lifted planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20078–20086.

Zhirui Dai, Arash Asgharivaskasi, Thai Duong, Shusen Lin, Maria-Elizabeth Tzes, George Pappas, and Nikolay Atanasov. 2024. Optimal scene graph planning with large language model guidance. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14062–14069. IEEE.

EW DlJKSTRA. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik*, 50:269–271.

Haolin Fan, Xuan Liu, Jerry Ying Hsi Fuh, Wen Feng Lu, and Bingbing Li. 2025. Embodied intelligence in manufacturing: leveraging large language models for autonomous industrial robotics. *Journal of Intelligent Manufacturing*, 36(2):1141–1157.

Muhammad Farhan, Henning Koehler, Robert Ohms, and Qing Wang. 2023. Hierarchical cut labelling-scaling up distance queries on road networks. *Proceedings of the ACM on Management of Data*, 1(4):1–25.

Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. 2024. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630(8017):625–630.

Antonio Ferrara, David García-Soriano, and Francesco Bonchi. 2025. Beyond shortest paths: Node fairness in route recommendation. *Proceedings of the VLDB Endowment*, 18(9):3230–3242.

Junhyoung Ha, Byungchul An, and Soonkyum Kim. 2023. Reinforcement learning heuristic a*. *IEEE Transactions on Industrial Informatics*, 19(3):2307–2316.

Jinquan Hang, Zhiqing Hong, Xinyue Feng, Guang Wang, Dongjiang Cao, Jiayang Qiao, Haotian Wang, and Desheng Zhang. 2024. Complex-path: Effective and efficient node ranking with paths in billion-scale heterogeneous graphs. *Proceedings of the VLDB Endowment*, 17(12):3973–3986.

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.

Daniel Harabor and Alban Grastien. 2013. An optimal any-angle pathfinding algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 23, pages 308–311.

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

Robert C Holte, Maria B Perez, Robert M Zimmer, and Alan J MacDonald. 1996. Hierarchical a*: Searching abstraction hierarchies efficiently. In *AAAI/IAAI, Vol. 1*, pages 530–535.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2023. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43:1 – 55.

Matthias Hüppi, Luca Bartolomei, Ruben Mascaro, and Margarita Chli. 2022. T-prm: Temporal probabilistic roadmap for path planning in dynamic environments. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10320–10327. IEEE.

Zhuoling Jiang, Xiaodong Zhang, and Pei Wang. 2023. Grid-map-based path planning and task assignment for multi-type agvs in a distribution warehouse. *Mathematics*, 11(13):2802.

Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. 2011. Anytime motion planning using the rrt. In *2011 IEEE international conference on robotics and automation*, pages 1478–1483. ieee.

Daniil Kirilenko, Anton Andreychuk, Aleksandr Panov, and Konstantin Yakovlev. 2023. Transpath: Learning heuristics for grid-based pathfinding via transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12436–12443.

Daniil Kirilenko, Anton Andreychuk, Aleksandr I Panov, and Konstantin Yakovlev. 2025. Generative models for grid-based and image-based pathfinding. *Artificial Intelligence*, 338:104238.

Sven Koenig and Maxim Likhachev. 2002. D* lite. In *Eighteenth national conference on Artificial intelligence*, pages 476–483.

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.

Teyun Kwon, Norman Di Palo, and Edward Johns. 2024. Language models as zero-shot trajectory generators. *IEEE Robotics and Automation Letters*, 9(7):6728–6735.

William Lee and Ramon Lawrence. 2013. Fast grid-based path finding for video games. In *Canadian Conference on Artificial Intelligence*, pages 100–111. Springer.

Lixing Liu, Xu Wang, Xin Yang, Hongjie Liu, Jianping Li, and Pengfei Wang. 2023a. Path planning techniques for mobile robots: Review and prospect. *Expert Systems with Applications*, 227:120254.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023b. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM computing surveys*, 55(9):1–35.

Tiantian Liu, Huan Li, Hua Lu, Muhammad Aamir Cheema, and Lidan Shou. 2021. Towards crowd-aware indoor path planning. *Proceedings of the VLDB Endowment*, 14(8):1365–1377.

Do Xuan Long, Duy Dinh, Ngoc-Hai Nguyen, Kenji Kawaguchi, Nancy F Chen, Shafiq Joty, and Min-Yen Kan. 2025. What makes a good natural language prompt? *arXiv preprint arXiv:2506.06950*.

Zihan Luo, Lei Li, Mengxuan Zhang, Wen Hua, Yehong Xu, and Xiaofang Zhou. 2022. Diversified top-k route planning in road network. *Proceedings of the VLDB Endowment*, 15(11):3199–3212.

Silin Meng, Yiwei Wang, Cheng-Fu Yang, Nanyun Peng, and Kai-Wei Chang. 2024. Llm-a*: Large language model enhanced incremental heuristic search on path planning. *Findings of EMNLP*.

Alex Nash, Kenny Daniel, Sven Koenig, and Ariel Felner. 2007. Theta^*: Any-angle path planning on grids. In *Aaai*, volume 7, pages 1177–1183.

Simon Odense, Kamal Gupta, and William G Macready. 2022. Neural-guided runtime prediction of planners for improved motion and task planning with graph neural networks. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12471–12478. IEEE.

Michal Pándy, Weikang Qiu, Gabriele Corso, Petar Veličković, Zhitao Ying, Jure Leskovec, and Pietro Liò. 2022. Learning graph search heuristics. In *Learning on Graphs Conference*, pages 10–1. PMLR.

Aleksandr I Panov, Konstantin S Yakovlev, and Roman Suvorov. 2018. Grid path planning with deep reinforcement learning: Preliminary results. *Procedia computer science*, 123:347–353.

Judea Pearl. 1984. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc.

Robbert Reijnen, Yingqian Zhang, Wim Nuijten, Caglar Senaras, and Mariana Goldak-Altgassen. 2020. Combining deep reinforcement learning with search heuristics for solving multi-agent path finding in segment-based layouts. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pages 2647–2654. IEEE.

Matteo Salvetti, Adi Botea, Alfonso Gerevini, Daniel Harabor, and Alessandro Saetti. 2018. Two-oracle optimal path planning on grid maps. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, pages 227–231.

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.

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.

Nathan R Sturtevant. 2012. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):144–148.

Xiaoxun Sun, Sven Koenig, and 1 others. 2007. The fringe-saving a* search algorithm-a feasibility study. In *IJCAI*, volume 7, pages 2391–2397.

Xiaoxun Sun, William Yeoh, Po-An Chen, and Sven Koenig. 2009. Simple optimization techniques for a*-based search. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 931–936.

Yuekun Sun, Xiaochong Tong, Yi Lei, Congzhou Guo, Yaxian Lei, Haoshuai Song, Zige An, Jiayi Tang, and Yibo Wu. 2024. A multi-scale path-planning method for large-scale scenes based on a framed scale-elastic grid map. *International Journal of Digital Earth*, 17(1):2383852.

Jingtao Tang, Zining Mao, and Hang Ma. 2025. Large-scale multi-robot coverage path planning on grids with path deconfliction. *IEEE Transactions on Robotics*.

Muhammad Taha Tariq, Yasir Hussain, and Congqing Wang. 2025. Robust mobile robot path planning via llm-based dynamic waypoint generation. *Expert Systems with Applications*, 282:127600.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023. On the planning abilities of large language models-a critical investigation. *Advances in Neural Information Processing Systems*, 36:75993–76005.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2022. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Binyu Wang, Zhe Liu, Qingbiao Li, and Amanda Prorok. 2020. Mobile robot path planning in dynamic environments through globally guided reinforcement learning. *IEEE Robotics and Automation Letters*, 5(4):6932–6939.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Fujing Xie and Sören Schwertfeger. 2024. Empowering robot path planning with large language models: osmag map topology & hierarchy comprehension with llms. In *2024 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 707–712. IEEE.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: a benchmark for real-world planning with language agents. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Ryo Yonetani, Tatsunori Taniai, Mohammadamin Barekatain, Mai Nishimura, and Asako Kanezaki. 2021. Path planning using neural a* search. In *International conference on machine learning*, pages 12029–12039. PMLR.

Xiao Zang, Miao Yin, Jinqi Xiao, Saman Zonouz, and Bo Yuan. 2023. Graphmp: Graph neural network-based motion planning with efficient graph search. *Advances in Neural Information Processing Systems*, 36:3131–3142.

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 Additional Related Works

A.1 Traditional Path Planning Algorithms

Classical graph-search methods form the basis of path planning. Exact Dijkstra's algorithm (DIJKSTRA, 1959) finds the optimal paths but it becomes inefficient on large grid maps. Heuristic search subsequently emerged as a more efficient alternative.The A* algorithm (Hart et al., 1968) augments uniform-cost search with admissible heuristics, preserving optimality while significantly reducing unnecessary expansions. The Best-First Search (BFS) (Pearl, 1984) emphasizes heuristic-driven node expansion, which may yield suboptimal paths. The Iterative Deepening A* (IDA*) (Korf, 1985) achieves optimality with linear memory cost by combining depth-first and heuristic search. The D* Lite (Koenig and Likhachev, 2002) reuses previous search information for rapid replanning in dynamic environments, while the Lifelong Planning A* (LPA*) (Koenig et al., 2004) extends this idea to large-scale and repeated searches. The Jump Point Search (JPS) (Harabor and Grastien, 2011) accelerates grid-based A* by pruning symmetric nodes

through key "jump points". The Theta* (Nash et al., 2007) introduces line-of-sight checks, which enables any-angle paths closer to Euclidean optimality. The Hierarchical A* (HA*) (Holte et al., 1996) reduces the complexity through multi-level abstraction, and the HPA* (Botea et al., 2004) improves efficiency through refined inter-level transitions. The ANYA (Harabor and Grastien, 2013) advanced the optimal any-angle search by operating over continuous intervals instead of discrete nodes. More recently, the Two-Oracular System (SRC + JPS+) (Salvetti et al., 2018) integrates dual-oracle frameworks for faster optimal search through synchronized tie-breaking and path compression.

A.2 Learning-Based Path Planning Algorithms

Recent advances in learning-based path planning have progressively integrated neural networks into the search frameworks. Early differentiable formulations such as Neural A* (Yonetani et al., 2021) enables end-to-end optimization by coupling convolutional encoders with the A* search. The RLHA* (Ha et al., 2023) introduces online reinforcement learning to adapt heuristic estimates during exploration. Building on structural reasoning, GNN-based methods such as GraphMP (Zang et al., 2023) and PHIL (Pándy et al., 2022) learns the graph heuristics from spatial topology, which could be extended for general-purpose planning tasks (Chen et al., 2024). Transformer-based approaches further expanded the heuristic modeling: TransPath (Kirilenko et al., 2023) and its generative successor (Kirilenko et al., 2025) capture the global dependencies and instance-specific path priors for grid and image-based navigation. Complementary hybrid frameworks combines learning with search or prediction, as in WHCA*-RL (Reijnen et al., 2020) for multi-agent coordination and GNN-based runtime guidance (Odense et al., 2022) for planner selection.

A.3 LLM-enhanced Path Planning Algorithm

Recent progress in Large Language Models (LLMs) has inspired their application in path planning. Early explorations such as Navigation with LLMs (Shah et al., 2023) demonstrated that the commonsense knowledge extracted from LLMs could serve as semantic heuristics to guide traditional planners, hinting at their potential in spatial reasoning. Building upon this idea, the LLM-Planner (Song et al., 2023) and NavGPT (Zhou et al., 2024) extend LLMs to embodied and vision-language navigation tasks, where the textual reasoning was grounded in physical or visual contexts. To systematically evaluate such reasoning capability, the PPNL (Aghzal et al., 2024) introduced a benchmark for testing spatial–temporal planning capacity with few-shot or fine-tuned LLMs, exposing their instability in spatial consistency. In parallel, other research like the Empowering Robot Path Planning (Xie and Schwertfeger, 2024) and Robust Mobile Robot Path Planning via LLM-based Waypoint Generation (Tariq et al., 2025) focuses on enhancing map understanding and adaptive waypoint selection, thus improves the robustness in complex environments. Later works further emphasized structural reasoning, as exemplified by the Optimal Scene Graph Planning (Dai et al., 2024), which translates the natural-language goals into formal temporal logic for optimal task execution. Complementarily, LLM-A* (Meng et al., 2024) integrated the LLM's global reasoning with A*'s local optimality, providing a scalable hybrid paradigm for route planning. Despite these advances, most existing algorithms still rely heavily on the prompt design, thus exhibit limited geometric precision. Recent work (Wei et al., 2022) shows the reasoning strategies such as chain-of-thought prompting can only partially mitigate the instability of LLM in large-scale spatial domains.

A.4 Literature Summary

Traditional algorithms demand extremely high runtime and memory costs, which makes them unsuitable for large-scale maps. The learning-based methods depend on pretraining for specific tasks and struggle to scale effectively or adapt to new, unseen scenarios. Research on LLM-enhanced path planning has evolved from using prompts to directly guide LLM in map reasoning to integrating LLM reasoning with conventional planning methods. Some of the works are based on prompt engineering to help the LLM reason on small-scale maps. But these LLMs often struggle with spatial illusion. The other LLM-enhanced algorithms improve performance through fine-tuning on large-scale domain datasets and directly learning from visual inputs such as images. However, they still require large datasets and rely on high-quality prompt design, showing unstable performance on large-scale unseen maps.

The SOTA LLM-enhanced algorithm, LLM-A* (Meng et al., 2024), advances this by integrat-

Density (%)	$N = 50$		$N = 100$		$N = 150$	
	Red.	Path	Red.	Path	Red.	Path
70	0.35	1.02	0.33	1.02	0.33	1.03
50	0.39	1.08	0.40	1.06	0.38	1.07
30	0.46	1.12	0.47	1.12	0.45	1.11

Red.: Redundant Ratio; Path: Path Length.

Table 4: Impact of Obstacle Density.

ing the global reasoning capacity of LLMs with the local optimization of A*. However, LLM-A* suffers from redundant or inconsistent waypoints and results in limited performance. Motivated by the remarkable paradigm of LLM-A* and its limitations, an innovative LLM-enhanced path planning algorithm, iLLM-A*, is introduced for much more efficient path planning.

B Detailed Preliminary Analysis

This appendix provides the detailed experimental settings, spatial structure analysis, and complexity derivation supporting Section 3.

B.1 Experimental Settings and Spatial Analysis

The evaluation considers 2D square grid maps with equal edges N. The obstacle length is randomly assigned within 5 and 10 grids. For each configuration, 20 map instances are generated randomly. We define a waypoint as redundant if removing it and recomputing the path yields both a shorter runtime and a shorter path length. The redundant ratio is the ratio of redundant waypoints to the total number of waypoints. Table 4 shows the path length and redundant ratio. The results demonstrate that when obstacle density decreases from 70% to 30%, the optimality gap increases from $2\% - 3\%$ to $11\% - 12\%$, and the redundant ratio increases from 33% to 47%.

B.2 Complexity Derivation of A* in LLM-A*

The total time complexity of operating the OPEN list (including inserting and sorting) is $O(N_{wp} N_{open} \log N_{open})$, where N_{wp} is the waypoint number and N_{open} is the list length. The complexity of checking the CLOSED list is $O(N_{open} N_{closed})$. Detecting collisions requires $O(N_{open} N_{closed} N_{ob})$. Thus, the total complexity is:

$$O(N_{wp} N_{open} \log N_{open} + N_{open} N_{closed} N_{ob}) \quad (5)$$

In the worst case, N_{open} and N_{closed} may approach N^2, explaining the exponential runtime increase.

C Prompt Design and Incremental Few-shot Learning Details

This appendix outlines the prompting strategy and Incremental Few-shot Learning Details. We design a structured prompt consisting of three components. Below, we explain each component and refer to the corresponding templates.

C.1 Static Prompt Template

The template in Table 6 defines the query rules. It arranges the LLM as a planning expert. Key constraints include strict obstacle avoidance and a minimum waypoint count. The template also mandates a standardized JSON output to ensure the path is executable.

C.2 Incremental Few-shot Examples

We utilize incremental learning through the examples in Table 7. This module provides validated map-waypoint pairs as references. Crucially, we update these examples incrementally based on performance. This mechanism allows the model to adapt its strategy to diverse environments.

C.3 Task-Specific Instructions

This final component activates the specific planning task (see Table 8). It integrates the current source, destination, and barriers. By combining these details with the previous rules, it triggers the model to generate the final path.

C.4 Details of Incremental Learning

This appendix details the performance validation metrics and update mechanisms for the few-shot repository. As defined, π_{LLM} represents the path guided by LLM-generated waypoints, and π_{base} represents the optimal baseline path. The path length, runtime, and memory of π_{LLM} are evaluated using the following inequalities:

$$\frac{|Length(\pi_{LLM}) - Length(\pi_{base})|}{Length(\pi_{base})} \leq \theta_{length}, \quad (6)$$

$$\frac{Runtime(\pi_{LLM})}{Runtime(\pi_{base})} \leq \theta_{runtime}, \quad (7)$$

$$\frac{Memory(\pi_{LLM})}{Memory(\pi_{base})} \leq \theta_{memory}, \quad (8)$$

where $Length(\pi)$ is the path length, $Runtime(\pi)$ is the runtime, and $Memory(\pi)$ is the maximum occupied storage used for planning

path π. If π_{LLM} satisfies Eq.(6)–(8), it implies that its length is within $(1 + \theta_{length})$ of the shortest path, and its runtime and memory costs are within $\theta_{runtime}$ and θ_{memory} of the optimized A*, respectively.

In our implementation, the three thresholds θ_{length}, $\theta_{runtime}$, and θ_{memory} are set to 0.1. This strict filtering ensures the quality of the validated examples. The repository is augmented with a maximum capacity of 10 examples; if more maps become available, the oldest ones are replaced to maintain the repository's relevance and diversity.

D Waypoint Selection Study

D.1 Empirical Study Settings

Similarly to LLM-A*, we require LLM to generate at least 5 waypoints. The datasets used for this evaluation are the same as Section 5.2.1. We compare four subset selection strategies: **Uniform Selection** (uniformly choosing some waypoints, abbr. Uniform), **Source-Prioritized Selection** (prioritizing waypoints closer to the source, abbr. Source), **Destination-Prioritized Selection** (prioritizing waypoints closer to the destination, abbr. Destination), and **Random Selection** (randomly choosing waypoints, abbr. Random). For each method, we select 1 to 4 waypoints and evaluate performance, with each group repeated over 30 runs.

The evaluation adopts a two-stage metric computation process. First, we repeat the algorithm for 30 trials and obtain the average values of the path length, runtime, and memory. Then, we normalize the runtime and memory to $[0, 1]$:

$$Score_{normalized} = \frac{x_{max} - x_{current}}{x_{max} - x_{min}}, \quad (9)$$

where $x_{current}$ is the mean value of runtime or memory, and x_{max} and x_{min} are the maximum and minimum values across all trails for the given map scale. Larger $Score_{normalized}$ value indicates better performance. The final scores are obtained by averaging $Score_{normalized}$ across different map scales.

D.2 Empirical Results Analysis

Table 5 shows the source method achieves the highest scores of $\frac{0.973}{0.984}$ for Memory and $\frac{0.972}{0.982}$ for Runtime using $1 - 2$ waypoints, while maintaining the optimality gap of the path length within only $6\% - 7\%$. The performance of all the selection methods significantly degrades given > 2 waypoints.

Metrics	Method	Number of Selected Waypoints 1	2	3	4
Runtime Score ↑	Source	0.972	**0.982**	0.221	0.413
	Uniform	0.420	0.780	0.743	0.374
	Random	0.354	0.716	0.662	0.238
	Destination	0.323	0.235	0.477	0.275
Memory Score ↑	Source	0.973	**0.984**	0.234	0.406
	Uniform	0.405	0.778	0.745	0.338
	Random	0.633	0.711	0.674	0.223
	Destination	0.054	0.233	0.466	0.261
Path Length (%) ↓	Source	104	103	104	104
	Uniform	**102**	105	105	103
	Random	105	104	103	104
	Destination	105	103	104	103

Bold values indicate the best performance.

Table 5: Waypoint Selection Performance Given Different Methods.

E Pseudocode of iLLM-A*

Algorithm1 presents the comprehensive workflow of iLLM-A*. Lines 1–3 describe the initialization, where the *OPEN* list is initialized with the source s, and the LLM generates candidate waypoints using the incremental-learning prompt. The experience-driven selection mechanism determines the final waypoints. Lines 4–8 describe the main search loop, which expands the node with the lowest f-cost until the destination s_g is reached or the *OPEN* list becomes empty. Lines 9–13 describe the neighboring waypoints connection process. The target is updated once the search node enters the active region and relevant f-values are locally adjusted. Lines 14–24 describe the A*-style expansion guided by LLM-informed heuristics: tentative costs are evaluated, parent relations and f-scores are updated when improvement occurs, and feasible neighbors are added to *OPEN*. The loop terminates as shown in Lines 25–26, followed by path reconstruction in Line 27. Lines 28–29 describe the incremental update of the few-shot prompt repository based on the trajectory and performance.

F Detailed Experimental Settings

F.1 Details of Synthetic Maps Generation

The edge lengths are $N = 50, 100, \ldots, 450$. For the smallest map ($N = 50$), we generate five test groups with obstacle counts of 1 to 5. For larger maps ($N \geq 100$), the obstacle number increases linearly with the map area. For example, when $N = 450$, the baseline of 3 obstacles scales to

Table 6: Static Prompt Template for LLM Waypoint Generation.

Module I: Static Prompt Template

 # Role
You are an expert specializing in computational geometry and path planning.

 # Goal
Generate an optimal path from source point to destination point based on the given source position, destination position, and obstacle information.
 # Constraints
Strictly adhere to the following rules:
1. **Obstacle Avoidance**: The path must not contact or intersect any obstacles in any form.
2. **Path Point Count**: The final path must contain at least 5 coordinate points (including source and destination points). **Interpolation Rules**: If the initially calculated path contains fewer than 5 points, uniformly insert additional points between the longest path segments until the quantity requirement is satisfied. Inserted points should ensure path smoothness and avoid unnecessary sharp angles.
3. **Path Optimality**: Under the premise of satisfying all above constraints, the generated path should approximate the geometrically shortest path.
 # Input and Output Format
Input: Source point `Source: [x, y]`, Destination point `destination: [x, y]`, Horizontal barriers `horizontal_barriers: [[y, x_start, x_end], ...]`, Vertical barriers `vertical_barriers: [[x, y_start, y_end], ...]`
Output: Must strictly follow the JSON array format: `Generated Path: [[x1, y1], [x2, y2], ..., [xN, yN]]`
 # Workflow
1. **Initial Pathfinding**: Based on A* algorithm logic, identify a shortest path that avoids all obstacles.
2. **Verification**: Check the path point count. If fewer than 5 points, add intermediate points according to Core Constraint
3. **Final Validation**: Before output, verify that the generated path completely satisfies all core constraints.

$3 \times (450/50)^2 = 243$ obstacles. The width of each small obstacle equals one grid, and the length is randomly distributed between 5 and 10 grids. For extra giant obstacles (see Scalability Analysis), the length is randomly distributed within $50\% - 60\%$ of the edge length. One giant cross-shaped obstacle lies in the intermediate region, forcing feasible paths to detour far from the straight line. The other includes 3 long bar-shaped obstacles parallel to an edge.

F.2 Learning-based baselines Training Details

For the learning-based baselines mentioned in the main text, we applied the following retraining protocols:

Neural A* (Yonetani et al., 2021): It is retrained with the official network architecture and hyperparameters, except for batch size (32) and number of epochs (50). Each scenario category uses 500 training maps sampled from the MovingAI dataset, strictly separated from 600 test instances.

RLHA* (Ha et al., 2023): It is retrained following the original reinforcement learning setup with the same modifications to batch size (32) and epochs (50). The same dataset partitioning protocol as Neural A* is applied.

G Full Evaluation Results

G.1 Scalability Analysis

We insert two kinds of giant obstacles absent from the few-shot example, as illustrated inFigure 4. One giant cross-shaped obstacle lies in the intermediate region between the source and destination, forcing feasible paths to detour far from the straight line connecting them. The other includes 3 long bar-shaped obstacles parallel to an edge, forcing feasible paths to detour multiple times.

Table 9 shows that iLLM-A* achieves near-linear scalability for the map with one intermediate giant obstacle, with times scaling linearly with the map size, ranging from $0.49s$ to $1.15s$

Table 7: Few-shot Examples for In-context Learning.

Module II: Dynamic Few-shot Examples

Input:
source: [94, 321]
destination: [706, 668]
horizontal_barriers: [[494, 166, 634], [474, 57, 386]]
vertical_barriers: [[247, 182, 632], [553, 387, 775]]
Output:
Generated Path: [[94, 321], [217, 211], [341, 275], [464, 387], [588, 421], [650, 544], [706, 668]]
[10 In-context demonstrations abbreviated]

Table 8: Task-Specific Instructions and Input Template.

Module III: Task Instructions

Generate intermediate waypoints for the following input. If input data is ambiguous or constraint conditions contain logical conflicts, explicitly identify the problematic areas. Ensure the path generation is completely based on the provided input data. A path that perfectly follows all constraints will be considered a successful response.
Source Point: {source}
Destination: {Destination}
horizontal_barriers: {horizontal_barriers}
vertical_barriers: {vertical_barriers}
Generated Path:

across all map sizes. In contrast, LLM-A* cannot complete the planning tasks on the map with $N \geq 350$. Given $N = 300$, the runtime of LLM-A* is $1199.2\times$ longer than that of iLLM-A* ($1043.31s$ vs. $0.87s$). The path length consistently remains within 100% and 105.28%. The memory of iLLM-A* is also only about half of that of LLM-A*. The results for maps with long bar-shaped obstacles are similar and are illustrated in Table 10. For maps with $N = 300$, the runtime of LLM-A* is about $1000\times$ longer than that of iLLM-A*. The results are consistent with those in Table 1, which shows the robustness of iLLM-A* in diverse maps, even with unseen giant obstacles in the Few-shot examples.

Map Size (N)	Runtime (s) ↓		Memory (MB) ↓		Path Length (%) ↓	
	LLM-A*	iLLM-A*	LLM-A*	iLLM-A*	LLM-A*	iLLM-A*
50	0.54	**0.49**	0.14	**0.07**	102.20	100.00
100	4.05	**0.53**	0.44	**0.30**	107.22	103.75
150	19.35	**0.55**	2.74	**1.11**	107.33	103.83
200	145.92	**0.65**	3.18	**1.34**	105.85	103.80
250	375.51	**0.83**	3.87	**1.56**	104.74	105.28
300	1043.31	**0.87**	5.63	**2.45**	103.85	104.82
350	—	**0.94**	—	**3.23**	—	102.91
400	—	**1.03**	—	**3.28**	—	103.83
450	—	**1.15**	—	**3.37**	—	105.42

"—" indicates runtime exceeds 20 minutes.

Table 9: Performance for the map with one intermediate cross-shaped obstacle (see Figure 4(a)).

Map Size (N)	Runtime (s) ↓		Memory (MB) ↓		Path Length (%) ↓	
	LLM-A*	iLLM-A*	LLM-A*	iLLM-A*	LLM-A*	iLLM-A*
50	0.69	**0.55**	0.24	**0.15**	104.78	100.00
100	6.34	**0.57**	0.87	**0.56**	100.65	100.83
150	26.31	**0.57**	1.78	**1.05**	105.06	103.67
200	125.43	**0.64**	3.28	**1.12**	104.95	105.11
250	436.23	**0.69**	5.14	**2.11**	103.86	104.41
300	964.74	**0.81**	6.42	**2.08**	105.92	103.20
350	—	**0.87**	—	**2.38**	—	104.22
400	—	**0.92**	—	**2.64**	—	105.75
450	—	**1.15**	—	**2.89**	—	104.54

"—" indicates runtime exceeds 20 minutes.

Table 10: Performance for the maps with long bar-shaped obstacles (see Figure 4(b)).

G.2 Stability Analysis Details

To further verify the stability of iLLM-A*, Table 11 illustrates the path length Std in different sizes of maps. iLLM-A* achieves significantly lower Std across all map sizes compared to LLM-A*. For instance, at $N = 50$, iLLM-A* achieves a Std of 0.00, and at $N = 250$, the Std of iLLM-A* is only 0.46, whereas LLM-A* reaches 5.56. This comparison confirms the better consistency of iLLM-A*.

G.3 Visualization of Planned Paths on MovingAI Maps

Table 12 compares the pre-training time across different methods. Neural-A* and RLHA* require tens of hours of pre-training, whereas iLLM-A* operates without training.

Figure 5 first visualizes the planed path examples of iLLM-A* and A* across 6 different MovingAI map scenarios. The paths of iLLM-A* closely match the optimal paths of A*, demonstrating its near-optimal path. Slight differences appear in the Warcraft III and Real-world City/Street Maps scenario, where the planned path still remain close to the optimal ones in path length. Some LLM-generated waypoints do not lie directly on the final path due to the circle-based neighboring waypoint connection (Section 4.4).

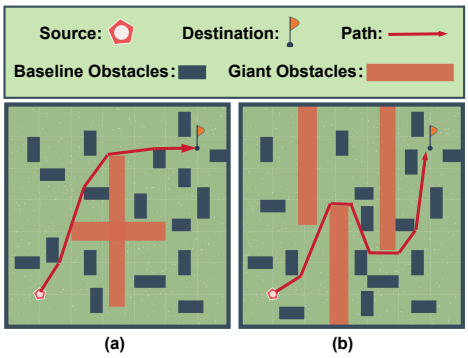

Figure 4: Basic Concept of The Gaint Obstacles: (a) Cross-Shaped Obstacles; (b) Long Bar-Shaped Obstacles.

Map Size (N)	Path Length Std (%) ↓ LLM-A*	iLLM-A*
50	0.63	**0.00**
100	1.67	**0.27**
150	3.06	**0.64**
200	4.97	**2.82**
250	5.56	**0.46**
300	3.46	**1.20**

Table 11: Path Length Std on Different Map Sizes.

G.4 Results on LLM Variants

In this section, we present the detailed evaluation results of iLLM-A* using different LLMs, as shown in Table 5 and 6. These comprehensive results further support the empirical findings discussed in the experimental sections. All experiments are conducted on the MovingAI maps with a grid size of $N = 512$. The few-shot prompts and search parameters across different LLMs are maintained identical to ensure that LLM is the only variable factor.

Table 13 shows the path length, runtime and memory of iLLM-A* using on different LLMs. On general, DeepSeek and Gemini achieve similar superio performance with Qwen. However, the three LLMs show somewhat different specific advantages in terms of different metrics. Specifically, DeepSeek and Gemini achieves a little bit longer runtime and similar memory, while a little bit shorter path length, than Qwen. The runtime of DeepSeek and Gemini is $1.6 - 7\times$ longer (worse), while the optimality gap of the path length of DeepSeek and Gemini is $1.2 - 13.1\times$ smaller (better) than that of Qwen. This is because DeepSeek and Gemini conduct is better at powerful reasoning on the maps, and thus generate higher quality waypoints. To summarize, iLLM-A* could achieve efficient path planning using different LLMs, while the specific advantages of different LLMs in reasoning may slightly impact the planning performance in terms of different metrics.

Map	PreTime (s) ↓ A*	JPS	iLLM-A*	Neural-A*	RLHA*
Baldur's Gate	0	0	0	72,356	43,436
Warcraft III	0	0	0	69,463	49,524
Dragon Age: Origins	0	0	0	84,368	53,564
StarCraft	0	0	0	101,345	47,325
Dragon Age 2	0	0	0	75,683	36,275
City/Street Maps	0	0	0	84,543	67,453

Table 12: PreTime comparison across different maps.

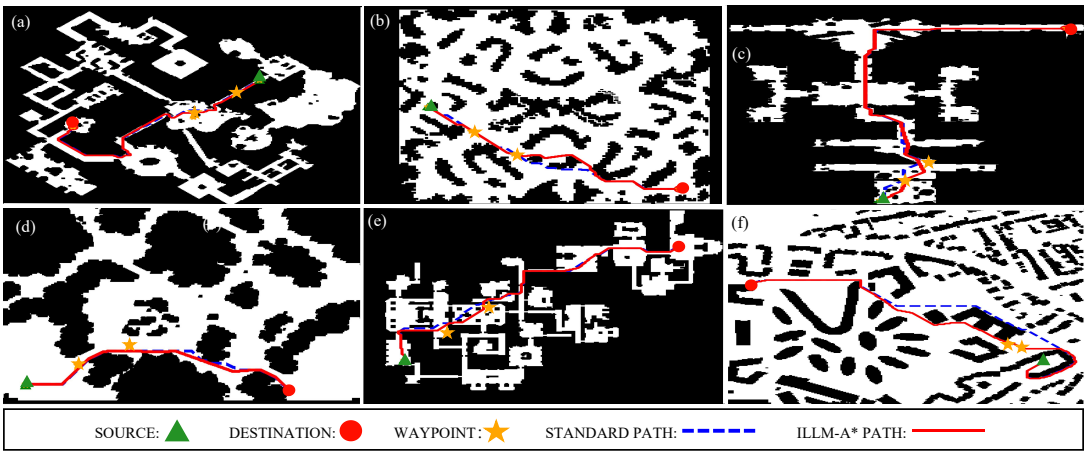

Figure 5: Visualization of Planed Path Examples of iLLM-A* on MovingAI Maps: (a) Baldur's Gate II, (b) Warcraft III, (c) Dragon Age: Origins, (d) StarCraft, (e) Dragon Age 2, (f) Real-world City/Street Maps.

Maps	**LLMs**	**Runtime (s)** ↓	**Memory (MB)** ↓	**Path Length (%)** ↓
Baldur's Gate	DeepSeek-V3.2-Thinking	4.33	**3.02**	**100.12**
	Gemini-2.5-Pro	3.14	3.74	101.33
	Qwen-TURBO	**0.89**	3.12	101.57
Dragon Age 2	DeepSeek-V3.2-Thinking	5.67	2.47	101.17
	Gemini-2.5-Pro	2.34	3.63	**100.97**
	Qwen-TURBO	**1.01**	**2.14**	102.03
Dragon Age: Origins	DeepSeek-V3.2-Thinking	4.37	3.42	102.71
	Gemini-2.5-Pro	3.31	4.01	**101.68**
	Qwen-TURBO	**0.61**	**3.25**	102.46
City/Street Maps	DeepSeek-V3.2-Thinking	5.69	4.07	**101.18**
	Gemini-2.5-Pro	3.35	4.21	102.54
	Qwen-TURBO	**1.03**	**3.46**	105.34
StarCraft	DeepSeek-V3.2-Thinking	3.06	**4.28**	**101.05**
	Gemini-2.5-Pro	2.41	4.92	102.20
	Qwen-TURBO	**1.54**	4.73	103.27
Warcraft III	DeepSeek-V3.2-Thinking	3.49	2.97	**101.13**
	Gemini-2.5-Pro	3.25	3.13	101.21
	Qwen-TURBO	**0.95**	**2.69**	103.32

Table 13: Performance comparison of different LLMs.

Algorithm 1 iLLM-A*

Require: Source s_0, destination s_g, OBSTACLE set obs, heuristic function $h()$, cost function $g()$, LLM $llm()$, Prompt template $Prompt$

OPEN list $\mathcal{O}_{open} \leftarrow \{s_0\}$, CLOSE list $\mathcal{C}_{close} \leftarrow \{\}$
TARGET list $\mathcal{T} \leftarrow$ experience_driven_selection($llm(s_0, s_g, obs, Prompt)$) // Mechanism I & II
TARGET state $t \leftarrow \mathcal{T}_{source}$, $g(s_0) \leftarrow 0$, $f(s_0) \leftarrow h(s_0)$, $Parent \leftarrow \{\}$
while $\mathcal{O}_{open} \neq \emptyset$ **do**
    $s_a \leftarrow \arg\min_{s \in \mathcal{O}_{open}} f(s)$
    if $s_a = s_g$ **then**
        break
    end if
    Remove s_a from $\mathcal{O}_{open}$, Add s_a to $\mathcal{C}_{close}$
    for all neighbors $s_n \in N[s_a]$ **do**
        if euclidean_distance(s_n, t) $< r$ and $s_g \neq t$ **then**
            $t \leftarrow T_{next}$, Partially update f-cost of states in $\mathcal{O}_{open}$ // Mechanism III
        end if
        if $s_n \in \mathcal{C}_{close}$ **then**
            continue
        end if
        $g_{tent} \leftarrow g(s_a) + cost(s_a, s_n)$
        if $s_n \notin \mathcal{O}_{open}$ or $g_{tent} < g(s_n)$ **then**
            $g(s_n) \leftarrow g_{tent}$, $Parent[s_n] \leftarrow s_a$
            $f(s_n) \leftarrow g(s_n) + h(s_n, s_g) + h(s_n, t)$
            if $s_n \notin \mathcal{O}_{open}$ **then**
                Add s_n to $\mathcal{O}_{open}$
            end if
        end if
    end for
end while
$\pi \leftarrow$ reconstruct_path(s_a)
$Prompt \leftarrow$ incremental_learning(π, $\mathcal{T}$, metrics)
return π