Constraint-Aware Route Recommendation from Natural Language via Hierarchical LLM Agents

Paper ID: 677

Abstract

Route recommendation aims to provide users with optimal travel plans that satisfy diverse and complex requirements. Classical routing algorithms (e.g., shortest-path and constraint-aware search) are efficient but assume structured inputs and fixed objectives, limiting adaptability to natural-language queries. Recent LLM-based approaches enhance flexibility but struggle with spatial reasoning and the joint modeling of route-level and POI-level preferences. To address these limitations, we propose RouteLLM, a hierarchical multi-agent framework that grounds natural-language intents into constraint-aware routes. It first parses user queries into structured intents including POIs, paths, and constraints. A manager agent then coordinates specialized sub-agents: a constraint agent that resolves and formally check constraints, a POI agent that retrieves and ranks candidate POIs, and a path refinement agent that refines routes via a routing engine with preference-conditioned costs. A final verifier agent ensures constraint satisfaction and produces the final route with an interpretable rationale. This design bridges linguistic flexibility and spatial structure, enabling reasoning over route feasibility and user preferences. Experiments show that our method reliably grounds textual preferences into constraint-aware routes, improving route quality and preference satisfaction over classical methods. Our code and data are publicly available[1].

1 Introduction

Route recommendation approaches(Zhang et al., 2024c) have become indispensable in modern daily life, spanning from high-level trip planning to detailed turn-by-turn navigation across diverse transportation scenarios. The rapid advancement of information technologies—including electronic devices, data storage, and 5G networks—has enabled

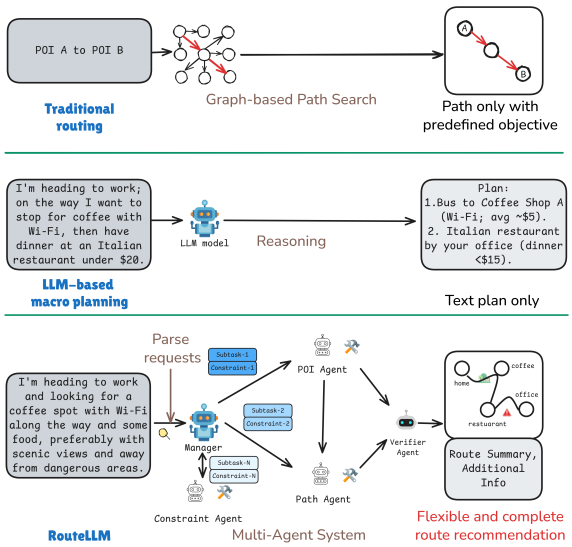


Figure 1: RouteLLM introduces a hierarchical multi-agent framework that bridges LLM reasoning with classical routing, enabling flexible, interpretable, and constraint-aware route recommendation.

the collection of vast spatiotemporal trajectory data reflecting daily commutes, travels, and activities (Zheng et al., 2009). Simultaneously, as people navigate a complex landscape of travel options, the growing demand for intelligent, personalized route recommendations has made the design of such systems a task of significant research and practical value, with applications spanning navigation, logistics, and comprehensive travel planning(Zheng, 2015; Liu et al., 2020; Xie et al., 2024).

Traditional route recommendation approaches have established a solid foundation for path-finding through various algorithmic paradigms. Classical search-based algorithms (Kleinberg, 2006) and heuristic search methods (Hart et al., 1968; Mandow and De La Cruz, 2008; Ulloa et al., 2020; Zhang et al., 2022, 2024b) have demonstrated effectiveness in identifying optimal paths based on predefined objectives such as shortest distance or minimal travel time. With advances in data storage and computing technologies, deep learning methods

[1] https://anonymous.4open.science/r/RouteLLM_ACL-DE52

(Jain et al., 2021; Almasan et al., 2022; Kong et al., 2024; Peng et al., 2021) have further enhanced route optimization by learning complex patterns from large-scale trajectory and network data. These approaches excel at providing precise, computationally efficient solutions for well-defined routing problems on graph structures. However, traditional methods face fundamental limitations in addressing modern users' diverse and evolving requirements. Most approaches operate within rigid frameworks optimizing for fixed objectives, lacking the flexibility to accommodate personalized preferences, contextual constraints, or complex multi-criteria decision-making. Furthermore, these systems require rigid, predefined parameter settings, making them inherently unsuitable for handling natural language queries or adapting to nuanced routing needs beyond standard optimization metrics.

Meanwhile, LLMs offer promising natural language understanding and reasoning capabilities for route recommendation, but current approaches reveal fundamental limitations across different paradigms. One category of methods attempts to directly input raw map data to LLMs for end-to-end route planning(Zhang et al., 2024d; Chen et al., 2024; Aghzal et al., 2023; Li et al., 2024). These approaches prove highly ineffective due to LLMs' inherently poor spatial understanding and inability to efficiently process large-scale geographical information, often resulting in spatially inconsistent or infeasible route suggestions. Another line of work focuses on enhancing traditional path search algorithms with LLM-guided optimization(Xiao and Wang, 2023; Gupta and Li, 2024a; Meng et al., 2024; Xiao and Yamasaki, 2025), but these methods remain constrained to fixed optimization objectives and fail to leverage LLMs' flexible reasoning strength for flexible user interaction. A third category treats route recommendation as a high-level planning problem, employing LLMs for text-based reasoning and itinerary generation(Xie et al., 2024; Zhang et al., 2024a; Wong et al., 2023). While these approaches successfully handle macro-level trip planning, they lack the spatial reasoning capabilities necessary for detailed route optimization and fail to bridge the gap between high-level planning and precise navigation. Consequently, existing LLM-based methods address only partial aspects of the route recommendation challenge, leaving a critical gap for systems that can seamlessly integrate objective flexibility with spatial precision.

To address the limitations of existing approaches, we propose RouteLLM, a hierarchical multi-agent framework that synergistically combines the precise path-finding of traditional algorithms with the flexible reasoning of LLM agents. Inspired by human planning, RouteLLM decomposes complex, multi-objective natural language requests into manageable sub-tasks with corresponding constraints. The framework employs specialized agents for parsing requests, coordinating the execution of sub-tasks, and synthesizing the results into a comprehensive route solution. This multi-agent design effectively addresses the core limitations of prior approaches by leveraging the complementary strengths of both technologies. By breaking down complex problems, the framework mitigates the LLM's poor spatial reasoning while maintaining the natural language flexibility users demand. The integration of traditional path-finding algorithms ensures spatial accuracy and efficiency, bridging the critical gap between natural language accessibility and spatial precision to provide a robust foundation for personalized route recommendation.

2 Related Work

2.1 Route Recommendation

Early route recommendation research was dominated by classical path-finding algorithms. Foundational methods such as graph search (Kleinberg and Tardos, 2006) and heuristic search (Hart et al., 1968; Mandow and De La Cruz, 2008; Ulloa et al., 2020; Zhang et al., 2022, 2024b) provided efficient solutions under single or multiple objectives, typically optimizing metrics like distance or travel time. With advances in data storage and computation, learning-based approaches emerged. Deep learning methods (Jain et al., 2021; Almasan et al., 2022; Kong et al., 2024; Peng et al., 2021) leverage large-scale trajectory and spatiotemporal data for more sophisticated optimization in logistics, delivery, and urban planning. Despite their efficiency, both classical and deep-learning methods remain rigid, offering limited support for personalized preferences or multi-criteria decision-making.

2.2 LLMs for Spatial and Route Reasoning

Large language models (LLMs) have been explored for their ability to interpret natural language queries and support flexible planning. Existing work falls into three categories, each with notable limitations: *End-to-End LLM Planners* attempt to process raw map data directly (Zhang et al., 2024d; Chen et al.,

2024; Aghzal et al., 2023; Li et al., 2024), but suffer from poor spatial comprehension and scalability issues, often producing infeasible routes. *LLM-Augmented Search* enhances traditional heuristics with LLM guidance (Xiao and Wang, 2023; Gupta and Li, 2024b; Meng et al., 2024; Xiao and Yamasaki, 2025), improving efficiency but remaining bound to fixed objectives and failing to exploit LLMs' full reasoning capabilities. *High-Level Trip Planning* applies LLMs to itinerary generation (Xie et al., 2024; Zhang et al., 2024a; Wong et al., 2023), enabling macro-level reasoning but lacking detailed spatial optimization. Bridging the gap between flexible natural language understanding and precise route optimization remains a central challenge.

3 Methodology

To bridge the critical gap between flexible, nuanced user language and the rigid, mathematical precision of route optimization algorithms, we designed RouteLLM, a multi-agent framework as illustrated in Figure 2. Our methodology systematically translates unstructured natural language into computable objectives through three key stages: (1) structured request parsing, (2) dependency-aware multi-agent coordination, and (3) adaptive, preference-weighted path planning.

3.1 Parsing and Decomposing User Intent into a Computable Route Plan

Human travel requests often blend multiple destinations with subjective preferences, such as scenery or specific amenities. Traditional route planners cannot interpret these nuanced requests because they rely on rigid A-to-B inputs. The first stage of our framework bridges this gap by parsing unstructured requests into computable objectives and constraints. This converts natural user language into structured data for optimization algorithms. To achieve this, our parsing approach decomposes requests using an [Object + Constraint] paradigm. Formally, the overall route planning task $\mathcal{T}$ is decomposed into interconnected sub-tasks: $\mathcal{T} = \{T_i \mid i \in S(K) = \{1, \ldots, K\}\}$ with corresponding constraints $\mathcal{C} = \{c_i(T_i) \leq 0 \mid i = 1, \ldots, K\}$. Crucially, this process can easily preserve the full spectrum of user intent, from explicit requirements to nuanced preferences, enabling downstream components to plan personalized and efficient routes.

Specifically, the Parser Agent extracts three primary categories of routing requirements: POI Requirements ($\mathcal{T}_{POI}$): Location-specific requests with filtering constraints (e.g., "visit a coffee shop with WiFi and quiet environment, then an Italian restaurant"). These requirements specify venue types with attribute-based filters, formalized as tuples $(type_i, \{attr_{i,j}\})$ where $type_i$ denotes the venue category and $\{attr_{i,j}\}$ represents the set of required attributes. Path Requirements ($\mathcal{T}_{path}$): Route-level preferences comprising two distinct types: **Specific constraints**: Well-defined binary restrictions (e.g., avoid toll roads, prefer highways) that can be directly encoded as hard constraints in the path search algorithm. **Abstract preferences:** Subjective requirements (e.g., scenic routes, energy efficiency, safety) requiring LLM-based interpretation to generate appropriate request-specific heuristic objectives as described in Section 3.3. Special Requirements ($\mathcal{T}_{special}$): Dynamic requirements that depend on real-time context or user-specific conditions not captured in the system's predefined ontology. Examples include weather-adaptive routing, live traffic event avoidance, and personalized accessibility accommodations. These requirements trigger auxiliary data retrieval and either (i) dynamically adjust constraints in both POI selection and path planning phases, or (ii) provide supplementary information without modification.

Among the three requirement categories, POI requirements translate directly into database queries, and path/special requirements can be encoded as hard constraints. Yet the core challenge lies in quantifying subjective Abstract preferences, which cannot be directly formalized. To solve this, the Parser Agent uses contextual prompting to extract preference weights. Rather than relying on supervised sentiment analysis, which requires extensive labeled datasets and may suffer from generalization issues, our approach adapts chain-of-thought sentiment reasoning(Fei et al., 2023) for preference extraction to: (1) identify abstract features mentioned in the input, (2) contextualize features with domain-specific knowledge, and (3) quantify preference intensity using a discrete scale. We formalize the extracted preferences as a vector $\mathbf{p} = (p_1, p_2, \ldots, p_m) \in \{0, 0.5, 1\}^m$, where m is the total number of possible abstract features (e.g., scenic, safe, energy-efficient), and each p_i represents the user's preference intensity for feature i. The discrete scale captures three levels of user intent: $p_i = 0$ indicates the feature was not mentioned and can be ignored; $p_i = 0.5$ reflects a moderate preference where compromise is accept-

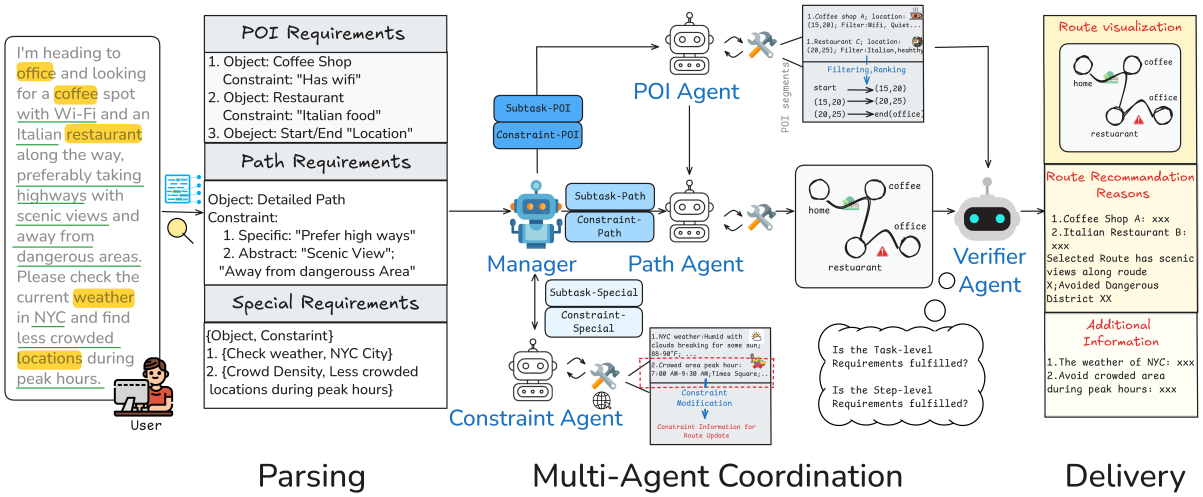


Figure 2: The operational workflow of the RouteLLM framework. A user's natural language request is first parsed into structured requirements. Then, a manager agent coordinates specialized agents (POI, Path, Constraint Agent) to resolve subtasks. Finally, the system delivers an optimized route with detailed reasoning.

able; and $p_i = 1$ represents a strong requirement treated as a hard constraint. This preference vector directly feeds into the multi-objective path planning algorithm (Section 3.3), where it guides the active objective decision and balance between competing objectives during route optimization.

3.2 Mitigating LLM Spatial Weaknesses via Multi-Agent Coordination

With the user's request structured into computable tasks, the next critical challenge is execution. Attempting to solve the entire plan with a single, monolithic LLM is ill-suited for this challenge for two primary reasons. First, LLMs exhibit a fundamental weakness in complex spatial reasoning, the very core of route planning. This deficit is often exacerbated by limited context windows, which can lead to forgotten constraints during a multi-step journey(Liu et al., 2023). Second, the problem itself is not uniform but is a composite of diverse subtasks. These range from structured database queries (for POIs) and algorithmic searches (for pathfinding) to dynamic requests requiring real-time context (like weather or traffic). Effective handling of such diversity requires a team of specialized agents, as a single, general-purpose LLM lacks both the specific tools and the contextual awareness needed for each distinct task. Therefore, to mitigate these intrinsic model weaknesses and effectively manage the problem's heterogeneous nature, our framework employs a multi-agent paradigm. This approach decomposes the single, large challenge into manageable sub-problems that can be systematically coordinated and assigned to specialized agents.

3.2.1 Hierarchical Planning and Decomposition

The cornerstone of this multi-agent paradigm is hierarchical planning and decomposition, orchestrated by the Management Agent. This stage converts parsed tasks into an executable plan by addressing two key challenges. First, it manages logical dependencies in route planning (e.g., selecting destinations before planning paths). Second, it decomposes operations strategically to mitigate LLM spatial reasoning limitations.

The planning process operates on two tiers to ensure logical coherence and robust execution. At the task level, the Management Agent identifies dependencies between sub-tasks. For example, Road block conditions must precede path optimization because routes depend on passable roads. Based on these dependencies, the agent determines an appropriate execution sequence. Beyond determining task order, the agent also decomposes individual tasks into finer-grained steps. At the step level, each task is decomposed into specific actions. For instance, "find a route" is broken into steps like "identify waypoints," "calculate distances," and "optimize path." This decomposition mitigates LLM spatial reasoning limitations by providing focused context for each action. Once the hierarchical plan is established, it is executed through a constraint-aware workflow. The Management Agent classifies constraints into two types: Local Constraints apply to individual sub-tasks (e.g., a

POI's attributes), while Global Constraints span the entire journey (e.g., total travel budget). This classification informs the execution strategy. Tasks without interdependencies can be processed in parallel for efficiency. For example, querying weather and searching for POIs can run concurrently. However, tasks with dependencies or those that modify global constraints must be coordinated sequentially to ensure route validity. This two-tier approach balances parallel efficiency with dependency management for task-specific agents.

3.2.2 Specialized Sub-Task Execution

Following the decomposition and planning by the Management Agent, the framework enters the execution stage. Here, the individual sub-tasks are dispatched to a team of specialized agents, each equipped with domain-specific tools and contextual knowledge to handle the local constraints ($\mathcal{C}_l$) of their assigned tasks through focused expertise. Our framework employs three primary types of specialized agents, corresponding to the sub-task categories identified during parsing:

POI Selection and Filtering. For POI Requirements ($\mathcal{T}_{\text{POI}}$), the POI Selection Agent is invoked. Its role is to translate abstract location requests, such as "a quiet coffee shop with WiFi," into concrete, route-feasible geographical points. To achieve this, the agent executes a three-step process: (1) it queries venue databases to retrieve candidates matching the required type (e.g., 'coffee shop'), (2) filters these results based on specific attribute constraints (e.g., 'has WiFi', 'quiet environment'), and (3) ranks the remaining venues using rating scores that incorporate user preferences.

Multi-Objective Path Optimization. For Path Requirements($\mathcal{T}_{path}$), the Path Optimization Agent is responsible for generating the actual travel segments $\mathbf{r}$ connecting the selected POIs. The agent considers both user preferences (e.g., scenic routes, avoid highways) and objective constraints (e.g., road closures, traffic restrictions) to produce feasible and optimized routes. We provide a more detailed treatment in Section 3.3.

Handling Dynamic and Special Constraints Finally, Special Requirements($\mathcal{T}_{special}$) are managed by the Constraint Agent. This agent's crucial role is to ground the route plan in the real-world's dynamic context by processing requests that depend on live information. To handle the diverse nature of these requests, the agent operates in a flexible dual-mode design: *Information Mode:* For requests that simply require supplementary data (e.g., current weather conditions or local event schedules), the agent retrieves the relevant information and passes it as additional relevant information to the Verifier Agent for inclusion in the final summary. This mode enriches the user's plan without altering the route's constraints. *Constraint Modification Mode:* For requests that must actively alter the route (e.g., "avoid areas with current flooding"), the agent retrieves real-time data, translates the conditions into new routing constraints, and propagates these updates to other relevant agents. This dual-mode architecture ensures that diverse special requirements are handled efficiently while maintaining a clear separation between the core routing logic and auxiliary information services.

Solution Synthesis and Presentation In the final stage of the framework, the verifier agent takes on the critical role of synthesizing the disparate outputs from all specialized agents into a single, cohesive, and user-friendly route recommendation. This process involves three key steps: integrating the partial results into coherent route plans, verifying these plans against global constraints, and finally, presenting the recommendations in an actionable, multi-modal format to users. Route Integration and Alternative Generation: The synthesis process begins by integrating the outputs from the specialized agents and creating complete plans.

Global Constraint Verification: Once one or more complete candidate solutions ($\mathbf{s} = \{POI_{set}, \mathbf{r}\}$) are assembled, the Verifier Agent performs a crucial verification step against all global constraints ($\mathcal{C}_g$) Unlike local constraints handled by individual agents, global constraints such as total travel time or cumulative budget require a holistic evaluation. The agent verifies each solution using the following function:

$$\mathcal{C}_g(\mathbf{s}) = \begin{cases} \text{feasible} & \text{if } g_j(\mathbf{s}) \leq \theta_j, \forall j \in \mathcal{G} \\ \text{infeasible} & \text{otherwise,} \end{cases} \quad (1)$$

where $g_j(\mathbf{s})$ is the value of the j-th global constraint, θ_j is its corresponding threshold and $\mathcal{G}$ is the set all global constraint. If a solution is infeasible, the agent applies a constraint relaxation strategy. Critical constraints (e.g., accessibility) may trigger a re-planning request to the Task-Specific Agents,

while soft constraints (e.g., a slight budget overrun) can be relaxed based on the user's preference vector, with a notification included in the final output. User-Oriented Presentation: After verifying the feasibility of the candidate routes, the final step is to transform the technical outputs into an intuitive and actionable format for the user. This is achieved through: *Natural Language Summarization:* Providing a concise summary of each route's key features (e.g., "This scenic route takes 15 minutes longer...") and detailed instructions. *Comparative Explanation:* Explicitly highlights differences between alternatives(e.g., "Option A is faster but misses the waterfront views..."). *Visual Representation:* Generates interactive maps and structured itineraries. This ensures users receive transparent, preference-aligned recommendations with clear justifications for routing decisions.·

3.3 Adaptive Multi-Objective Path Planning

Prior route recommendation methods rely on fixed, predefined objectives with rigid inputs. Our approach fundamentally reimagines this process by using LLMs as adaptive reasoning layers that translate human intentions into planning objectives, creating a dynamic bridge between natural language preferences and algorithmic optimization. We model the road network as a graph $G = (V, E)$. Each edge $e \in E$ is associated with m-dimensional non-negative cost vector $\mathbf{c}(e)$, representing objectives such as distance and scenic cost. For a given path $\pi = \langle v_0, \ldots, v_k \rangle$, its cumulative cost is the vector sum of the costs of its edges:

$$\mathbf{g}(\pi) = \sum_{i=0}^{k-1} \mathbf{c}(v_i, v_{i+1}) \,. \tag{2}$$

The LLM translates the user's abstract preferences into two key outputs: an active objective set $\mathcal{A} \subseteq \{1, \ldots, m\}$ and a corresponding weight vector w. This process tailors the search to the user's specific needs. For instance, an objective the user does not care about (e.g., 'scenic') is excluded from $\mathcal{A}$. A moderately important preference results in a reduced weight(e.g., $w_j = 0.5$) to the original cost landscape. while in the absence of specific preferences, the system defaults to a single-objective search($\mathcal{A} = distance$ using single-objective A*). With the search parameters thus defined by the LLM, the subsequent challenge is to find the optimal set of routes that balance the user's preferences. Since there is typically no single "best" route in a multi-objective context, the goal is to find an ϵ-approximate Pareto set using a suitable multi-objective search algorithm.(e.g., NAMOA*). This approach relies on the concept of ϵ-dominance to compare solutions. For any two solutions with cost vectors x and y, we say x ϵ-dominates y if:

$$\begin{aligned} x \preceq_\varepsilon y \Leftrightarrow & \forall j \in \mathcal{A} : x_j \leq (1+\varepsilon) y_j \\ & \text{and} \quad \exists k \in \mathcal{A} : x_k < (1+\varepsilon) y_k . \end{aligned} \tag{3}$$

This process yields the top k non-dominated solutions, which are then passed to the Delivery Agent. The agent provides a final natural language explanation of each route's unique trade-offs (e.g., "this route is faster but bypasses the scenic waterfront").

4 Experiment

We evaluate RouteLLM through experiments highlighting three aspects: (1) parsing natural language into structured tasks, (2) adapting to user preference changes while respecting hard constraints, and (3) transfer to a real-world case study. To support this, we construct a benchmark dataset that mirrors realistic routing scenarios and enables systematic testing across varying complexity levels.

5 Experiment

5.1 Experimental Setup

A significant challenge in this domain is the lack of a suitable benchmark for language-driven route planning. To address this, our experimental setup consists of two main parts. First, for systematic and controlled testing, we constructed a new benchmark inspired by (Xie et al., 2024), which includes synthetic maps and diverse queries. Second, to demonstrate real-world applicability, we conducted a separate case study using OpenStreetMap (OpenStreetMap contributors, 2025) data. The overall evaluation framework focuses on fine-grained route optimization and text-to-route bridging, using GPT-4o as the core agent LLM. Full details on the data construction process are provided in Appendix A.1.

Map Environments. We construct synthetic yet realistic map environments that reflect diverse urban topologies, including dense city grids, suburban tree-structured networks, scenic tourist areas with loop routes, and mixed urban–rural transitions. Each map is a 50×30 grid with 8-directional connectivity, enriched with POIs (e.g., restaurants, coffee shops, gym, and park) and edge attributes such as distance, traffic, toll status, and scenic value.

Table 1: Parsing Analysis. Best results in **bold**.
Struct = schema valid rate, POI = POI_F1, Const = Constraint_F1, Pref = Preference_F1

Method	POI	Const	Pref	Struct	F1
Direct	0.905	1	0.738	1	0.880
CoT	0.905	1	0.377	1	0.761
RouteLLM	**0.915**	**1**	**0.788**	**1**	**0.901**

This controlled setup ensures reproducibility while providing sufficient diversity for evaluation.

Query Generation. To evaluate the system across varying levels of complexity, we generated 100 natural language routing queries with GPT-4o. These queries were stratified by difficulty into three tiers: Simple (40%), Medium (40%), and Hard (20%). The difficulty level is defined by the complexity of the request, which includes the number of Points of Interest (POIs) to visit—up to 2 for Simple, 3 for Medium, and 5 for Hard—and between 2 to 5 requirements of basic to mixed types. Each query is annotated with a gold-standard parse that includes (1) required POIs and attributes, (2) path constraints and weighted preferences, and (3) special requirements such as time-sensitive or weather-dependent conditions. We ensured that every generated query has at least one feasible solution.

5.2 Parsing Reliability Analysis

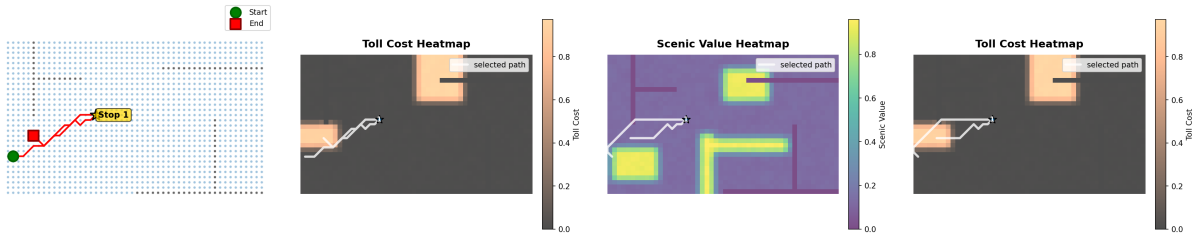
This experiment analyzes the parsing reliability of the agent. To assess our parsing approach, we compare RouteLLM's structured decomposition against two LLM prompting techniques: (1) **Direct**: which uses vanilla prompting without structured guidance, (2) **CoT** (Wei et al., 2022): which employs chain-of-thought reasoning. Our methods, (3) **RouteLLM**: utilize the [Object + Constraint] paradigm with CoT-guided reasoning and schema assurance via TrustCall. For this evaluation, each query in our dataset is annotated with a gold-standard parse. Performance is measured by the F1 scores for three key extraction tasks—POI requirements (POI), path constraints (Const), and user preferences (Pref)—as well as the schema validity rate (Struct). The overall F1 score is the average of the three component extraction scores. As shown in Table 1, all methods achieve perfect schema compliance (Struct = 1), ensuring structured outputs for downstream agents. However, baseline methods struggle with semantic extraction on diverse or ambiguous queries. Direct prompting achieves only 0.738 F1 on preference extraction, while CoT performs even worse at 0.377, suggesting that generic reasoning prompts may not adapt well to nuanced preference interpretation. Our RouteLLM parser addresses this through domain-specific contextual prompting, achieving 0.788 F1 on preferences and 0.901 overall F1.

5.3 Route Adaptation under User Preferences

This experiment demonstrates how changes in user preferences expressed in text are bridged to downstream route adaptations. We evaluate the system's ability to adapt routes based on varying user preferences. Starting with a baseline query (Scenario A), we modify the request to prioritize different objectives: scenic routes (B), safety (C), and shortest distance (D). Figure 3 and Table 2 present the results. *Baseline vs. Scenic Preference:* The baseline route (SP2 in Figure 3b) minimizes toll costs(1.70) but scores poorly on scenery (9.42). When the user prioritizes scenery (Scenario B), the system generates a new path (SP3 in Figure 3c) that navigates through high-value scenic areas (yellow regions in the heatmap). This adaptation improves the scenic cost from 9.42 to 4.22, but increases toll costs to 7.25 (SP4 in Figure 3d). This trade-off demonstrates the system's multi-objective balancing capability. *Other preference scenarios:* Scenario C (safer route) produces identical costs to the baseline, indicating the original route was already optimal for safety. Scenario D (shortest path) reduces distance cost from 35.46 to 32.97, with tolls increase to 2.60. These variations confirm the system responds appropriately to different preference weights. The results demonstrate two critical capabilities: (1) accurate translation of natural language preferences into quantifiable route objectives, and (2) transparent trade-offs between competing criteria. This traceability enables users to understand why RouteLLM made specific routing decisions, supporting interpretability and iterative refinement.

5.4 Case Study

This experiment validates the framework on real-world street networks. We deploy RouteLLM on New York City's Greenwich Village using real map and POI data from OSMnx (Boeing, 2025). Figure 4 shows routes generated from natural language queries (full queries in Appendix A.2). *Dynamic query adaptation.* The framework responds to both structural and preference-based query changes. Figure 4a and Figure 4b demonstrate basic adaptability: when the user modifies the starting location,

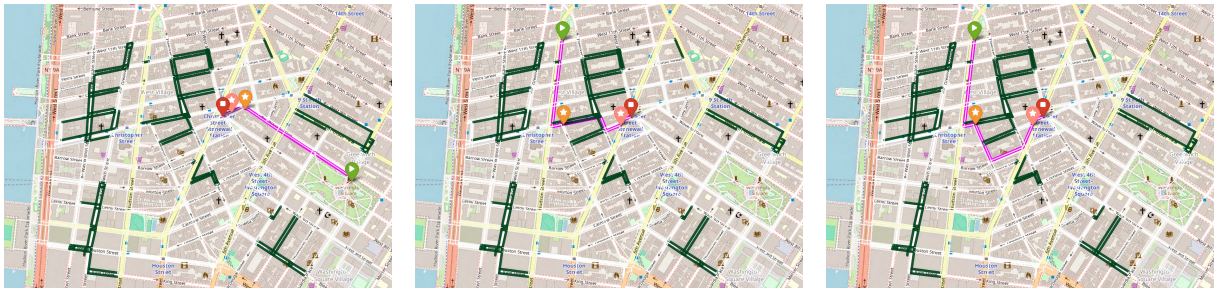


(a) SP1 (b) SP2 (c) SP3 (d) SP4

Figure 3: SP1: Path generated from the original; SP2: Same path overlaid on the toll cost heatmap; SP3: Path generated after modifying the query to prioritize scenic value, shown on the scenic value heatmap; SP4: Scenic-prioritized path overlaid on the toll cost heatmap.

Table 2: Comparison of different scenarios on multiple cost dimensions. Lower is better.

Scenario	cost_danger	cost_dist	cost_energy	cost_scenic	cost_slope	cost_toll
A_Baseline	7.54	35.46	3.45	9.42	3.59	1.70
B_More_Scenic	7.86	36.38	3.90	4.22	3.92	7.25
C_More_Safe	7.54	35.46	3.45	9.42	3.59	1.70
D_Shortest_Path	7.41	32.97	3.54	8.59	3.63	2.60



(a) R1 (b) R2 (c) R3

Figure 4: Case study on NYC Greenwich Village. Pink lines show generated routes, bubbles mark POIs and start/end points, green lines indicate high-scenic cost streets. (a) R1: Route after changing start point; (b) R2: Route prioritizing shorter paths; (c) R3: Route prioritizing scenic views.

the system regenerates the route accordingly. This responsiveness to concrete query edits is essential for interactive route planning. *Preference-driven adaptation:* Figure 4 demonstrates how preference changes affect routing decisions. When prioritizing shorter paths (R2), the system generates a direct route that passes through high-scenic (marked in green) cost streets. When emphasizing scenic views (R3), the route deviates to follow high-scenic streets despite increased distance. This contrast confirms the framework translates natural language preferences into measurable route trade-offs. The case study validates two aspects of real-world applicability. First, the system handles complex urban street networks with realistic POI distributions. Second, it adapts routes based on preference nuances expressed in natural language, not just binary constraints. These results suggest the framework generalizes beyond controlled environments to realistic deployment scenarios.

6 Conclusion

In this work, we introduced RouteLLM, a novel multi-agent framework that bridges natural language route requests with precise, constraint-aware route recommendations. Through extensive experiments, we demonstrated its ability to reliably parse complex queries, adapt to user preferences, and generalize to real-world scenarios. Our results highlight the strong potential of combining LLM-driven reasoning with classical path optimization to deliver flexible yet robust route planning. Future work will explore scaling to larger, real-world datasets and extending personalization capabilities.

Limitations

Simulated Dataset and Lack of Real-world Data. To our knowledge, no comparable benchmarks currently exist. While our simulated grid-based dataset can approximate real-world scenarios and the case study illustrates effectiveness, more complex and systematic real-world datasets would enable a more complete framework and more robust evaluation. Route recommendation in real-world contexts is inherently complex, and unified datasets are difficult to obtain due to diverse objectives and incomplete information. In future work, we aim to construct real-world datasets to address this gap.

Enhancements to the framework. Several components of our framework can be improved. For example, the Management Agent could be extended to handle more complex contexts and constraints, such as temporal dependencies, while the multi-agent architecture itself could be further strengthened. Future directions also include multimodal queries and outputs, as well as reinforcement learning for enhanced personalization and optimization. These aspects are beyond the scope of this work, but remain promising extensions.

Demonstration focus rather than strict comparison. Given the limitations of the current dataset, conducting strict, large-scale comparative analyses is challenging. Our experiments, therefore, emphasize a qualitative demonstration of RouteLLM's core capabilities rather than exhaustive performance benchmarking. We are also aware of several promising avenues for future improvement. These include enhancing the capabilities of the planning stage, improving the efficiency of large-scale tool invocation, incorporating online learning to better adapt to individual user preferences, and supporting multi-modal inputs. Advancing research in these directions will likely require the construction of more comprehensive, real-world datasets.

References

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

Paul Almasan, José Suárez-Varela, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2022. Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case. *Computer Communications*, 196:184–194.

Geoff Boeing. 2025. Modeling and analyzing urban networks and amenities with osmnx. *Geographical Analysis*.

Wanyi Chen, Meng-Wen Su, Nafisa Mehjabin, and Mary L Cummings. 2024. Can llms plan paths in the real world? *arXiv preprint arXiv:2411.17912*.

Hao Fei, Bobo Li, Qian Liu, Lidong Bing, Fei Li, and Tat-Seng Chua. 2023. Reasoning implicit sentiment with chain-of-thought prompting. *arXiv preprint arXiv:2305.11255*.

Devaansh Gupta and Boyang Li. 2024a. A training data recipe to accelerate a* search with language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 6681–6695, Miami, Florida, USA. Association for Computational Linguistics.

Devaansh Gupta and Boyang Li. 2024b. A training data recipe to accelerate a* search with language models. *arXiv preprint arXiv:2407.09985*.

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.

Jayant Jain, Vrittika Bagadia, Sahil Manchanda, and Sayan Ranu. 2021. Neuromlr: Robust & reliable route recommendation on road networks. *Advances in Neural Information Processing Systems*, 34:22070–22082.

J Kleinberg. 2006. Tardos. algorithm design. *Reading: Addison-Wesley*.

J Kleinberg and É Tardos. 2006. Algorithm design. alternative etext formats.

Yuntao Kong, Kun Yi, Lijun Wang, Cheng Peng, Le-Minh Nguyen, and Qiang Ma. 2024. Rpmtd: A route planning model with consideration of tourists' distribution. *IEEE Access*, 12:69488–69504.

Zhonghang Li, Lianghao Xia, Jiabin Tang, Yong Xu, Lei Shi, Long Xia, Dawei Yin, and Chao Huang. 2024. Urbangpt: Spatio-temporal large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5351–5362.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*.

Shan Liu, Hai Jiang, Shuiping Chen, Jing Ye, Renqing He, and Zhizhao Sun. 2020. Integrating dijkstra's algorithm into deep inverse reinforcement learning for food delivery route planning. *Transportation Research Part E: Logistics and Transportation Review*, 142:102070.

Lawrence Mandow and José Luis Pérez De La Cruz. 2008. Multiobjective a* search with consistent heuristics. *Journal of the ACM (JACM)*, 57(5):1–25.

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. *arXiv preprint arXiv:2407.02511*.

OpenStreetMap contributors. 2025. Openstreetmap. Accessed: 2025-10-02.

Ningyezi Peng, Yuliang Xi, Jinmeng Rao, Xiangyuan Ma, and Fu Ren. 2021. Urban multiple route planning model using dynamic programming in reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):8037–8047.

Carlos Hernández Ulloa, William Yeoh, Jorge A Baier, Han Zhang, Luis Suazo, and Sven Koenig. 2020. A simple and fast bi-objective search algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 143–151.

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.

IpKin Anthony Wong, Qi Lilith Lian, and Danni Sun. 2023. Autonomous travel decision-making: An early glimpse into chatgpt and generative ai. *Journal of Hospitality and Tourism Management*, 56:253–263.

Hengjia Xiao and Peng Wang. 2023. Llm a*: Human in the loop large language models enabled a* search for robotics. *arXiv preprint arXiv:2312.01797*.

Ling Xiao and Toshihiko Yamasaki. 2025. Llm-advisor: An llm benchmark for cost-efficient path planning across multiple terrains. *arXiv preprint arXiv:2503.01236*.

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. *arXiv preprint arXiv:2402.01622*.

Cong Zhang, Derrick Goh Xin Deik, Dexun Li, Hao Zhang, and Yong Liu. 2024a. Planning with multi-constraints via collaborative language agents. *arXiv preprint arXiv:2405.16510*.

Han Zhang, Oren Salzman, Ariel Felner, Carlos Hernández Ulloa, and Sven Koenig. 2024b. Aa* pex: Efficient anytime approximate multi-objective search. In *Proceedings of the International Symposium on Combinatorial Search*, volume 17, pages 179–187.

Han Zhang, Oren Salzman, TK Satish Kumar, Ariel Felner, Carlos Hernández Ulloa, and Sven Koenig. 2022. A* pex: efficient approximate multi-objective search on graphs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, pages 394–403.

Shiming Zhang, Zhipeng Luo, Li Yang, Fei Teng, and Tianrui Li. 2024c. A survey of route recommendations: Methods, applications, and opportunities. *Information Fusion*, 108:102413.

Zeyang Zhang, Xin Wang, Ziwei Zhang, Haoyang Li, Yijian Qin, and Wenwu Zhu. 2024d. Llm4dyg: can large language models solve spatial-temporal problems on dynamic graphs? In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4350–4361.

Yu Zheng. 2015. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):1–41.

Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. 2009. Mining interesting locations and travel sequences from gps trajectories. In *Proceedings of the 18th international conference on World wide web*, pages 791–800.

A Appendix

A.1 Map Environment Construction

Grid Map Generation Our benchmark employs a set of synthetic yet realistic grid maps designed to test the framework under diverse conditions. Each map is defined as a 50×30 grid with 8-directional connectivity. This scale was chosen to be computationally manageable while remaining complex enough to simulate a variety of real-world urban topologies. To facilitate multi-objective planning, each edge in the grid is assigned a vector of six cost attributes: distance, scenic value, energy consumption, danger level, slope, and toll cost. All attribute values are normalized to a range of [0,1], where a higher value uniformly represents a higher cost (e.g., lower scenic quality or greater danger). To better imitate real-world scenarios, where features like scenic views or dangerous areas are often spatially clustered rather than uniformly random, we developed a two-step assignment process. First, all edges are initialized with a low base cost, drawn randomly from a narrow distribution around 0.1. Second, we designate several polygonal "zones" of varying shapes on the map for each attribute. Edges within these zones are assigned a high cost, drawn from a distribution around 0.9. This method ensures that features exhibit spatial coherence, creating more realistic and challenging scenarios for the routing agent. Finally, to simulate urban obstacles, a small number of edges are designated as impassable. However, we ensure that the overall graph remains fully connected, guaranteeing that a feasible path exists between any two nodes.

Grid Map POI Generation To create a functional environment for routing queries, we populate the grid map with 50 Points of Interest (POIs). These are divided into four categories to simulate a variety of common destinations: 20 Restaurants, 15 Coffee Shops, 10 Gyms, and 5 Parks. Each POI is defined by more than just its coordinates and category. To support the complex, multi-constraint queries that RouteLLM is designed to handle, each category is enriched with a set of specific attributes. For instance, as detailed in Table 3, 'Restaurants' include fields for cuisine, rating, and is_vegetarian_friendly, while 'Coffee Shops' have a boolean is_work_friendly tag to indicate the presence of WiFi and a quiet environment. After defining this schema, we utilized GPT-4o to populate the 50 POI entries with diverse and plausible values for each attribute, such as generating unique restaurant names, ratings, and specific opening hours. Spatially, these 50 POIs are scattered across the grid in a roughly uniform random distribution, avoiding any specific clustering. This placement is a deliberate design choice to ensure the pathfinding environment is both challenging and realistic, creating a mix of short-distance (nearby POIs) and long-distance (far-apart POIs) planning scenarios.

Grid Map Query Generation The 100 natural language queries for our benchmark were generated with the assistance of GPT-4o. To achieve this, we provided the model with the context of our synthetic environment, including the map topology, POI locations, and the full schema of attributes. We then prompted GPT-4o to generate a diverse set of user routing queries that adhered to the difficulty stratification (40% Simple, 40% Medium, 20% Hard) outlined in the Experimental Setup. Following this automated generation, each query underwent a manual curation process. During this step, we refined the wording for clarity and realism and, critically, verified that at least one feasible solution existed for each query on its corresponding map. Queries that were ambiguous or found to be unsolvable were discarded and regenerated.

NYC Greenwich Village Case Study Map Generation For the case study, we constructed a map environment based on a real-world location: New York City's Greenwich Village. We used the OSMnx (Boeing, 2025) library to retrieve the pedestrian street network and relevant Points of Interest (POIs), such as restaurants and cafes, within a specific geographic bounding box(-74.0130, 40.7280, -73.9960, 40.7360).

A primary challenge when using real-world OSM data is its inherent complexity and incompleteness; many POIs lack the specific attributes (e.g., rating, average_cost) required for our multi-constraint queries. To create a usable and realistic dataset, we implemented a multi-step data preparation pipeline: *Standardization:* We first normalized raw data fields that were inconsistent. For example, diverse cuisine tags (e.g., 'pizza', 'sushi') were mapped to a standard set (e.g., 'Italian', 'Japanese'), and varied opening_hours strings were parsed into a consistent "HH:MM-HH:MM" format. *Attribute Enrichment:* To address missing data, we programmatically generated plausible values for critical attributes. For instance, missing rating and average_cost values were simulated by

drawing from normal distributions with category-specific parameters (e.g., cafes were assigned a higher average rating than fast-food restaurants). This ensures that every POI has a complete set of attributes for the agent to reason about. *Grid Abstraction:* Finally, to make the data compatible with our framework's underlying representation, the real-world longitude and latitude coordinates of each POI were mapped onto our 50×30 grid. This rigorous process resulted in a case study environment that is both grounded in a real-world location and fully compatible with our system's data requirements, enabling a meaningful evaluation of its applicability.

A.2 Queries for Case Study

The case study utilized a set of carefully designed natural language queries to test RouteLLM's adaptability in a real-world scenario. The queries were structured to evaluate the system's response to both concrete parameter changes and nuanced preference shifts. The key queries used to generate the results in Figure 4 are summarized below:

R1 Query: Hi there, can you plan a walking route for me this evening? I'm starting at the Washington Square Arch and need to end up at the Christopher Street-Sheridan Square subway station. First, I'd like to find a nice Italian restaurant for dinner. It should be pretty good-at least 4.0 stars-and have some good vegetarian options. My budget is around $80. After dinner, I'd like to grab a drink at either a quiet bar or a nice cafe before I head to the subway. For the route itself, I'm not in a huge rush, so I'd really prefer to walk along the more scenic and well-lit (safer) streets through the Village. Thanks!"

R2 Query: Hi there, can you plan a walking route for me this evening? I'm starting at the Chelsea Market and need to end up at the Christopher Street-Sheridan Square subway station. First, I'd like to find a nice Italian restaurant for dinner. It should be pretty good-at least 4.0 stars-and have some good vegetarian options. My budget is around $80. After dinner, I'd like to grab a drink at either a quiet bar or a nice cafe before I head to the subway. For the route itself, I'm not in a huge rush, so I'd really prefer to walk as short as possible, tho I want well-lit (safer) streets through the Village. Thanks!"

R3 Query: "Hi there, can you plan a walking route for me this evening? I'm starting at the Chelsea Market and need to end up at the Christopher Street-Sheridan Square subway station. First, I'd like to find a nice Italian restaurant for dinner. It should be pretty good-sat least 4.0 stars-and have some good vegetarian options. My budget is around $80. After dinner, I'd like to grab a drink at either a quiet bar or a nice cafe before I head to the subway. For the route itself, I'm not in a huge rush, so I'd really prefer to walk along the more scenic and well-lit (safer) streets through the Village. Thanks!"

A.3 No-Detour Spatial Filter

To improve practical application, we designed an optional POI filtering strategy to avoid unnecessary detours. Although some POIs might fit the user's requirements, they can introduce inefficient travel. Crucially, to ensure the final route is logical and efficient, the agent employs a no-detour filtering mechanism. This spatial heuristic evaluates the arrangement of candidate POIs and eliminates orderings that create an inefficient path—for example, preventing a sequence like A→C→B when the optimal geographic progression is A→B→C. We find this specialized process of structured querying and spatial filtering is more reliable than asking a generalized LLM to reason about locations directly on a map.

A.4 Prompts for Parsing Reliability Analysis

Direct Prompting

You are a travel planning assistant. Your task is to extract the user's requirements plan into a single, valid JSON object that strictly follows the provided schema format.
[The Common Definitions for POIs, Preferences, and JSON Structure were inserted here.]
Do not add explanations or markdown.

Chain-of-Thought Prompting

You are an expert JSON parsing assistant. First, reason through the user's request by following these steps in your head:
1. Identify POI requirements using [Object, Constraints] paradigm.
2. Extract route preferences.
3. Note any hard limits.
[The Common Definitions for POIs, Preferences, and JSON Structure were inserted here.]
[Two detailed examples of user requests and the corresponding correct JSON output were provided here.]
After your reasoning is complete, output ONLY the raw JSON object. Do not include your reasoning steps. Do not add explanations or markdown.

RouteLLM Prompting

You are a highly precise and structured data extraction agent. Your sole purpose is to convert a user's travel request into a single, valid JSON object.
$< CONTEXT >$
[Context description was provided here.]
$< /CONTEXT >$

$< SCHEMA_AND_RULES >$
[The target JSON schema and formatting rules were provided here.]
$< /SCHEMA_AND_RULES >$

$< INSTRUCTIONS >$
Step 1: Parse Route Preferences (soft_prefs)
[Rules for assigning preference weights were provided here.]
Step 2: Parse POI Stops (poi_stops)
[Detailed instructions for handling sequential (AND) vs. optional (OR) logic were provided here.]
Step 3: Apply POI Filters
$< POI_SPECIFICATION >$
[The detailed POI attribute schema was provided here.]
$< /POI_SPECIFICATION >$
$< /INSTRUCTIONS >$

$< FEW_SHOT_EXAMPLES >$
[Two detailed examples of user requests and the corresponding correct JSON output were provided here.]
$< /FEW_SHOT_EXAMPLES >$

Table 3: Schema of POI Attributes by Category

Category	Field	Description / Possible Values
Restaurant	`coordinate (x,y)`	Grid coordinate of the POI (x ∈ [0,49], y ∈ [0,29])
	`category`	"restaurant"
	`cuisine`	Cuisine type – one of ["Chinese", "American", "Italian", "Mexican", "Indian", "Mediterranean", "French"]
	`rating`	Customer rating, float between 0.0 and 5.0 (one decimal place)
	`average_cost`	Approx. average cost for a meal (integer, in currency units)
	`is_vegetarian_friendly`	Boolean indicating if vegetarian options are available (`true / false`)
	`opening_hours`	Opening hours as a time range string in "HH:MM-HH:MM" (24-hour format)
Coffee Shop	`coordinate (x,y)`	Grid coordinate of the POI (x ∈ [0,49], y ∈ [0,29])
	`category`	"coffee_shop"
	`is_work_friendly`	Boolean indicating if it's work-friendly (e.g., has WiFi, quiet)
	`average_cost`	Typical cost of a coffee/snack (integer in currency units)
	`rating`	Customer rating, float between 0.0 and 5.0 (one decimal place)
	`opening_hours`	Opening hours as a time range string in "HH:MM-HH:MM" (24-hour format)
Gym	`coordinate (x,y)`	Grid coordinate of the POI (x ∈ [0,49], y ∈ [0,29])
	`category`	"gym"
	`rating`	Customer rating, float between 0.0 and 5.0 (one decimal place)
	`average_cost`	Cost of membership or drop-in (integer in currency units)
	`has_swimming_pool`	Boolean indicating if a swimming pool is available (`true / false`)
	`opening_hours`	Operating hours, format "HH:MM-HH:MM" (e.g., "06:00-22:00")
Park	`coordinate (x,y)`	Grid coordinate of the POI (x ∈ [0,49], y ∈ [0,29])
	`category`	"park"
	`has_entry_fee`	Boolean indicating if the park charges an entry fee
	`rating`	Visitor rating, float between 0.0 and 5.0 (one decimal place)
	`opening_hours`	Open hours, format "HH:MM-HH:MM" (e.g., "05:30-19:30")