Improving Spatial and Temporal Awareness of Large Language Models for Personalized Travel Planning

Anonymous ACL submission

Abstract

Travel itinerary planning requires coordinating multiple tasks, like flight and hotel bookings, while meeting spatial and temporal constraints. Although Large Language Models (LLMs) show promise in tackling complex planning tasks, they often struggle with maintaining geographical consistency, managing real-time travel constraints, and providing accurate logistical details. We present TRAVLINKTO, a system designed to enhance the spatial and temporal awareness of LLMs for travel itinerary planning. TRAVLINKTO ensures geographical coherence by leveraging city graphs to optimize routes across cities. It integrates LLM-generated content with real-time data from external sources, such as flight schedules and restaurant booking systems, through an iterative self-refinement process that utilizes automated query generation, execution, and parameterized planning templates. We evaluate TRAVLINKTO on the TravelPlanner dataset. Experimental results show that TRAVLINKTO significantly outperforms existing LLM-based methods, enhancing both the quality and efficiency of travel planning.

1 Introduction

Travel itinerary planning systems aim to automate the creation of personalised travel plans. Given a user query — such as "three-day trip to London with family" — these systems generate detailed itineraries that include attractions, restaurants, accommodations, and transportation arrangements. An effective planning system should understand the user's query to generate itineraries that satisfy spatial (e.g., locations) and temporal (e.g., time) constraints, as well as other requirements such as budget and preferences. Additionally, it should adapt to real-world changes (e.g., flight schedule adjustments) or unexpected disruptions to ensure a valid travel plan.

Unlike traditional approaches that rely on predefined rules or structured data , LLMs (Guo et al., 2024) have opened up new opportunities for automated travel planning (Xie et al., 2024; Valmeekam et al., 2023). LLMs can understand complex natural language queries and user preferences, enabling more intuitive travel planning interactions. With carefully designed instruction strategies (Schulhoff et al., 2024), LLMs can function as domain experts to generate precise and personalised travel suggestions (Guo et al., 2024). Furthermore, by integrating with external databases and APIs (Wang et al., 2024b; Li et al., 2023), LLMs can access real-time travel information and dynamically adjust plans based on current conditions, overcoming the limitations of recommendation-based systems (Valliyammai et al., 2017; Lim et al., 2015) and traditional deep learning approaches (Chen et al., 2023).

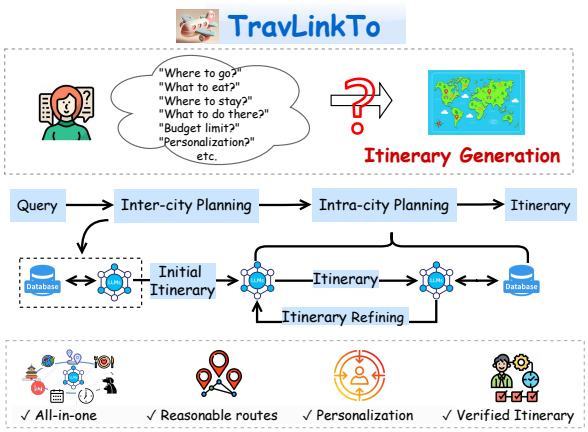


Figure 1: Overview of TRAVLINKTO, a hierarchical framework for automated travel planning. The system first conducts inter-city planning followed by intra-city activity scheduling, leveraging both city graph structure and external databases to ensure reasonable routes and personalised recommendations.

However, LLMs still face challenges in travel planning tasks. **First**, despite their sophisticated language understanding, these models struggle with spatial reasoning and geographic relationships, which are fundamental to creating feasible

itineraries (Bhandari et al., 2023). For example, they may schedule two far-apart attractions in an unreasonable timeframe without considering the geographic distances between them. **Second**, even with access to external APIs or databases, LLMs are still influenced by their outdated training data. This reliance on historical information often leads to the generation of inconsistent or hallucinated content, such as recommending non-existent flights, closed attractions, or relocated venues (Huang et al., 2024). **Third**, LLMs struggle to manage multiple constraints simultaneously, such as finding hotels that meet budget and location preferences while aligning with transportation schedules and time constraints (Chen et al., 2024; Wang et al., 2024a).

We present TRAVLINKTO, a novel hierarchical framework designed to address the aforementioned limitations by enhancing temporal and spatial awareness of LLMs for travel itinerary planning. An overview of TRAVLINKTO is depicted in Figure 1. To enable spatial reasoning, TRAVLINKTO models inter-city routes as a city-level graph, ensuring LLMs itineraries respect geographic connectivity and distances. Such a structured representation empowers LLMs to generate logically consistent, geographically feasible itineraries. To enhance reliability, TRAVLINKTO uses a multi-agent framework to retrieve external knowledge and iteratively refine itineraries via templates. Cross-referencing content with external knowledge and providing refinement feedback reduces hallucinations and ensures adherence to temporal, budgetary, and user constraints. Integrating graph-based spatial reasoning with iterative refinement enables TRAVLINKTO to effectively adapt LLMs to complex travel scenarios, delivering coherent, efficient, and personalised itineraries.

We evaluate TRAVLINKTO on the TravelPlanner dataset (Xie et al., 2024), comparing it with ReAct (Yao et al., 2023) using multiple LLMs (GPT-3.5-Turbo, GPT-4-Turbo, and Gemini Pro) (Team et al., 2023; OpenAI, 2024, 2022). Experimental results show that TRAVLINKTO significantly improves the reasoning capabilities of LLMs in travel planning tasks in terms of constraint satisfaction, planning efficiency, and adaptability to complex scenarios.

This paper makes the following contributions:

- A new approach to integrate knowledge graphs and LLMs for location-aware, long-duration travel planning;
- An iterative self-refinment mechanism for mitigating LLM hallucination;
- An efficient tool-based constraint handling system for improving planning efficiency and user satisfaction.

Table 1: Comparison of different approaches in travel itinerary planning systems

Type	Method	Capabilities			
		Parse	Match	Plan	Adapt
Mathematical	Integer linear program	None	None	Weak	Weak
	Ant colony optimization	None	None	Medium	Medium
Recommendation	Collaborative filtering	Weak	Medium	None	Weak
	Matrix factorization	Weak	Medium	None	Weak
DL-based	knowledge graph	Weak	Strong	Strong	Strong
	Reinforce Learning	Weak	Medium	Strong	Medium
LLM-based	Standalone LLMs	Strong	Medium	Weak	None
	LLMs with prompting	Strong	Medium	Weak	None
	LLMs with RAG	Strong	Medium	Weak	Strong
	Tools-based LLMs	Strong	Medium	Medium	Strong
Hierarchical	**TRAVLINKTO**	Strong	Strong	Strong	Strong

Support level: None < Weak < Medium < Strong

Code availability. The code associated with this work is available at: https://anonymous.4open.science/r/TRAVELINKTO-B728/

2 Related Work

Table 1 positions TRAVLINKTO with respect to prior travel planning systems.

Early optimisation methods (e.g., integer linear programming (Sylejmani et al., 2024), ant colony optimisation (Dorigo et al., 2006)) focused on mathematical optimisation of travel routes. Deep learning later enabled more personalised approaches, from recommendation-based methods (collaborative filtering (Schafer et al., 2007), matrix factorization (Xu et al., 2017)) for POI suggestions to advanced techniques like knowledge graph (Hogan et al., 2021) and reinforcement learning (Zheng et al., 2024) for capturing complex travel patterns.

Recent advances in LLMs have created new possibilities for travel planning through natural language understanding and contextual reasoning. To enhance base LLM capabilities, approaches like prompt engineering (Liu et al., 2023), Retrieval Augmented Generation (RAG) (Edge et al., 2024), and tool integration (Yang et al., 2023), multi-agent collaborate (Guo et al., 2024) have emerged to combine language understanding with real-time travel data and specialized reasoning.

LLM-based travel planning systems encounter key challenges. Notably, their spatial reasoning is limited; for instance, a Seattle to Los Angeles trip might incorrectly route through Chicago, highlighting a lack of geographical awareness. Furthermore, these planners struggle to maintain factual accuracy

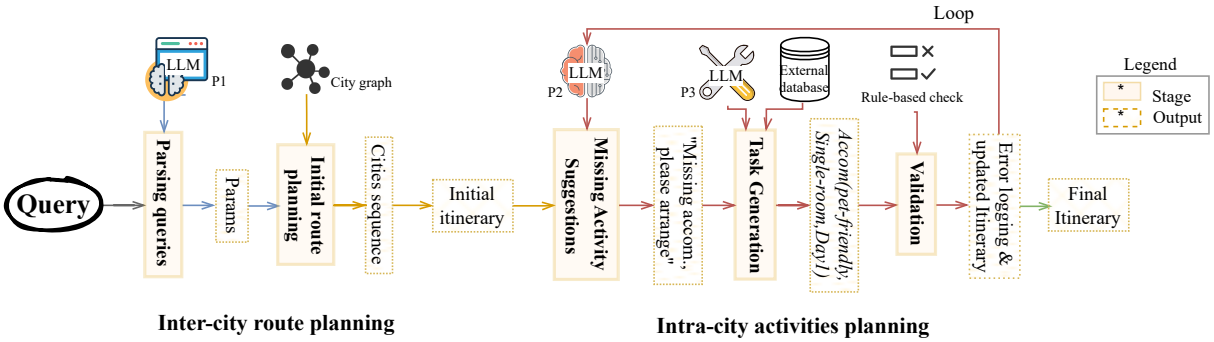


Figure 2: TRAVLINKTO uses prompt engineering to instruct a common LLM to play different roles in the processing pipeline. It combines LLMs and external tools to improve the quality of the generated plan.

in multi-turn conversations with external databases, drifting from facts due to their stochastic nature. Finally, managing simultaneous travel constraints leads to suboptimal solutions in budget, time, and preferences.

To mitigate these limitations, TRAVLINKTO augments LLM capabilities by incorporating graph-based spatial reasoning and interesting LLMs with external tools within an iterative self-refinement mechanism to improve the temporal awareness and reliability of LLM-based travel planning.

3 Our Approach

3.1 Overview

Figure 2 depicts the workflow of TRAVLINKTO for planning inter-city travel (e.g., city sequences) and intra-city activities (e.g., hotels, lunches, and attractions).

TRAVLINKTO employs a two-stage processing pipeline. The first stage establishes inter-city routes using a city-level graph, ensuring geographical and logical coherence. The second stage refines intra-city activity scheduling through an iterative refinement process. This process integrates LLMs and external tools to instantiate daily activities such as hotels and attractions using a customizable and parameterized template. To construct a comprehensive itinerary, TRAVLINKTO retrieves information (e.g., hotel availability and flight schedules) from external databases. It leverages few-shot prompt engineering to map user queries to predefined functions, which are then interpreted and executed to fetch relevant data. Additionally, TRAVLINKTO incorporates a self-refinement mechanism to minimize errors across the processing pipeline, ensuring that the generated itineraries adhere to temporal, budgetary, and user-defined constraints.

Table 2 presents the key components of TRAVLINKTO with simplified example LLM prompts used by TRAVLINKTO. A complete list of prompts, templates, APIs and examples is given in the Appendix. As a working example, consider responding to a user query for planning "*A 3-day trip with travel from Seattle to Los Angeles, including a pet-friendly room from May 20th, 2025*". After parsing the user query, TRAVLINKTO iteratively executes the processing pipeline in Figure 2, outlined as follows.

Table 2: Overview of System Components in TRAVLINKTO

	Process (Module)	Description (Method | Purpose | Example)
Prompts	Parsing Queries	Few-shot | Parameter extraction | "Extract travel parameters..." (P1).
	Activity Planning	Zero-shot | Itinerary review | "Verify completeness..." (P2). Few-shot | Function calling | "Identify missing items..." (P3).
Tools	Transportation	Transit options and trip details.
	Attractions	Attraction route planning and visitor information.
	Accommodations	Lodging options and property details.
	Restaurants	Dining recommendations and venue information.
	City Network	City connectivity and travel mode.
	Attraction Network	Attraction connectivity and visit duration.
Utils	Code Interpreter	Python function execution and processing.
	Validator	Input validation and schema verification.

Detailed implementations are provided in Appendix 7.2

Parsing queries. We use an LLM API (e.g., ChatGPT) to map the query into structured parameters, such as the origin, destination, duration, preferences, and constraints, to be stored in JSON format. This is achieved by using an example-based few-shot prompt (P1 in Table 2) to leverage LLMs for natural language understanding and structured data extraction.

Inter-city route planning. Given a JSON document of user query parameters, TRAVLINKTO employs a simple yet effective sampling algorithm to construct initial inter-city routes. It selects routes from a city graph database to ensure the geographical validity of the travel sequence. For example, the initial itinerary for our working example could be: "*Day 1: Travel from Seattle to Los Angeles, Day*

2: Explore Los Angeles, Day 3: Return to Seattle." Section 3.3 details this planning process.

Intra-city activity scheduling. With the initial itinerary, TRAVLINKTO populates a daily activity template (Figure 14 in the Appendix) by iteratively refining the plan for each day. In each iteration, it presents the partially completed itinerary to the LLM, prompting it to suggest actions for missing elements (P2 in Table 2). The LLM feedback is then used to generate a structured JSON query with predefined functions via few-shot prompting (P3 in Table 2). The *Code Interpreter* of TRAVLINKTO in Table 2 then interprets and executes this query to retrieve relevant information from external databases, such as restaurant and hotel booking systems.

Validation. At the end of each inter-city planning iteration, TRAVLINKTO validates and refines the generated information. It checks consistency with user preferences and ensures spatial alignment with the planned route. Discrepancies are logged and fed back into the pipeline for adjustment in the next iteration (refinement loop in Figure 2). This iterative refinement process continues until a complete and valid itinerary is generated or a maximum of 30 iterations is reached. If the limit is reached, TRAVLINKTO resets to the query parsing stage, asks the LLM to regenerate user parameters, and retries the pipeline, with a maximum of retries (10 in this work).

3.2 Parsing Queries

TRAVLINKTO extracts travel requirements and user preferences via a few-shot prompt, "*Extract and structure the following travel parameters from the user query, returning the results in JSON format. Output sample:* `{"travel_days": 2, "origin": "Washington", ... }`" to extract key travel parameters stored in a structured JSON format. Here, we want to extract two distinct parameter categories: *Route Parameters* (origin, destination, stops, duration) for overall route planning, and *User-specific Parameters* (e.g., preferences for cuisine, accommodation, activities) for detailed arrangements at each destination.

3.3 Inter-City Route Planning

If the extracted user parameters include multiple cities, TRAVLINKTO employs a route-planning algorithm to determine the optimal sequence of visits using city-scale graphs. In this work, we integrate TRAVLINKTO with two directed graphs: a city-level graph and an attraction-level graph, though this approach can be extended to incorporate other graphs to capture additional data dependencies.

The city-level graph $G = (V, E)$ represents cities as nodes V (including attributes such as name, region, and description) and transport links as edges E (characterized by travel mode, time, and cost). Meanwhile, the attraction-level graph $G' = (V', E')$ models attractions as nodes V' (with properties like name, type, and description), while edges E' encode direct connections based on distance and popularity.

For inter-city route planning, we construct a feasible travel route given a start city, an end city, and intermediate stops, using a degree-based sampling approach. First, we identify the set of reachable cities R_o that connect the origin city o to the target destination. From this set, we select two anchor cities, c_1 and c_n, while the top $n-2$ most connected cities serve as intermediate stops, forming the complete city set $C = \{c_1, c_2, \ldots, c_n\}$. The final route is expressed as $P = (p_0, p_1, \ldots, p_{n+1})$, where $p_0 = p_{n+1} = o$ and $p_1, p_2, \ldots, p_n = C$. This method ensures that the selected cities are well-connected, maximizing feasibility while allowing customization based on user-defined weights or preferences.

TRAVLINKTO implements a retry mechanism in case no valid route is found during one sample iteration. In this case, a new city set is generated, and the algorithm is retried. After 15 unsuccessful attempts, an error is reported, prompting a return to the query-parsing stage, where the LLM is asked to refine the extracted user parameters.

3.4 Intra-city Activity Planning

Given an initial city visit plan (e.g., *"Day 1: Seattle to Los Angeles, Day 2: Los Angeles, Day 3: Return to Seattle"*), TRAVLINKTO prompts an LLM to generate structured queries for external databases to instantiate a customizable, predefined daily template for daily activities like restaurant reservations and accommodation bookings.

3.4.1 Template-guided itinerary sketch

Using the daily activity template as a guideline, TRAVLINKTO first expands the city sequence into a skeleton itinerary, structured in JSON format with placeholders (i.e., `todo`) for missing components. For example, a skeleton itinerary for our working example would be: *"Day 1": "cities": ["Seattle", "Los Angeles"], "transportation": "todo", "accom-*

modation": "todo", "attractions": "todo", "restaurants": "todo", [...]" The skeleton itinerary also incorporates user preferences and constraints extracted from the user query. These may include preferences such as pet-friendly accommodation and dietary requirements, as well as constraints like budget limits or travel duration. We note that TRAVLINKTO automatically omits irrelevant elements from the template. For instance, if the user does not specify a hotel preference, the itinerary will exclude a placeholder for accommodation constraints.

3.4.2 Identifying missing activities

After generating the skeleton itinerary, TRAVLINKTO queries an LLM to generate actions to gather data from external databases to populate the placeholders. For this task, we utilize a zero-shot prompt, "*Identify missing items and execution errors. Provide actionable feedback for both incomplete elements and error solutions.*" (P3 in Table 2). The prompt instructs the LLM to detect gaps and potential issues in the itinerary. For instance, if the LLM detects the absence of accommodation details (e.g., a todo in the hotel element of day 2), it will generate an LLM prompt such as, "*Accommodation arrangement needed for city X,*" to be used within the iterative planning loop. Essentially, we guide the LLM using chain-of-thought-like reasoning to improve the generated plan in our processing pipeline.

3.4.3 Data retrieval

Using the LLM-generated prompts from the previous step, TRAVLINKTO appends examples to instruct the LLM to generate structured JSON queries (Table 2). For instance, if the LLM prompt indicates missing accommodation details, TRAVLINKTO provides a list of relevant functions and their descriptions related to accommodation booking. It then feeds the example prompt to the LLM, guiding it in constructing a JSON query. For our working example, the LLM may generate the following response: *"function": "filter_db", "argument": "DBName": "accommodations", "city": "Los Angeles", "room_type": "pet-friendly"*. This JSON query is then parsed by the TRAVLINKTO Code Interpreter, which executes the request by searching the accommodation database for available pet-friendly rooms in Los Angeles using the "`filter_db`" function. TRAVLINKTO provides a Python API to support the integration of external databases, including transportation, attractions, restaurants, and accommodations. Additionally, it offers built-in query functions such as `filter_db` for database filtering and `cities_confirm` for route verification, which can also be easily extended and customized using its API.

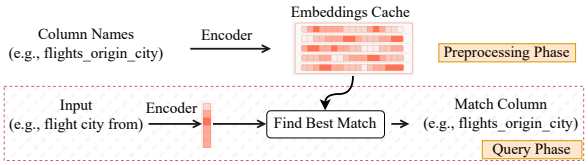


Figure 3: Column name matching process

As LLM-generated JSON queries are not guaranteed to be syntactically or semantically valid, we need to take extra steps to correct the errors. These are described in the following paragraphs.

JSON query sanitization. To correct syntax errors and parameter inconsistencies (e.g., `flights_source_city` vs. `departure_city`), TRAVLINKTO validates and sanitizes JSON queries using a built-in script. It removes invalid characters, ensures proper brace matching, and enforces predefined type specifications, including string length, character restrictions, numerical ranges, and date formats (`MM-DD-YYYY`).

Function name mapping. To ensure JSON queries contain the correct database API calls, TRAVLINKTO uses string similarity computation to map the generated `function` and `DBName` to predefined system functions. Specifically, it applies the Levenshtein distance (Miller et al., 2009), which measures the minimum single-character edits needed to transform one string into another. This allows mismatched function names (e.g., "data.filter") to be corrected to predefined system functions (e.g., `filter_db`).

Semantic correction. For semantically similar keys with low lexical overlap (e.g., "departure city" and "source location"), TRAVLINKTO uses text embeddings for semantic alignment of argument keys and database columns. Column names, contextualized with their database (e.g., `flights_source_city`), are encoded using the OpenAI `text-embedding-3-small` embedding model (OpenAI, 2022) and cached to match with the embedding of the input. We then select the best-matching column via cosine similarity with cached embeddings, as shown in Figure 3.

Retrieval information and error logging. Finally, the TRAVLINKTO Code Interpreter executes the generated query against the relevant

database (e.g., "`accommodations`" with parameters `"city": "Los Angeles", "room_type": "pet-friendly"`). A successful execution retrieves the required information (e.g., pet-friendly accommodations in Los Angeles) and updates the skeleton itinerary with the retrieved data.

During execution, two scenarios may arise. If multiple matching records exist, TRAVLINKTO selects one according to the user preference or the default setting (e.g., choosing the lowest-cost option). If no matches are found, it logs an error and returns to the query parsing stage to reprocess user parameters. To prevent duplicates, TRAVLINKTO maintains a record of previous results and filters out repeated entries before making a selection.

3.4.4 Validation and optimization

During intra-city planning iteration, TRAVLINKTO validates the current itinerary to ensure it meets all constraints extracted from the user query, focusing on spatial coherence and temporal consistency.

For spatial coherence, TRAVLINKTO verifies that retrieved information aligns with the planned city route. Any misaligned items are logged to re-execute the relevant processing pipeline. For temporal consistency, it checks whether hotel minimum stay requirements match the planned durations in each city. If conflicts arise, TRAVLINKTO resolves them by merging adjacent stays, redistributing days, or reordering cities while maintaining the total trip duration.

4 Experimental Setup

4.1 Datasets

We evaluate TRAVLINKTO using the TravelPlanner benchmark (Xie et al., 2024). This dataset consists of a validation set with 180 queries categorized into three difficulty levels, and a test set of 1,000 randomly distributed queries. The difficulty levels are defined by specific constraints. Table 4 in the Appendix contains detailed information about these constraints for each difficulty level.

4.2 Evaluation Metrics

We evaluate the generated travel plans based on three dimensions defined in TravelPlanner (Xie et al., 2024): *Commonsense* (8 criteria), *Hard Constraints* (5 criteria), and *Environmental Feasibility* (2 criteria). More descriptions for criteria are in Table 5 of Appendix. Each criterion is assessed using binary values (1 for satisfaction, 0 for failure). In this work, we employ four metrics for evaluation. The *Delivery Rate* measures whether a plan is generated within 30 rounds. The *Commonsense Pass Rate* and *Hard Constraint Pass Rate* evaluate their respective criteria using micro-average (proportion satisfied) and macro-average (1 if all are satisfied, 0 otherwise). The *Final Pass Rate* is 1 only if all criteria are met for a query.

4.3 Competing Baselines

Greedy search. Following TravelPlanner (Xie et al., 2024), we implement a greedy search algorithm as a baseline method. The algorithm first selects destination cities from search results based on the specified trip duration. Each daily itinerary adopts a cost-minimizing strategy for transportation, dining, and accommodation selections while randomly choosing attractions.

LLMs and planning strategies. TRAVLINKTO can be interfaced with any LLM. In our evaluation, we interface TRAVLINKTO with OpenAI's GPT-3.5-Turbo and Google Gemini, using `text-embedding-3-small` as the embedding model. For comparison, we implement the ReAct framework (Yao et al., 2023) with GPT-3.5-Turbo, GPT-4-Turbo (OpenAI, 2024), and Gemini Pro (Team et al., 2023). React is a reasoning framework for LLMs that processes user queries by reasoning, invoking predefined functions to interact with external databases and generating a final plan. The key difference between TRAVLINKTO and ReAct is that TRAVLINKTO employs graph-based spatial recommendations and a self-refinement mechanism, whereas ReAct relies on free-form summarization and LLM-based reasoning.

We compare TRAVLINKTO with TRE. It shares a similar structure with TRAVLINKTO, including comparable query parsing and identification of missing data retrieval, but differs in the validation stage. TRE utilizes an LLM to summarize the retrieved information based on a specifically designed prompt. More details are in Appendix 7.5.

5 Experimental Results

5.1 Overall Results

Table 3 reports the performance of TRAVLINKTO integrated with two LLMs, comparing it against other LLMs and the greedy search strategy. Higher metric values indicate better performance, with the best results highlighted in bold.

Table 3: Main results of different LLMs and planning strategies on the TravelPlanner validation and test set (in %). Higher scores indicate better performance.

	Validation (#180)						**Test** (#1000)					
	Delivery Rate	Commonsense Pass Rate		Hard Constraint Pass Rate		Final Pass Rate	Delivery Rate	Commonsense Pass Rate		Hard Constraint Pass Rate		Final Pass Rate
		Micro	Macro	Micro	Macro			Micro	Macro	Micro	Macro	
Greedy Search	100.0	74.4	0.0	60.8	37.8	0.0	100.0	72.0	0.0	52.4	31.8	0.0
GPT-4-Turbo	89.4	61.1	2.8	15.2	10.6	0.6	93.1	63.3	2.0	10.5	5.5	0.6
Gemini Pro	28.9	18.9	0.0	0.5	0.6	0.0	39.1	24.9	0.0	0.6	0.1	0.0
TRAVLINKTO_Gemi	36.7	28.1	6.7	10.7	7.8	3.3	74.9	52.0	8.2	1.3	2.8	2.4
Improve (↑)	27.0%	48.7%	6.7	10.2	7.2	3.3	91.6%	109.6%	8.2	0.7	2.7	2.4
GPT-3.5-Turbo	86.7	54.0	0.0	0.0	0.0	0.0	91.8	57.9	0.0	0.5	0.6	0.0
TRAVLINKTO_GPT3.5	**98.9**	**89.1**	**45.0**	**37.1**	**20.0**	**13.9**	**98.7**	**79.7**	**25.9**	**21.4**	**11.3**	**6.7**
Improve (↑)	14.1%	65.0%	45.0	37.1	20.0	13.9	7.5%	37.7%	25.9	20.9	10.7	6.7

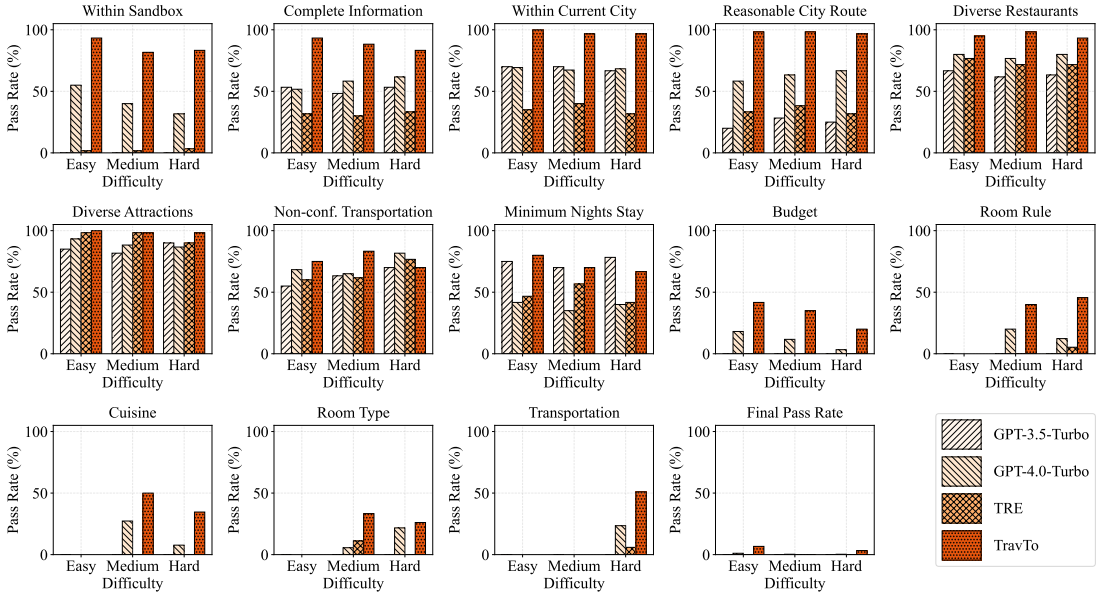


Figure 4: Performance comparison across different models and constraint types. Higher pass rates indicate better performance.

TRAVLINKTO outperforms all competing baselines across various evaluation metrics. When utilizing GPT-3.5-Turbo as the base LLM, TRAVLINKTO enhances the delivery rate from 86.7% to 98.9% and increases the final pass rate from 0% to 13.9%. For Gemini Pro, the improvements are from 28.9% to 36.7% in delivery rate and from 0% to 3.3% in final pass rate. Notably, despite using a relatively weaker base LLM (GPT-3.5-Turbo), TRAVLINKTO surpasses the ReAct scheme of GPT-4-Turbo, achieving a higher delivery rate (98.9% versus 89.4%) and a final pass rate (13.9% versus 0.6%). This demonstrates TRAVLINKTO's effectiveness in completing tasks and meeting constraints in LLM-based travel planning.

TRAVLINKTO shows significant advancements in managing both commonsense and hard constraints. For commonsense constraints, it achieves relative improvements ranging from 37% to 65% in micro-average pass rates and from 6% to 45% in macro-average pass rates compared to baseline models. Regarding hard constraints, while ReAct with GPT-3.5-Turbo achieves a pass rate of 0%, TRAVLINKTO using the same LLM reaches a micro-average pass rate of 37.1% and a macro-average pass rate of 20.0%.

These enhancements are primarily attributed to the benefits offered by TRAVLINKTO, including spatial-aware recommendations (see Section 5.2), a self-refinement mechanism (see Section 5.3), and practical sets of function calling (see Section 5.4). Figure 4 shows how TRAVLINKTO enhances the base LLMs across user constraints of varying difficulty levels, as defined in Table 5 of the Appendix.

5.2 Spatial-aware Recommendations in inter-city route planning

For inter-city route planning, four key metrics are evaluated: *Transport Consistency*, *Reasonable City Route*, *Within Budget*, and *Transportation*.

TRAVLINKTO significantly outperforms the baseline models in all metrics. Specifically, in *Reasonable City Route*, TRAVLINKTO reaches 96.7-98.3%, substantially outperforming GPT-3.5-Turbo (20.0-28.3%) and GPT-4-Turbo (58.3-66.7%), and in *Transportation* constraints, it reaches 51.0% compared to baseline models' 0-23.5%.

To further evaluate our system's effectiveness in cross-city travel planning, Figure 5 compares performance across two scenarios: multi-city (120 queries) and direct round-trip (60 queries). In multi-city queries, TRAVLINKTO achieves a final pass rate of 11.7% (vs. baselines' near-zero) and outperforms GPT-4-Turbo in both common (Macro: +7.8%, Micro: +18.3%) and complex constraint metrics (Macro: +22.2%, Micro: +15%). For direct round-trip queries, it attains a macro pass rate of 20.6% (vs. GPT-4-Turbo's 0.6%). These results demonstrate the effectiveness of our city-graph-based route optimization in handling complex travel scenarios.

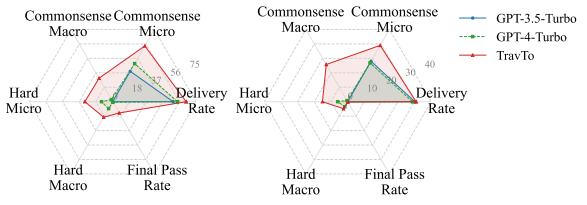
5.3 Self-refinement Mechanism in intra-city activities planning

For planning intra-city activities, we evaluate metrics across various categories, including options diversity (restaurants and attractions), user preferences (room types, cuisines, and house rules), location accuracy (activity-city matching), temporal constraints (minimum stay requirements) and environmental feasibility (within sandbox). TRAVLINKTO significantly outperforms baseline models in all metrics, demonstrating both enhanced user requirement satisfaction and hallucination mitigation.

Most notably, TRAVLINKTO demonstrates exceptional performance in hallucination prevention and location accuracy. In *Within Sandbox* tests, it achieves 81.7-93.3% pass rates (vs. GPT-3.5-Turbo's 0% and GPT-4-Turbo's 31.7-55.0%), while for *Within Current City*, it reaches 96.7-100% accuracy (vs. GPT-3.5-Turbo's 66.7-70.0%). The system also shows significant improvements in handling user preferences, achieving substantial gains in *Room Rule* (40.0-45.6%), *Cuisine* (34.6-50.0%), and *Room Type* (26.1-33.3%) compared to baseline models' 0% performance.

5.4 Planning Efficiency with Impact of Iteration Rounds

Figure 6 shows TRAVLINKTO's performance on 1000 queries across three difficulty levels and trip



(a) Multi-City (b) Direct Route

Figure 5: Performance comparison on multi-city and direct round-trip travel planning tasks.

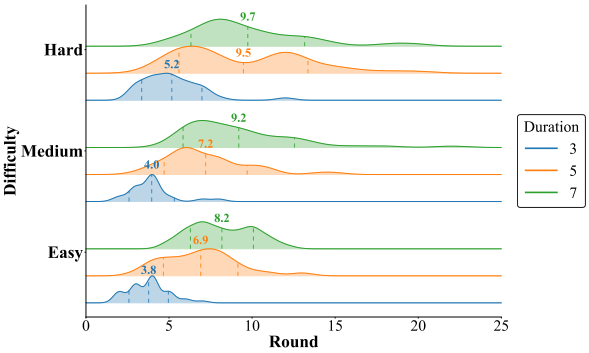


Figure 6: Distribution of rounds by difficulty and trip duration, with means (dashed lines) and standard deviations.

durations (3/5/7-day). Easy queries require 3.77–8.73 rounds on average (σ : 1.19–2.92), medium queries need 3.96–9.20 rounds (σ : 1.36–3.35), and hard queries take 5.17–9.74 rounds (σ : 2.00–3.88). All queries are completed within the 30-round limit, with longer trips generally requiring more rounds. The efficiency gains stem from TRAVLINKTO's structured JSON-like commands that enable batch generation and multi-parameter queries, allowing simultaneous processing of different planning aspects (e.g., transportation, accommodation) while avoiding iterative searches and potential errors.

6 Conclusions

We have presented TRAVLINKTO, a novel framework to enhance the spatial and temporal reasoning of LLMs for automated travel planning. The approach adopts a hierarchical strategy, utilizing knowledge graphs for inter-city routing to enhance spatial awareness while implementing an iterative self-refinement mechanism to improve planning efficiency and reliability. Extensive experiments on TravelPlanner datasets show that TRAVLINKTO significantly outperforms baseline models across multiple metrics, effectively mitigating hallucination and handling complex multi-city, long-duration travel planning tasks.

Limitations

Despite the promising results, we acknowledge several limitations of our current work.

First, the inter-city route planning in Section 3.2 is based on a relatively small dataset of 315 cities from TravelPlanner. While effective for our experiments, real-world applications would need to handle a much larger scale of cities. Our current simple route planning methods may need to be replaced with more efficient algorithms like Bidirectional A-star (Islam et al., 2016) or Multi-source Dijkstra Algorithm(Eklund et al., 1996) for larger city graphs.

Second, our current approach is limited to text-based itinerary generation. Future work could explore multi-modal travel planning, incorporating visual information and other modalities to enhance the planning experience.

Third, the template-based validation in Section 3.4.4 relies on fixed activity categories (transportation, meals, attractions, accommodation). A more flexible approach that can automatically generate templates based on user preferences would benefit personalized travel planning.

Finally, the quality of our results heavily depends on the underlying language model's ability to identify and process input parameters correctly. Traditional nature language process (Yang et al., 2024) methods might be needed as a fallback mechanism in real-world deployments when the base model fails to recognize crucial parameters.

References

Prabin Bhandari, Antonios Anastasopoulos, and Dieter Pfoser. 2023. Are large language models geospatially knowledgeable? *Preprint*, arXiv:2310.13002.

Lei Chen, Jie Cao, Haicheng Tao, and Jia Wu. 2023. Trip reinforcement recommendation with graph-based representation learning. *ACM Trans. Knowl. Discov. Data*.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. 2024. T-eval: Evaluating the tool utilization capability of large language models step by step. In *Proc. 62nd Annu. Meeting Assoc. Comput. Linguistics*, pages 9510–9529.

Marco Dorigo, Mauro Birattari, and Thomas Stutzle. 2006. Ant colony optimization. *IEEE Comput. Intell. Mag.*, 1(4):28–39.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *Preprint*, arXiv:2404.16130.

P.W. Eklund, S. Kirkby, and S. Pollitt. 1996. A dynamic multi-source dijkstra's algorithm for vehicle routing. In *Proceedings. ANZIIS 96*, pages 329–333.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *Preprint*, arXiv:2402.01680.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. 2021. Knowledge graphs. *ACM Comput. Surv.*, 54(4).

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2024. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.*

Fahad Islam, Venkatraman Narayanan, and Maxim Likhachev. 2016. A*-connect: Bounded suboptimal bidirectional heuristic search. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2752–2758.

Jinyang Li, Binyuan Hui, GE QU, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *Proc. 37th Conf. Neural Inf. Process. Syst. Datasets Benchmarks Track*.

Kwan Hui Lim, Jeffrey Chan, Christopher Leckie, and Shanika Karunasekera. 2015. Personalized tour recommendation based on user interests and points of interest visit durations. In *Proc. 24th Int. Conf. Artif. Intell.*

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

Frederic P. Miller, Agnes F. Vandome, and John McBrewster. 2009. *Levenshtein Distance: Information theory, Computer science, String (computer science), String metric, Damerau? Levenshtein distance, Spell checker, Hamming distance*. Alpha Press.

OpenAI. 2022. ChatGPT.

OpenAI. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

J. Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. 2007. *Collaborative Filtering Recommender Systems*, pages 291–324. Berlin, Heidelberg.

Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, Pranav Sandeep Dulepet, Saurav Vidyadhara, Dayeon Ki, Sweta Agrawal, Chau Pham, Gerson Kroiz, Feileen Li, Hudson Tao, Ashay Srivastava, Hevander Da Costa, Saloni Gupta, Megan L. Rogers, Inna Goncearenco, Giuseppe Sarli, Igor Galynker, Denis Peskoff, Marine Carpuat, Jules White, Shyamal Anadkat, Alexander Hoyle, and Philip Resnik. 2024. The prompt report: A systematic survey of prompting techniques.

K. Sylejmani, V. Abdurrahmani, A. Ahmeti, et al. 2024. Solving the tourist trip planning problem with attraction patterns using meta-heuristic techniques. *Inf. Technol. Tourism*, 26:633–678.

Google Gemini Team, Rohan Anil, Sebastian Borgeaud, Yuhuai Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, et al. 2023. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

C Valliyammai, R PrasannaVenkatesh, C Vennila, and S Gopi Krishnan. 2017. An intelligent personalized recommendation for travel group planning based on reviews. In *Proc. 8th Int. Conf. Adv. Comput.*

Karthik Valmeekam, Sarath Sreedharan, Matthew Marquez, Alberto Olmo, and Subbarao Kambhampati. 2023. On the planning abilities of large language models (a critical investigation with a proposed benchmark). *Preprint*, arXiv:2302.06706.

Hongru Wang, Yujia Qin, Yankai Lin, Jeff Z. Pan, and Kam-Fai Wong. 2024a. Empowering large language models: Tool learning for real-world interaction. In *Proc. 47th Int. ACM SIGIR Conf. Res. Dev. Inf. Retrieval*, page 2983–2986.

Jianguo Wang, Eric Hanson, Guoliang Li, Yannis Papakonstantinou, Harsha Simhadri, and Charles Xie. 2024b. Vector databases: What's really new and what's next? (vldb 2024 panel). *Proc. VLDB Endow.*, 17(12):4505–4506.

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 *Proc. 41st Int. Conf. Mach. Learn.*

Zhenxing Xu, Ling Chen, Yimeng Dai, and Gencai Chen. 2017. A dynamic topic model and matrix factorization-based travel recommendation method exploiting ubiquitous data. *IEEE Trans. Multimedia*, 19(8):1933–1945.

Lu Yang, Wenhe Jia, Shan Li, and Qing Song. 2024. Deep learning technique for human parsing: A survey and outlook. *IJCV*, 132(8):3270–3301.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023. Gpt4tools: Teaching large language model to use tools via self-instruction. *Preprint*, arXiv:2305.18752.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *Proc. 11th Int. Conf. Learn. Represent.*

Yu Zheng, Qianyue Hao, Jingwei Wang, Changzheng Gao, Jinwei Chen, Depeng Jin, and Yong Li. 2024. A survey of machine learning for urban decision making: Applications in planning, transportation, and healthcare. *ACM Comput. Surv.*, 57(4).