ItiNera: Integrating Spatial Optimization with Large Language Models for Open-domain Urban Itinerary Planning

Yihong Tang[1,2*], **Zhaokai Wang**[3*], **Ao Qu**[1,4*], **Yihao Yan**[1*], **Zhaofeng Wu**[4]
Dingyi Zhuang[1,4], **Jushi Kai**[3], **Kebing Hou**[1], **Xiaotong Guo**[4]
Jinhua Zhao[4✉], **Zhan Zhao**[2✉], **Wei Ma**[5✉]

[1]Tutu AI [2]University of Hong Kong [3]Shanghai Jiao Tong University
[4]Massachusetts Institute of Technology [5]The Hong Kong Polytechnic University

yihongt@connect.hku.hk {wangzhaokai,json.kai}@sjtu.edu.cn {qua,zfw,dingyi,xtguo,jinhua}@mit.edu
{yanyihao,houkebing}@tutu-ai.com zhanzhao@hku.hk wei.w.ma@polyu.edu.hk

Abstract

Citywalk, a recently popular form of urban travel, requires genuine personalization and understanding of fine-grained requests compared to traditional itinerary planning. In this paper, we introduce the novel task of Open-domain Urban Itinerary Planning (OUIP), which generates *personalized* urban itineraries from user requests in natural language. We then present ItiNera, an OUIP system that integrates spatial optimization with large language models to provide customized urban itineraries based on user needs. This involves decomposing user requests, selecting candidate points of interest (POIs), ordering the POIs based on cluster-aware spatial optimization, and generating the itinerary. Experiments on real-world datasets and the performance of the deployed system demonstrate our system's capacity to deliver personalized and spatially coherent itineraries compared to current solutions. Source codes of ItiNera are available at `https://github.com/YihongT/ITINERA`.

1 Introduction

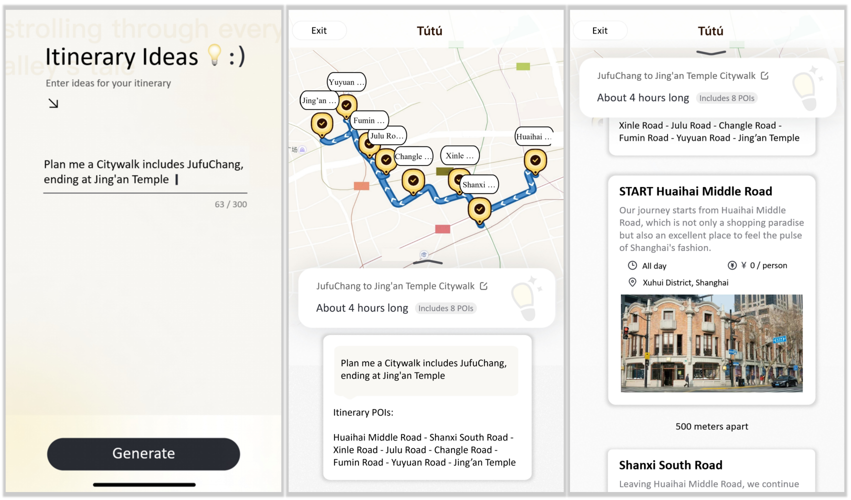
As a novel form of urban travel, citywalk (Germano, 2023) invites travelers to wander through city streets and immerse themselves in local culture, offering a more dynamic, immersive, and fine-grained travel experience compared to traditional tourism. Planning a citywalk is a complex urban itinerary planning problem (Halder et al., 2024), involving travel-related information gathering, POI selection, route mapping, and customization for diverse user needs. Specifically, citywalk differs from traditional tourism by (1) Dynamic Information: involving rapidly changing POIs and needing up-to-date information on temporary events, (2) Personalization: prioritizing individual preferences over widely recognized POIs, and (3) Diverse Constraints: considering complex constraints like personal interests and accessibility requirements. An example of the OUIP problem is shown in Fig. 1.

Existing itinerary planning studies focus on traditional tourism. They consider coarse-grained user requirements such as geographical constraints (Rani et al., 2018) and time budgets (Hsueh and Huang, 2019) to improve the quality of an itinerary (Chen et al., 2013; Sylejmani et al., 2017). While these optimization-based approaches maintain the quality of POIs and spatial coherency, they struggle to address dynamic and detailed personal demands, leading to itineraries that lack personalization and diversity.

Recently, large language models (LLMs) (OpenAI, 2023) have shown impressive applications in understanding user needs and following instructions. However, their limitations in itinerary planning are evident (Xie et al., 2024): (1) Pure LLMs cannot refer to specific POI lists, resulting in outdated or hallucinated POIs. (2) LLMs lack the optimization capabilities required for planning tasks, leading to suboptimal itineraries. Consequently, LLM-generated itineraries can be circuitous, lack detail, and include impractical information.

To address these limitations, in this work, we first define the Open-domain Urban Itinerary Planning (OUIP) problem, which involves *generating personalized travel itineraries based on user requests in natural language*. Then, we propose ItiNera, a holistic OUIP system that integrates spatial optimization with LLMs. ItiNera comprises five LLM-assisted modules: *User-owned POI Database Construction* (UPC), *Request Decomposition* (RD), *Preference-aware POI Retrieval* (PPR), *Cluster-aware Spatial Optimization* (CSO),

*Equal contribution. ✉Corresponding authors.



Plan me a Citywalk includes JufuChang, ending at Jing'an Temple

JufuChang to Jing'an Temple Citywalk

About 4 hours long Includes 8 POIs

Itinerary POIs:

Huaihai Middle Road - Shanxi South Road - Xinle Road - Julu Road - Changle Road - Fumin Road - Yuyuan Road - Jing'an Temple

START Huaihai Middle Road

Our journey starts from Huaihai Middle Road, which is not only a shopping paradise but also an excellent place to feel the pulse of Shanghai's fashion.

Shanxi South Road

Leaving Huaihai Middle Road, we continue

Figure 1: The OUIP problem and the OUIP system.

and *Itinerary Generation* (IG), to deliver personalized and spatially coherent itineraries.

Our overall contributions are:

- We introduce the OUIP problem to provide personalized urban travel itineraries based on users' natural language inputs and propose metrics to measure the quality of generated itineraries.
- We develop ITINERA, an LLM-based OUIP system that combines spatial optimization with LLMs to create fine-grained urban itineraries tailored to users' requests.
- Extensive experiments on the real-world dataset and performance of the deployed system show that ITINERA creates personalized, spatially coherent, and high-quality urban itineraries that meet user requirements.

2 Related Work

LLMs in Urban Applications Since ChatGPT, LLMs have demonstrated strong knowledge and reasoning capabilities. Recent studies highlight the potential of LLMs in urban data processing (Yan et al., 2023) and urban planning (Zhou et al., 2024). These works reveal LLMs' capabilities in predicting human mobility patterns (Mo et al., 2023; Xue et al., 2022) and emphasize their predictive strength (Wang et al., 2023). In transportation, LLMs contribute to traffic safety analysis (Zheng et al., 2023a), enhance traffic forecasting (de Zarzà et al., 2023), and automate accident report generation (Zheng et al., 2023b) , showing their applicability in urban transportation. Leveraging LLMs for travel planning has recently gained public interest. TravelPlanner (Xie et al., 2024) proposes a sandbox environment with various tools for benchmarking LLMs on multi-day travel planning, revealing LLMs' current limitations for complex planning tasks. Unlike TravelPlanner, our system focuses on fine-grained OUIP, addressing urban itinerary planning within a single day, but can be seamlessly extended to multi-day travel planning.

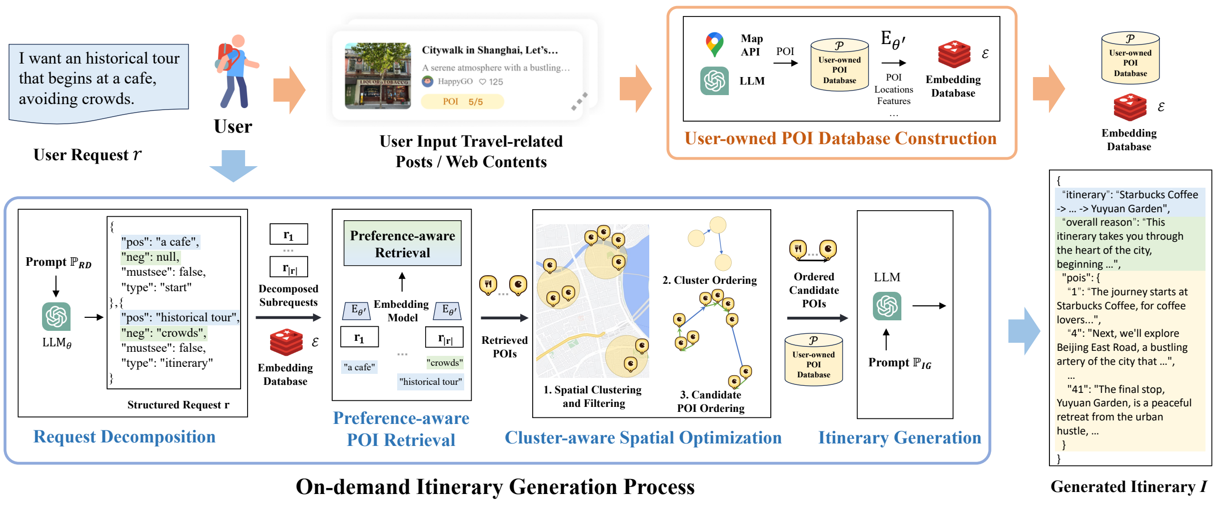
Itinerary Planning (IP) Current research on IP focuses on creating itineraries based on a set of POIs. Some methods directly optimize the spatial utilities of the itinerary, while others define IP as an Orienteering Problem (OP) and consider constraints that include time (Zhang and Tang, 2018; Hsueh and Huang, 2019), space (Rani et al., 2018), must-see POIs (Taylor et al., 2018), categories (Bolzoni et al., 2014) , and their combinations (Gionis et al., 2014; Yochum et al., 2020), to indirectly ensure the spatial coherence and quality of the itinerary. However, their ability to personalize is limited. Recommendation-based methods (Ho and Lim, 2022; Tang et al., 2022) could be applied to the IP task, but they depend on historical user behavior data. Overall, existing IP methods struggle with open-domain, user natural-language inputs, failing to generate personalized itineraries, making them unsuitable for OUIP.

3 Methodology

We formalize the OUIP problem and explain how ITINERA generates itineraries, as shown in Fig. 2.

3.1 Open-domain Urban Itinerary Planning (OUIP) Problem

To enable personalized OUIP, an open-domain system is essential. Such a system allows users to freely express their diverse requirements and expec-


I want an historical tour that begins at a cafe, avoiding crowds.
User Request r
User
Citywalk in Shanghai, Let's...
A serene atmosphere with a bustling...
HappyGO 125
POI 5/5
User Input Travel-related Posts / Web Contents
Map API
LLM
POI
User-owned POI Database
$\mathcal{P}$
$E_{\theta'}$
POI Locations Features ...
Embedding Database
$\mathcal{E}$
User-owned POI Database Construction
Prompt $\mathbb{P}_{RD}$
LLM$_\theta$
Structured Request r
Request Decomposition
r_1
$r_{|r|}$
Decomposed Subrequests
Preference-aware Retrieval
Embedding Model
"a cafe"
"crowds"
"historical tour"
Preference-aware POI Retrieval
Retrieved POIs
1. Spatial Clustering and Filtering
2. Cluster Ordering
3. Candidate POI Ordering
Cluster-aware Spatial Optimization
Ordered Candidate POIs
Prompt $\mathbb{P}_{IG}$
Itinerary Generation
On-demand Itinerary Generation Process
Generated Itinerary I

```
{
  "pos": "a cafe",
  "neg": null,
  "mustsee": false,
  "type": "start"
},{
  "pos": "historical tour",
  "neg": "crowds",
  "mustsee": false,
  "type": "itinerary"
}
```

```
{
  "itinerary": "Starbucks Coffee -> ... -> Yuyuan Garden",
  "overall reason": "This itinerary takes you through the heart of the city, beginning ...",
  "pois": {
    "1": "The journey starts at Starbucks Coffee, for coffee lovers...",
    "4": "Next, we'll explore Beijing East Road, a bustling artery of the city that ...",
    ...
    "41": "The final stop, Yuyuan Garden, is a peaceful retreat from the urban hustle, ...
  }
}
```

Figure 2: An overview of the proposed ITINERA system.

tations, enabling the planning of urban itineraries tailored to their specific needs and purposes.

Formally, given a user request r in natural language and the user-owned POI database $\mathcal{P} = \{p_j\}_{j=1}^{N}$, the OUIP problem aims to find an itinerary generator $\mathcal{G}$ to select and order a subset of POIs from $\mathcal{P}$ to create a coherent travel itinerary I as an ordered list of POIs that optimally aligns with the user's requests r while adhering to spatio-temporal considerations: $I \sim \mathcal{G}\left(\mathcal{P}|r\right)$.

3.2 User-owned POI Database Construction

Travelers often have specific places they want to visit or particular requirements for the POIs in their itinerary. To ensure a fully personalized itinerary planning process, we have developed an automated pipeline that extracts POIs and relevant details from social media, catering to these individual needs. Users can input travel post links, and the pipeline uses LLMs to extract POIs and their descriptions, calls Map APIs and embedding models to obtain their locations and embeddings, and integrates the information into the user-owned POI database $\mathcal{P}$ and embedding database $\mathcal{E}$.

The user-owned POIs enable users to create personalized POI databases, maintain timely POI information, and customize travel itineraries, enhancing itinerary experiences. We execute a daily routine to aggregate POIs from trending posts across multiple cities and maintain an up-to-date, dynamic and comprehensive POI database. This database serves as the initial set of POIs for any new user, substantially mitigating the potential cold start issue for POI acquisition. The pipeline and the database format are detailed in §E and §B.

3.3 Request Decomposition

Upon receiving user requests, we use LLMs to structure and extract information. A single user request r can be decomposed into multiple independent subrequests, each reflecting preferences at different levels and classified by granularity, specificity, and attitude. Granularity includes (1) POI-level and (2) itinerary-level subrequests. Specificity has (1) specific and (2) vague subrequests. Attitude distinguishes (1) positive subrequests (likes) and (2) negative subrequests (dislikes).

We prompt the LLM to decompose the user request r based on these categories. Formally, we obtain the resulting set of structured subrequests $\mathcal{R} = \{\mathbf{r}_i\}_{i=1}^{|\mathcal{R}|}$ through: $\mathcal{R} \sim \mathrm{LLM}\left(\mathbb{P}_{RD}(r)\right)$, where $\mathbb{P}_{RD}$ wraps the request r with instructions and examples (see §F.1). Here, 'pos' and 'neg' indicate attitude-level subrequests. 'Mustsee' is a boolean for specificity-level subrequests, and 'type' indicates granularity-level subrequests, which can be one of ["start" (POI-level origin), "end" (POI-level destination), "POI" (POI-level), "itinerary" (itinerary-level)].

3.4 Preference-aware POI Retrieval

After obtaining decomposed subrequests, we select POIs from the user-owned POI database $\mathcal{P}$ that match their preferences. While LLMs excel in language comprehension, they are limited by context window size and input token cost. Given the vast amount of POI data and LLM inference speed limitations, we design a preference-aware embedding-based retrieval approach. For a subrequest $\mathbf{r}_i$, we first use an embedding model $\mathrm{E}_{\theta'}$ to encode the 'pos' and 'neg' fields: $\mathbf{e}_i^{\mathrm{pos}} = \mathrm{E}_{\theta'}\left(\mathbf{r}_i^{\mathrm{pos}}\right); \mathbf{e}_i^{\mathrm{neg}} =$

$\mathrm{E}_{\theta'}(\mathbf{r}_i^{\text{neg}})$, where θ' denotes the parameters of the E, and $\mathbf{e}_i^{\text{neg}}, \mathbf{e}_i^{\text{pos}} \in \mathbb{R}^d$ are embeddings.

Ideally, we want the queried POIs to best fit the positive subrequest while avoiding the negative subrequest. To achieve this, we use the positive embedding $\mathbf{e}_i^{\text{pos}}$ to retrieve k POIs from $\mathcal{P}$ to obtain $\mathcal{P}_i^{\text{pos}} = \{p_{i,j}^{\text{pos}}\}_{j=1}^k$ and the corresponding embeddings $\mathcal{E}_i^{\text{pos}} = \{\mathbf{e}_{i,j}^{\text{pos}}\}_{j=1}^k$ from $\mathcal{E}$ with top similarity scores $\mathcal{S}_i^{\text{pos}} = \{s_{i,j}^{\text{pos}}\}_{j=1}^k$, where $p_{i,j}^{\text{pos}}$ and $s_{i,j}^{\text{pos}}$ represent the jth POI and score for ith positive sub-request. Next, we compute the similarity scores between $\mathcal{E}_i^{\text{pos}}$ and $\mathbf{e}_i^{\text{neg}}$ and rerank the POIs based on the gap between positive and negative similarity scores. Using $\mathcal{E} \in \mathbb{R}^{N\times d}$ to denote the pre-computed POI embeddings in the user-owned database, the process is:

$$\mathcal{P}_i^{\text{pos}}, \mathcal{S}_i^{\text{pos}}, \mathcal{E}_i^{\text{pos}} = \text{score}^k(\mathbf{e}_i^{\text{pos}}, \mathcal{E}) \tag{1}$$

$$\mathcal{P}_i^{\text{neg}}, \mathcal{S}_i^{\text{neg}}, \mathcal{E}_i^{\text{neg}} = \text{score}(\mathbf{e}_i^{\text{neg}}, \mathcal{E}_i^{\text{pos}}) \tag{2}$$

$$\mathcal{P}_i, \mathcal{S}_i = \text{rank}(\mathcal{P}_i^{\text{pos}}, \mathcal{S}_i^{\text{pos}} - \mathcal{S}_i^{\text{neg}}), \tag{3}$$

where the $\text{score}(\cdot)$ function measures embedding similarities, and the superscript k indicates it returns the top-k results . The $\text{rank}(\cdot)$ reorders POIs from highest to lowest similarity scores.

Lastly, we select the top-k POIs with the highest summed scores from the union of all retrieved POIs to form the final set $\mathcal{P}^{rt}$ for the user request r:

$$\mathcal{P}^{rt}, \mathcal{S}^{rt} = \text{rank}^k\left(\cup_{i=1}^{|\mathcal{R}|}(\mathcal{P}_i, \mathcal{S}_i)\right) \cup (\mathcal{P}^{\text{must}}, \mathcal{S}^{\text{must}}), \tag{4}$$

where $\mathcal{S}^{\text{must}}$ has large values to ensure must-see POIs are included. During the union process, scores for the same POI under different subrequests are summed to obtain the final score.

3.5 Cluster-aware Spatial Optimization

3.5.1 Spatial Clustering and Filtering

A spatially coherent itinerary enhances the travel experience by allowing travelers to move efficiently between clusters of POIs, reducing transit time and effort (Bolzoni et al., 2014). Therefore, spatially filtering and sequencing the retrieved POIs is essential. To achieve this, we compute spatial clusters of the retrieved POIs and select candidates based on proximity and matching scores, addressing cluster-aware spatial optimization by solving a hierarchical traveling salesman problem (Jiang et al., 2014). Given the retrieved POIs $\mathcal{P}^{rt}$, we create a spatial proximity graph G using a distance matrix D, where each node is a POI and edges connect nodes within a distance threshold τ. A community detection algorithm divides G into clusters. We iteratively select the cluster with the highest summed similarity score until the total number of selected POIs reaches a threshold N^c, forming the candidate POIs $\mathcal{P}^c$ for the user request r. The process is detailed in Algo. 1.

Algorithm 1 Spatial Clustering & Candidate POI Selection

```
    Input: Retrieved POI set $\mathcal{P}^{rt}$ with scores $\mathcal{S}^{rt}$, Distance threshold $\tau$, Candidate POIs num threshold $N^c$
    Output: Spatial Clusters $\mathcal{C}$, Candidate POIs $\mathcal{P}^c$
 1: // SPATIAL CLUSTERING
 2: $G \leftarrow (V, E)$ with $V \leftarrow \mathcal{P}^{rt}, E \leftarrow \emptyset; \mathcal{C} \leftarrow \emptyset; \mathcal{P}^c \leftarrow \emptyset$
 3: for $p_a^{rt}, p_b^{rt} \in V$ with $a \neq b$ do
 4:     if $distance(p_a^{rt}, p_b^{rt}) < \tau$ then
 5:         $E \leftarrow E \cup \{(p_a^{rt}, p_b^{rt})\}$
 6:     end if
 7: end for
 8: while $V \neq \emptyset$ do
 9:     $c \leftarrow$ largest clique in $G$
10:     $\mathcal{C} \leftarrow \mathcal{C} \cup \{c\}; V \leftarrow V \setminus c$
11: end while
12: // SELECTION OF CANDIDATE POIS
13: for each cluster $c \in \mathcal{C}$ do
14:     $s_c^{rt} \leftarrow \sum_{p_j \in c} s_j^{rt}$
15: end for
16: Sort $\mathcal{C}$ in descending order of $\mathcal{S}^c = \{s_c^{rt}\}_{c=1}^{\mathcal{C}}$
17: while $|\mathcal{P}^c| < N^c$ do
18:     $c_{\max} \leftarrow \arg\max_{c \in \mathcal{C}} s_c^{rt}$
19:     $\mathcal{P}^c \leftarrow \mathcal{P}^c \cup c_{\max}; \mathcal{C} \leftarrow \mathcal{C} \setminus \{c_{\max}\}$
20: end while
21: return $\mathcal{C}, \mathcal{P}^c$
```

3.5.2 POI Ordering via Solving Hierarchical TSP

Algorithm 2 Hierarchical TSP for POI Ordering

```
    Input: Spatial clusters $\mathcal{C}$, candidate POIs $\mathcal{P}^c$, distance matrix $D$
    Output: Ordered list of candidate POIs $\mathcal{P}^{\text{order}}$
 1: // POI ORDERING
 2: $\mathcal{C}^{\text{order}} \leftarrow$ SolveTSP$(\mathcal{C}, D)$; $\mathcal{P}^{\text{order}} \leftarrow \emptyset$
 3: for each cluster $c$ in $\mathcal{C}^{\text{order}}$ do
 4:     $p^c_{\text{start}}, p^c_{\text{end}} \leftarrow$ GetClusterEndpoints$(c, \mathcal{C}^{\text{order}}, D)$
 5:     $c_{\text{path}} \leftarrow$ SolveConstraintTSP$(c, p^c_{\text{start}}, p^c_{\text{end}}, D)$
 6:     $\mathcal{P}^{\text{order}} \leftarrow \mathcal{P}^{\text{order}} \cup c_{\text{path}}$
 7: end for
 8: // START POI SELECTION AND POI REORDERING
 9: $p_{\text{start}} \leftarrow$ Select$(\mathcal{P}^{\text{order}})$
10: $\mathcal{P}^{\text{order}} \leftarrow$ Reorder$(\mathcal{P}^{\text{order}}, p^{\text{order}}_{\text{start}})$
11: return $\mathcal{P}^{\text{order}}$
```

After obtaining the spatial clusters $\mathcal{C}$, we optimize the access order of the candidate POIs for a spatially coherent itinerary by determining the access order of each cluster and solving TSPs within each cluster with start and end POI constraints. Start and end points are selected based on the proximity of POIs in adjacent clusters, as shown in Fig. 2. This process, outlined in Algo. 2, optimizes and ensures coherent traversal among selected POIs using an efficient hierarchical TSP approach. 'SolveTSP' and 'SolveTSPWithEndpoints' handle standard and constrained TSPs, respectively, while 'GetClusterEndpoints' determines the start

Method	Shanghai							Qingdao						
	Rule-based Metrics				LLM-Eval ↑ (%)			Rule-based Metrics				LLM-Eval ↑ (%)		
	RR ↑ (%)	AM ↓ (meters)	OL ↓	FR ↓ (%)	PQ	IQ	Match	RR ↑ (%)	AM ↓ (meters)	OL ↓	FR ↓ (%)	PQ	IQ	Match
Ground Truth	/	124.4	0.44	/	68.9	61.5	80.9	/	356.6	0.31	/	75.4	63.9	71.4
IP	6.4	1573.3	2.96	/	30.3	26.2	17.8	7.6	4134.3	2.86	/	23.6	16.8	20.2
Ernie-Bot 4.0	15.7	513.5	0.91	15.2	42.1	46.5	42.5	27.2	776.2	0.78	21.4	43.4	38.2	33.3
GPT-3.5	16.6	422.4	0.83	13.5	40.4	43.1	45.4	25.5	691.5	0.55	22.0	33.4	39.0	46.6
GPT-4	18.0	267.2	0.56	8.2	45.0	48.2	46.9	27.3	569.4	0.49	19.6	46.6	48.7	48.4
GPT-4 CoT	18.4	258.3	0.49	7.5	/	/	/	30.2	542.6	0.43	17.8	/	/	/
ItiNera (ours)	**31.4**	**86.0**	**0.42**	/	**69.8**	**64.6**	**72.0**	**35.4**	**225.8**	**0.26**	/	**71.2**	**68.2**	**67.8**
Method	Beijing							Hangzhou						
	Rule-based Metrics				LLM-Eval ↑ (%)			Rule-based Metrics				LLM-Eval ↑ (%)		
	RR ↑ (%)	AM ↓ (meters)	OL ↓	FR ↓ (%)	PQ	IQ	Match	RR ↑ (%)	AM ↓ (meters)	OL ↓	FR ↓ (%)	PQ	IQ	Match
Ground Truth	/	218.3	0.53	/	61.9	57.3	77.0	/	70.9	0.34	/	47.5	53.2	66.3
IP	3.3	3034.2	2.26	/	27.8	18.2	20.4	1.8	1744.4	1.52	/	34.8	31.4	22.5
Ernie-Bot 4.0	18.8	379.4	0.74	12.8	31.2	34.8	32.1	12.9	605.2	1.17	24.4	43.6	34.3	38.2
GPT-3.5	19.7	347.8	0.58	14.3	29.2	40.5	43.8	14.4	665.4	1.16	19.8	41.2	40.8	32.8
GPT-4	20.6	342.6	0.52	11.1	45.4	43.6	45.2	14.8	746.1	1.09	23.2	46.2	39.6	39.4
GPT-4 CoT	21.0	327.7	0.54	10.2	/	/	/	15.5	455.0	1.09	18.6	/	/	/
ItiNera (ours)	**28.4**	**79.2**	**0.46**	/	**59.2**	**67.6**	**75.2**	**21.4**	**30.5**	**0.12**	/	**61.6**	**65.4**	**68.3**

Table 1: Overall results on four datasets. LLM-evaluated metrics are win rates against GPT-4 CoT.

and end points for each cluster. The starting point, p_{start}, is identified by 'Select' from subrequests $\mathcal{R}$ or by prompting an LLM with $\mathcal{P}^c$ and r. Finally, the 'Reorder' function arranges the POIs in the original order of precedence starting from p_{start}. Further details are in §D.

3.6 Itinerary Generation

Selecting a subset from $\mathcal{P}^{\text{order}}$ ensures a spatially coherent itinerary, but a high-quality itinerary must also meet constraints like time availability and practicality. It should, for example, avoid consecutive restaurant visits and schedule activities appropriately, such as bars in the evening or coffee shops in the morning. Traditional optimization algorithms can become overly complex and lack variability (Yochum et al., 2020; Taylor et al., 2018), hindering itinerary diversity. To address this, we leverage the advanced reasoning and planning capabilities of LLMs to generate final itineraries that meet these diverse constraints.

Specifically, the primary objective of this module is to effectively utilize LLM to select an optimal subset from $\mathcal{P}^{\text{order}}$, which closely aligns with user requests while adhering to various constraints. This process can be formally defined as follows:

$$I \sim \text{LLM}\left(\mathbb{P}_{IG}\left(\mathbf{r}, \mathcal{P}^{\text{order}}, I_{ex}\right)\right), \qquad (5)$$

where I_{ex} indicates extra natural language input that improves the language quality of the generated itinerary. The prompt $\mathbb{P}_{IG}$ for generating the final itinerary contains the following parts: (1) User request information; (2) Candidate POIs with context; (3) Task description; (4) Specific constraints; (5) Language style; (6) Output format. The full prompt is provided in §F.4.

4 Experiments

4.1 Experimental Setup

We collect an urban travel itinerary dataset from four Chinese cities in collaboration with a leading travel agency specializing in single-day citywalk. Each data sample contains a user request, the corresponding urban itinerary plan, and detailed POI data. In total, the dataset covers 1233 top-rated urban itineraries and 7578 POIs. For detailed data format, sample entries, and key preprocessing methodologies employed, refer to §B.

We use GPT-4 for final itinerary generation to ensure quality and GPT-3.5 for other interactions to speed up responses. Our system and data are originally in Chinese, and we provide a translated version in this paper. Additional implementation details are in §C.

4.2 Evaluation Metrics

A satisfactory itinerary must be spatially coherent and aligned with the user's needs, so we designed the following evaluation metrics.

Rule-based Metrics (1) **Recall Rate (RR)**: the recall rate of POIs in the ground truth itinerary, which evaluates the accuracy of understanding user requests and recommending personalized POIs. (2)

Variants	UPC	RD	PPR	CSO	IG	Rule-based Metrics			LLM-Eval ↑ (%)		
						RR ↑	AM ↓	OL ↓	PQ	IQ	Match
GPT-4 CoT	×	×	×	×	✓	18.4	258.3	0.49	/	/	/
GPT-4 CoT + UPC	✓	×	✓	×	✓	34.2	240.2	0.52	65.5	61.8	70.6
ITINERA w/o RD	✓	×	✓	✓	✓	22.6	35.4	0.18	68.2	61.5	60.5
ITINERA w/o PPR	✓	✓	×	✓	✓	28.2	84.6	0.38	66.7	63.4	62.2
ITINERA w/o CSO	✓	✓	✓	×	✓	32.8	242.8	1.04	72.1	60.2	74.2
ITINERA w/ GPT-3.5	✓	✓	✓	✓	✓	27.6	79.4	0.56	67.3	58.8	61.4
ITINERA w/ LLaMA 3.1 8B	✓	✓	✓	✓	✓	27.8	90.6	0.45	66.9	58.6	63.5
ITINERA (full)	✓	✓	✓	✓	✓	31.4	86.0	0.42	69.8	64.6	72.0

Table 2: Ablation study on Shanghai dataset.

Average Margin (AM): the average difference per POI between the total distance of the generated itinerary and the shortest distance (via TSP). (3) **Overlaps (OL)**: the number of self-intersection points in the generated itinerary. AM and OL measure spatial optimization for POI visit order, with lower values being better. (4) **Fail Rate (FR)**: the percentage of POIs from LLM not matched with queried map service POIs, which assesses the LLM's information accuracy, as failed POIs are inaccessible and impact the user experience.

LLM-Evaluated Metrics The rule-based metrics are intuitive, but some aspects, like POI appeal and alignment with user requests, are hard to quantify. Thus, we propose several LLM-evaluated metrics: (1) **POI Quality (PQ)**: how interesting and diverse the POIs are; (2) **Itinerary Quality (IQ)**: the overall quality and coherence of the itinerary; (3) **Match**: the alignment between the itinerary and the user request. We use GPT-4 to rank two itineraries and compute the win rate, repeated at least 10 times for reliability. Our LLM-evaluated metrics have been shown to be consistent with human judgments, as discussed in Sec. 4.5.

4.3 Overall Results

We consider the following baselines:

- IP (Gunawan et al., 2014): A traditional IP method. We simplify it to use LLM for time budgeting and to consider POI ratings as utilities.
- Ernie-Bot 4.0 (Sun et al., 2021): The best-performing model on Chinese LLM tasks, selected as our dataset and system are in Chinese.
- GPT-3.5, GPT-4 and GPT-4 CoT (OpenAI, 2023): ChatGPT models with or without Chain-of-Thought (Wei et al., 2022).

The baseline IP and our method do not compute the Fail Rate since the candidate POIs are all from the dataset.

The result is shown in Tab. 1. our proposed ITINERA outperforms all baselines across all metrics and achieves better or comparable results compared with ground truth data. It shows a ≈30% improvement in rule-based metrics over the best baseline, demonstrating superior personalization of user experiences. It maintains spatial coherence, generating itineraries only ≈100 meters longer per POI than the shortest TSP-solved path. ITINERA is also the only method to outperform GPT-4 CoT in LLM-evaluated metrics, especially in Match. These results highlight ITINERA's effectiveness in enhancing spatial coherence and aligning with user requests in OUIP.

4.4 Ablation Study

To validate the effect of each component, we compare the following variants of ITINERA:

- GPT-4 CoT + UPC: integrates the UPC module to LLMs to generate itineraries based on user-owned POIs.
- ITINERA w/o RD: uses the entire user input string's embedding to retrieve POIs.
- ITINERA w/o PPR: quantifies the contribution of the PPR module compared to our full system.
- ITINERA w/o CSO: removes the CSO module and lets the LLM in the IG module determine the order of candidate POIs for the final itineraries.
- ITINERA w/ GPT-3.5 or LLaMA 3.1 8B: replaces GPT-4 with either GPT-3.5 or LLaMA 3.1 8B for generating the final itinerary.

We remove Fail Rate in the ablation study since all variants equipped with UPC never generate POIs not present in the database.

The results in Tab. 2 show that UPC enhances the Recall Rate and Match of the GPT-4 CoT baseline. Variants "w/o RD," "w/o PPR," and "w/ GPT-3.5" have lower Recall Rate, POI Quality, Itinerary Quality, and Match than our full model, indicating a trade-off between spatial optimization and alignment with user requests. This parallels other conditional generation tasks, where aligning with human preferences can reduce inherent system abil-

Method	Rule-based		POI Quality			Itinerary Quality			Match		
	AM	OL	Expert	User	LLM	Expert	User	LLM	Expert	User	LLM
GPT-4 CoT	511.4	0.79	3.2	3.6	30%	2.5	3.0	32%	2.9	2.6	28%
ITINERA	107.6	0.44	3.8	4.3	70%	3.2	3.8	68%	3.6	3.5	72%

Table 3: Deployed System Performance.

ity (Saharia et al., 2022; Di et al., 2021). Removing the CSO module worsens the Average Margin and Overlaps but improves Recall Rate, POI Quality, and Match, showing the full model balances alignment with spatial ability. "W/o PPR" shows that the PPR module can address LLM context window limitations and save costs. Finally, "w/ GPT-3.5" outperforms the GPT-3.5 baseline, demonstrating our system's adaptability to different LLMs.

To validate that our method is a general framework compatible with both open-source LLMs and commercial ones, we conduct experiments with LLaMA 3.1-8B-Instruct (Dubey et al., 2024), a state-of-the-art model suitable for consumer GPUs. LLaMA 3.1 offers performance comparable to GPT-3.5. Nevertheless, the performance of open-source models still lags behind commercial models. Considering the maintenance cost of hosting open-source models locally, we opt to use commercial models through API following most companies.

4.5 Deployed System Performance

Our deployed system is currently accessible to a select group of users recommended by our partnered travel agency. To verify the effectiveness of our system in real-world scenarios, we conduct human evaluations. Human evaluation has been extensively employed in prior research on generative tasks (Saharia et al., 2022; Rombach et al., 2022; Zhuo et al., 2023) where objective metrics fail to adequately assess specific dimensions of output quality. We invite 464 regular users of our system (*User*) and 33 experienced travel assistants from our partnered travel agency (*Expert*) to compare the two itineraries (randomly ordered) generated by GPT-4 CoT and our system based on their requests.

The average evaluation results in Tab. 3 show that our method is preferred by both experts and regular users across all metrics, especially for Match, validating the effectiveness of our system in real-world scenarios. The human evaluation results are consistent with the LLM evaluation win rate, indicating that the proposed LLM-evaluated metrics are appropriate and adaptable when rule-based evaluation is insufficient.

4.6 Qualitative Results

We further conduct a qualitative study to demonstrate the importance of integrating LLM with spatial optimization. Consider a user request "I'm seeking an artsy itinerary that includes exploring the river's bridges and ferries", we visualize the results from ITINERA and GPT-4 CoT in Fig. 3.

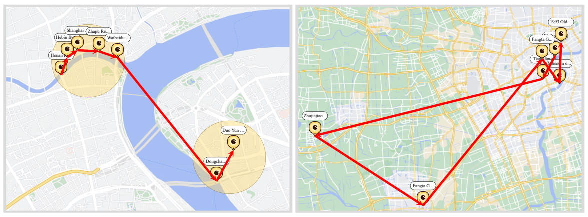


Figure 3: Generated itineraries of ITINERA (left) and GPT-4 CoT (right).

We find that our itinerary better matches the user preferences. The itinerary passes several bridges along the Huangpu River, includes a ferry crossing, and concludes at the art-atmosphere-rich Duoyun Bookstore, offering a restful endpoint for users. In contrast, the POIs selected by GPT are more mainstream. Moreover, our spatial arrangement is more logical, avoiding detours and concentrating selected POIs within two spatial clusters. The itinerary generated by GPT is spatially poor, has a disordered sequence of visits, and contains excessively distant POIs. Beyond this example, GPT also risks hallucinating non-existent POIs, highlighting the superiority of our system in comparison.

5 Conclusion

We introduce the OUIP problem and a solution ITINERA that integrates LLMs with spatial optimization. ITINERA enables the generation of personalized and spatially coherent itineraries directly from natural language requests. Experiments on the real-world dataset and deployed system performance validate the effectiveness of our approach. This study not only sets a new benchmark for itinerary planning technologies but also broadens venues for further innovations in leveraging LLMs for complex problem-solving in urban contexts.

Acknowledgement

We thank Han Zheng at MIT and Tiange Luo at UMich for participating in discussions about the initial idea and providing valuable insights.

Limitations

Despite the success of ITINERA in generating personalized itineraries, our system has several limitations. First, while the spatial optimization module works well in many cases, it may face efficiency challenges in highly complex urban environments. Moreover, although LLMs provide significant language processing capabilities, they still exhibit limitations in spatial reasoning and real-time decision-making, which may impact the quality of the generated itineraries in specific scenarios.

Ethical Statement

This study adheres to strict ethical guidelines to protect user privacy and data. The personalized POI database is user-owned, and all data processing follows legal data security and user consent standards. We have designed the system to be fair and inclusive, avoiding biases in itinerary recommendations across diverse user groups. Additionally, we emphasize environmental responsibility by ensuring that the system promotes sustainable urban tourism without adversely affecting local culture or ecosystems. Finally, transparency in the pipeline is a priority, ensuring users understand how their itineraries are generated.

References

Paolo Bolzoni, Sven Helmer, Kevin Wellenzohn, Johann Gamper, and Periklis Andritsos. 2014. Efficient itinerary planning with category constraints. In *Proceedings of the 22nd ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 203–212.

Gang Chen, Sai Wu, Jingbo Zhou, and Anthony KH Tung. 2013. Automatic itinerary planning for traveling services. *IEEE transactions on knowledge and data engineering*, 26(3):514–527.

I de Zarzà, J de Curtò, Gemma Roig, and Carlos T Calafate. 2023. Llm multimodal traffic accident forecasting. *Sensors*, 23(22):9225.

Shangzhe Di, Zeren Jiang, Si Liu, Zhaokai Wang, Leyan Zhu, Zexin He, Hongming Liu, and Shuicheng Yan. 2021. Video background music generation with controllable music transformer. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 2037–2045.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Antonello Germano. 2023. Citywalk: embracing urban charms and captivating generation z.

Aristides Gionis, Theodoros Lappas, Konstantinos Pelechrinis, and Evimaria Terzi. 2014. Customized tour recommendations in urban areas. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 313–322.

Aldy Gunawan, Zhi Yuan, and Hoong Chuin Lau. 2014. A mathematical model and metaheuristics for time dependent orienteering problem. PATAT.

Sajal Halder, Kwan Hui Lim, Jeffrey Chan, and Xiuzhen Zhang. 2024. A survey on personalized itinerary recommendation: From optimisation to deep learning. *Applied Soft Computing*, 152:111200.

Ngai Lam Ho and Kwan Hui Lim. 2022. Poibert: A transformer-based model for the tour recommendation problem. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 5925–5933. IEEE.

Yu-Ling Hsueh and Hong-Min Huang. 2019. Personalized itinerary recommendation with time constraints using gps datasets. *Knowledge and Information Systems*, 60(1):523–544.

Jingqing Jiang, Jingying Gao, Gaoyang Li, Chunguo Wu, and Zhili Pei. 2014. Hierarchical solving method for large scale tsp problems. In *International symposium on neural networks*, pages 252–261. Springer.

Baichuan Mo, Hanyong Xu, Dingyi Zhuang, Ruoyun Ma, Xiaotong Guo, and Jinhua Zhao. 2023. Large language models for travel behavior prediction. *arXiv preprint arXiv:2312.00819*.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Septia Rani, Kartika Nur Kholidah, and Sheila Nurul Huda. 2018. A development of travel itinerary planning application using traveling salesman problem and k-means clustering approach. In *Proceedings of the 2018 7th International Conference on Software and Computer Applications*, pages 327–331.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu

Karagol Ayan, Tim Salimans, et al. 2022. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 35:36479–36494.

Yu Sun, Shuohuan Wang, Shikun Feng, Siyu Ding, Chao Pang, Junyuan Shang, Jiaxiang Liu, Xuyi Chen, Yanbin Zhao, Yuxiang Lu, et al. 2021. Ernie 3.0: Large-scale knowledge enhanced pre-training for language understanding and generation. *arXiv preprint arXiv:2107.02137*.

Kadri Sylejmani, Jürgen Dorn, and Nysret Musliu. 2017. Planning the trip itinerary for tourist groups. *Information Technology & Tourism*, 17:275–314.

Yihong Tang, Junlin He, and Zhan Zhao. 2022. Hgarn: Hierarchical graph attention recurrent network for human mobility prediction. *arXiv preprint arXiv:2210.07765*.

Kendall Taylor, Kwan Hui Lim, and Jeffrey Chan. 2018. Travel itinerary recommendations with must-see points-of-interest. In *Companion Proceedings of the The Web Conference 2018*, pages 1198–1205.

Xinglei Wang, Meng Fang, Zichao Zeng, and Tao Cheng. 2023. Where would i go next? large language models as human mobility predictors. *arXiv preprint arXiv:2308.15197*.

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

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*.

Hao Xue, Bhanu Prakash Voutharoja, and Flora D Salim. 2022. Leveraging language foundation models for human mobility forecasting. In *Proceedings of the 30th International Conference on Advances in Geographic Information Systems*, pages 1–9.

Yibo Yan, Haomin Wen, Siru Zhong, Wei Chen, Haodong Chen, Qingsong Wen, Roger Zimmermann, and Yuxuan Liang. 2023. When urban region profiling meets large language models. *arXiv preprint arXiv:2310.18340*.

Phatpicha Yochum, Liang Chang, Tianlong Gu, and Manli Zhu. 2020. An adaptive genetic algorithm for personalized itinerary planning. *IEEE Access*, 8:88147–88157.

Yu Zhang and Jiafu Tang. 2018. Itinerary planning with time budget for risk-averse travelers. *European Journal of Operational Research*, 267(1):288–303.

Ou Zheng, Mohamed Abdel-Aty, Dongdong Wang, Chenzhu Wang, and Shengxuan Ding. 2023a. Trafficsafetygpt: Tuning a pre-trained large language model to a domain-specific expert in transportation safety. *arXiv preprint arXiv:2307.15311*.

Ou Zheng, Dongdong Wang, Zijin Wang, and Shengxuan Ding. 2023b. Chat-gpt is on the horizon: Could a large language model be suitable for intelligent traffic safety research and applications? *ArXiv*.

Zhilun Zhou, Yuming Lin, and Yong Li. 2024. Large language model empowered participatory urban planning. *arXiv preprint arXiv:2402.01698*.

Le Zhuo, Zhaokai Wang, Baisen Wang, Yue Liao, Chenxi Bao, Stanley Peng, Songhao Han, Aixi Zhang, Fei Fang, and Si Liu. 2023. Video background music generation: Dataset, method and evaluation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15637–15647.

A Demonstration of the Deployed System

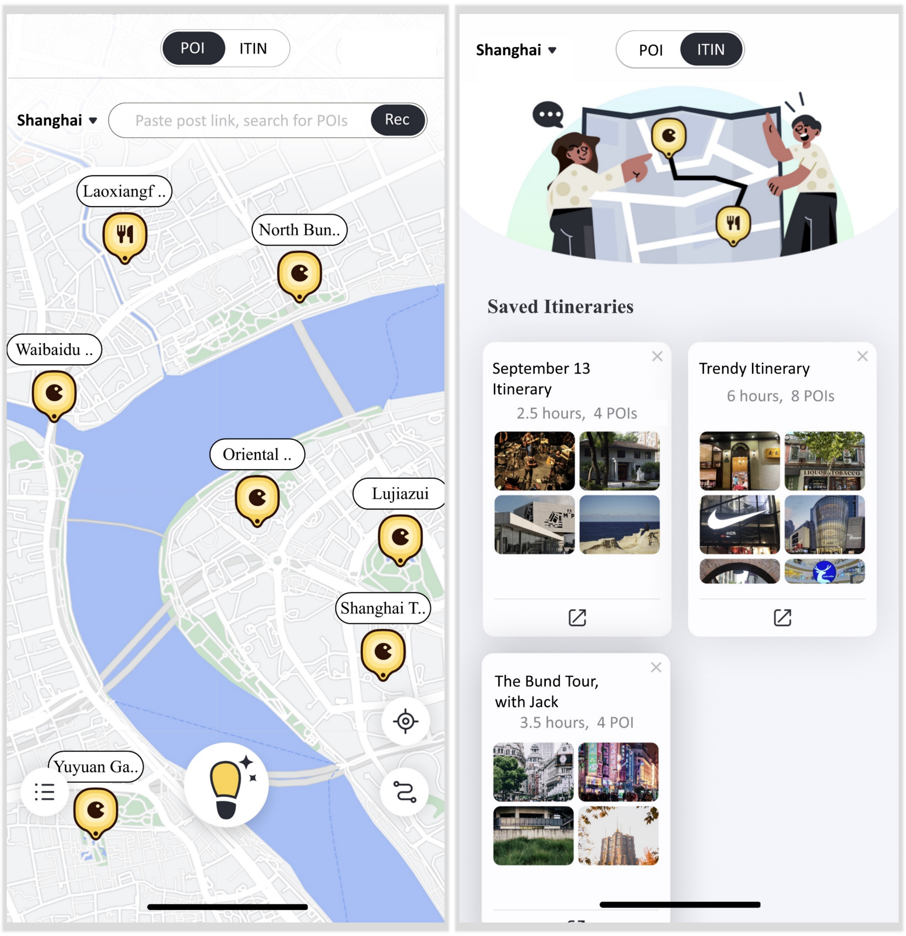


Figure 4: Screenshots of the deployed system: POI view & Itinerary view.

We provide screenshots of our deployed system in Fig. 4. The left screenshot shows the POI interface, where users can add new POIs by direct searching or pasting a link of a travel-related post. They can filter their desired POIs to display on their personal map. The POI icon represents its category (entertainment, restaurant, etc.). Users can select several POIs by pressing the bottom right button to create an itinerary. They can also use our system to generate an itinerary from natural language input (the left figure of Fig. 1).

The right screenshot shows the itinerary interface. Users can browse the itineraries they have created and generated. They can tap one itinerary to see the details (the middle and right figures of Fig. 1).

B Dataset

In this section, we provide the data format of the collected real-world dataset. Specifically, the data for each city contains two tables: one is the POI table, which primarily stores the POIs and their features, and the other is the Itinerary table, which is used to store users' natural language requests and the corresponding ground truth itineraries.

id	name	address	city	description	longitude	latitude	rating	category	context
1	The Bund	Zhongshan East 1st Rd, Huangpu	Shanghai	The Bund is a waterfront area ...	121.4906033011	31.2377704249	5.0	site	...

Table 4: POI data sample.

The sample POI data is shown in Table 4, where the context column is a concatenation of the strings from all the previous columns. The embedding of each POI is also obtained by calling $E_{\theta'}$ to embed the context field. The resulting embedding, $\mathcal{E}$, contains rich semantic information about the POIs.

user_request	itinerary
I'm seeking an artsy itinerary that includes exploring the river's bridges and ferries.	[1, 3, 6, ...]

Table 5: Itinerary data sample.

The sample itinerary data is shown in Table 5, which contains two columns: one for the user's request and the other storing a list of POI IDs representing the ground truth itinerary (label) for the user's request.

C Implementation Details

C.1 Method Implementation

We use the OpenAI text-embedding-ada-002 model for embedding purposes. The spatial coherence of itineraries is optimized through an open-source TSP solver[1]. Integration of POI data, including geographical coordinates, user ratings, categorizations, and physical addresses, is facilitated through the Amap API[2].

C.2 Baseline Settings

We use the same itinerary generation prompt for all baselines, including basic task requirements and output format, as in $\mathbb{P}_{IG}$ in §F.4. For GPT-4 CoT, we extend the prompt by integrating "thoughts", detailed in §F.5.

We adopt the $\mathbb{P}_{IT}$ for the baseline IP for time budgeting. We prompt the LLM baselines to generate itineraries based on user requests. We searched for each POI in the generated itinerary using the Map API. Here, the database associated with the Map API is considered to be the current collection of all existing POIs. We leverage fuzzy string matching[3] to determine if there is a match with specific POIs. The failed POIs contribute to the failure rate metrics. For the matched POIs, attributes of the POI (such as location) are attached to the itinerary for subsequent evaluation.

[1]https://github.com/fillipe-gsm/python-tsp
[2]https://lbs.amap.com/
[3]https://github.com/seatgeek/thefuzz

D Cluster-aware Spatial Optimization Supplementary

We present the details of the implementation of algorithms involved in cluster-aware spatial optimization.

D.1 SolveTSP

Algorithm 3 Simulated Annealing for TSP

```
1: procedure SimulatedAnnealing(cities, $T_{init}$, $T_{min}$, $\alpha$)
2:     $solution \leftarrow$ RandomSolution($cities$)
3:     $T \leftarrow T_{init}$
4:     while $T > T_{min}$ do
5:         $newSolution \leftarrow$ Neighbor($solution$)
6:         $costDifference \leftarrow$ Cost($newSolution$) $-$ Cost($solution$)
7:         if $costDifference < 0$ or $\exp(-costDifference/T) >$ Random() then
8:             $solution \leftarrow newSolution$
9:         end if
10:        $T \leftarrow \alpha \times T$
11:    end while
12:    return $solution$
13: end procedure
```

'SolveTSP' implements a simulated annealing algorithm for efficiently solving the TSP problem with a large set of candidates. Simulated annealing is a classic metaheuristic approach where the model iteratively proposes a new solution and replaces the current solution if a certain condition is satisfied until the temperature goes down to zero. We detail the implementation for simulated annealing in Algo 3.

- RandomSolution: Generates a random permutation of the cities as the initial solution.
- Neighbor: Produces a new solution by making a small change to the current solution. In our implementation, we consider four types of operations including swapping two randomly selected cities, inverting a subroute, inserting a randomly selected city to another position, and inserting a randomly selected subroute to another position.
- Cost: Calculates the total distance of the proposed solution's path.
- T_{init} and T_{min}: The initial and minimum temperatures for the SA algorithm. In our implementation, T_{init} is set to 5000 and T_{min} is set to 0.
- α: The cooling rate that determines how fast the temperature decreases. In our implementation, α is set to 0.99.

D.2 SolveTSPWithEndpoints

In each cluster, the dataset typically comprises a limited set of candidate points. Consequently, the prioritization shifts towards optimizing the accuracy of the resultant solution rather than focusing solely on computational efficiency. To address the Traveling Salesman Problem (TSP) with predetermined starting and ending points, we adopt a linear programming (LP) methodology. We detail the formulation of the linear program in Alg. 4.

Algorithm 4 SolveTSPWithEndpoints

```
Require: $dist$, $start\_point$, $end\_point$
 1: Solve the following linear program:
        Minimize:     $\min \sum_{i \neq j} x_{ij} \cdot dist[i][j]$
                      //Ensures each internal node in optimal path has in-degree 1 and out-degree 1
        Subject to:   $\sum_{i \neq k} x_{ik} = 1, \quad \forall k \neq s, e$
                      $\sum_{i \neq k} x_{ki} = 1, \quad \forall k \neq s, e$
                      //Add constraints for source node and sink node
                      $\sum_{i \neq s} x_{si} = 1$
                      $\sum_{i \neq s} x_{is} = 0$
                      $\sum_{i \neq e} x_{ie} = 1$
                      $\sum_{i \neq e} x_{ei} = 0$
                      //Eliminates all subtours
                      $\sum_{i \in S} \sum_{j \notin S, j \neq i} x_{ij} \leq |S| - 1, \quad \forall S \subset \{1, \ldots, n\}, S \neq \emptyset, S \neq \{1, \ldots, n\}$
                      //Add binary variable constraints
                      $x_{ij} \in \{0, 1\}, \quad \forall i, j$
 2: Return Optimal path
```

E Overview of the POI extraction pipeline

Prompt for Extracting POI Names and Locations

```
# Guidelines

## Task Background
Your task is to identify and extract mentioned cations in the posts/travelogues provided by users to help them quickly
find these places on a map. Now, based on the content of the post and in context, carry out the extraction and description
of Points of Interest (POIs) mentioned in the post. Focus primarily on places that can be visited, rather than merely on
place names.

## Notes on Handling POIs
1. Comprehensive Definition of POI: Typically used to describe a specific geographical location or site, such as
restaurants, hotels, streets, attractions, museums, bars, cafes, malls, etc. These locations or sites may have specific
value or interest to users or travelers.
2. Characteristics of POI: Specific places recommended or mentioned in the post that are usable for dining, entertainment,
etc.
3. Specificity: A POI refers to a specific, particular place, not a broad geographical area or city name.
4. Uniqueness: When a text is separated by symbols like "/", "&", ",", for example, "Julu Road/Tianzifang", it often
represents two POIs, in this case, "Julu Road" and "Tianzifang" should be extracted separately.
5. Examples of POI: Specific restaurants, performance venues, attractions, shops, streets, etc.
6. Non-POI Examples: Collections of places, food names, types of cuisine, performance groups, exhibition events, etc.

## Post Structure
Title: The post's title.
Text: The main body content of the post.
Text in the images: text recognized from the images.
Transcribed text: text transcribed from the video.

## Task Process
1. Extraction: Based on your reasoning, judgment, and knowledge, extract all mentioned POIs from the post.
2. Verification: In the context of each POI, ensure all POIs fit the definition and are specific places.
3. Address Information: In the context of each POI, find related address information that can be searched on a map, such
as "158 Julu Road, Shanghai."
4. Handling No Information: If no location information is available, return an empty POI list: {{}}.
5. Formatting: Organize information into the specified JSON structure.

## Output Format

### Specific Format

{{
    "POI Name": "Related Address Information for the POI"
}}

### Examples

Example 1:
If the original post mentions "Lao Nong Tang Noodle Shop in Luxi: A time-honored noodle shop that appears on Shanghai's
must-eat list all year round!", the output for this POI should be
{{
"Lao Nong Tang Noodle Shop in Luxi": null
}}

Example 2:
If the original post mentions "Red Baron (Jianye District Wentiyi Road branch)
Looking around, the most striking red on the entire Wentiyi Road, seamlessly blending with Mixue Bingcheng", the output
for this POI should be
{{
"Red Baron (Jianye District Wentiyi Road branch)": null
}}

## Output Standards

- The output is a dictionary, with keys being the POI names and values being the related address information for the POI.
If address information is missing, please use "null" to fill in.
- Ensure the output is in valid JSON format and can be parsed by Python json.loads.


## Task Start

Please begin processing the post content: ```{post_info}```.

Note: Ensure the return format follows {{Point of Interest Name: Related Address Information}}. Ensure it can be
json.loads parsed.
```

The prompt for extracting POI names and locations is provided above. As illustrated in Sec. 3.2, we design a pipeline to automatically extract POIs and relevant information from user-generated content on various social media platforms. The pipeline consists of the following steps:

- Scrape text, images, and videos from the input link of a travel-related post.
- Use automatic speech recognition to obtain transcription from the video and optical character recognition (OCR) to extract text from the images. Merge them with the original text to obtain the post information.
- Use GPT-3.5 to extract POI names and locations from the post information.
- Use map service API to look up the extracted POI names, obtain the coordinates, and extract POI names, similar to the evaluation pipeline in Sec.C.
- Use GPT-3.5 to generate POI descriptions from the POI names and post information.

The prompt for generating POI descriptions is provided below.

Prompt for Generating POI Descriptions

```
{post_info}

Based on the content of the above post, please write out the reasons for recommending each location to tourists in the
following list.
Consider what can be done at this location, its features, and why it is fun.
If the original post lacks information, you may appropriately supplement based on your knowledge, but please ensure
brevity.
The related information for each point should not exceed 30 words.
The results should be output in JSON format, specifically in the form {{Place Name: Information related to the place from
the original post}}, where "Information related to the place from the original post" should be a sentence or phrase, like
a string.
If a place does not have any relevant information, fill in the description related to the place from the original post
with "null".
```

We execute an automated process to extract POIs from the most recent trending posts and update a comprehensive POl database. At a regular interval of 24 hours, we obtain recent trending travel-related posts across multiple cities on social media platforms and run the above pipeline to extract POI names, locations, and descriptions to maintain the database.

F Prompts

F.1 Prompt for Decomposing User Requests

Prompt for Decomposing User Requests

```
Please help me break down a user request into multiple independent requirements, each including both positive and negative
requirements. Return the results in the following format directly based on the **User Request** given, without writing any
code.

### Output Format:

Return a list, where each item is a dictionary representing an independent requirement, with the following key-value pairs:
- **pos**: The positive requirement, representing what the user wants, excluding any negative requirements.
- **neg**: The negative requirement, generally what the user does not want, dislikes, or refuses. All negative
requirements must be captured in this field, for example, "non-spicy" should extract "spicy", "don't want crowded places"
should extract "crowded", "hate noisy" should extract "noisy".
- **mustsee**: Indicates whether this requirement represents a specific place name. If so, this field is `true`,
otherwise, it is `false`.
- **type**: Indicates whether the requirement is for a "place" or an "itinerary", with place having sub-types "place",
"starting point", and "ending point". Overall, this field can have the values "place", "starting point", "ending point",
or "itinerary".

- Your return should be a list in the following format:
[
    {{
        "pos": "positive requirement", (excluding negative requirements)
        "neg": "negative requirement" (what's not wanted, disliked, refused, not wanting to go or see, any negated
        requirement),
        "mustsee": true (whether it's a must-see point, all specific places should be set to true),
        "type": "place"
    }},
    ...
]
- The **positive requirement** must not be empty, and it must not include any negative requirements. All negative
requirements should be summarized in the value of the "neg" field.
- Set to null in cases where there are no **negative requirements** for a specific place.
- Sometimes users only describe what they do not want (negative requirements), in such cases, you should summarize a
**positive requirement** based on the **negative requirement**. For example, if a user says 'don't want spicy food', the
output should include: "pos" corresponding to "food", "neg" corresponding to "spicy".
- Independent requirements must have specific descriptions or demands to be considered a requirement, for example,
"recommend a route" does not count as an independent requirement.
- "mustsee" must be a specific place name, not a general term.
- If a place is definitively a "starting point" or "ending point", then the value of the "type" field should be "starting
point" or "ending point", respectively. "Starting points" and "ending points" are must-see points, with the "mustsee"
field set to true.
- A place can only be considered as a "starting point" or "ending point" if it is a specific attraction or location, and
there can only be at most one "starting point" and one "ending point".
- The return should not include any other content.

### Example Outputs:

Example 1:
User Request: "I want to start by visiting Sinan Mansions, then find something fun to do nearby, and I don't want it to be
crowded"
Output:
[
    {{
        "pos": "Sinan Mansions",
        "neg": null,
        "mustsee": true,
        "type": "starting point"
    }},
    {{
        "pos": "fun places near Sinan Mansions",
        "neg": "crowded",
        "mustsee": false,
        "type": "place"
    }}
]


### mustsee Field Assignment Examples
"mustsee" true for specific place names: "Hualian Supermarket", "Old Mac Coffee Shop", "Wukang Mansion", "Nanluoguxiang",
...
"mustsee" false for general place names: "supermarket", "milk tea shop", "bar", "coffee", ...

### Output Guidelines
- Return a list, each item in the list is a dictionary containing "pos", "neg", "mustsee", and "type" key-value pairs.
- Return as a JSON List.
- The list can be empty; if empty, just return a JSON list.
- The output should not include any other information, ensuring it can be parsed by json.loads.

### User Request
```

```
{user_req}


### Task Overview
Your task is to analyze and break down the **User Request** into independent requirements and return them.
1. First, separate the different independent requirements, breaking down each into positive and negative requirements.
2. Positive requirements should only include what the user wants, and negative requirements should only include what the
user does not want.
3. For each independent requirement, refer to **mustsee field assignment examples** to assign a value to the "mustsee"
field, analyzing whether the **positive requirement** is a specific place name. If so, set "mustsee" to true, otherwise
set it to false.
4. Refer to the **examples** and **output format** to complete the other fields.

#### Notes:
- Do not include duplicate independent requirements; ensure each independent requirement corresponds to different key
points in the user's needs.
- "Itinerary" requirements should be for the whole itinerary, such as including several places, approximate time, etc.,
all others are place requirements.
- The "type" field can only be one of ["place", "itinerary", "starting point", "ending point"].
- All attractions must be completely separated, such as "Nanluoguxiang and Drum Tower" must be split into "Nanluoguxiang"
and "Drum Tower" as two requirements.

Now, based on the **user Request**, refer to the **example outputs**, and return according to the **output guidelines**
and **output format**.
```

F.2 Prompt for Indicating Travel Time of an Itinerary

Prompt for Travel Time Indication

```
Please play the role of a top AI Travel Time Planning Assistant. Your job is to determine the time needed for a day's
itinerary based on the user request. If the user request is empty, please default to ["4"].

## Task Overview
Your task is to return the required time for a day's itinerary based on given the user request. If the user request
mention specific time constraints for the route, return the itinerary time directly as per the user's request, up to a
maximum of 8 hours (return ["8"] if it exceeds 8 hours). Please return the itinerary time directly based on the user
request, no need to write any code.

## User Request
{user_reqs}

## Input-Output Examples
- **Example 1**:
- **User Request**: "I'd like to visit a museum, enjoy authentic cuisine, and experience nightlife."
- **Output**: ["8"]

- **Example 2**:
- **User Request**: "I want to tour historical buildings and take in the city views“
- **Output**: ["6"]

- **Example 3** (In this example, the user specifies approximately **five hours** for the route):
- **User Request**: "I plan to explore the Huangpu River and Yu Garden for about five hours."
- **Output**: ["5"]

## Output Specifications
- Return a list of length 1, containing a single integer representing the required itinerary duration (in hours). The
value range is 1 to 8.
- Return as a JSON List with only one element inside.
- The list can be empty; please only return a JSON list.
- Ensure your output contains no additional information and can be parsed by json.loads.

Now, based on the **User Request**, return the time required for a day's itinerary according to the **Output
Specifications**.
```

In this work, we utilize the inference capability of LLMs to estimate the duration of an itinerary based on a user request, which is used to instruct the IG module to generate an itinerary with a reasonable duration. For more complicated considerations, such as stay duration and travel time between POIs, we leave them for future research.

F.3 Prompt for Identifying the Start POI

Prompt for Start POI Identification

```
Please act as a top-tier AI travel planning assistant. Your job is to return the index of the best starting point for a
day trip itinerary based on user needs and provided candidate POIs. If the candidate POIs are empty, please default to
returning ["0"].

### Task Overview
Your task is to return the index of the best starting point for an itinerary based on the given candidate POIs and the
user request. Directly return the starting point's index based on user needs and candidate POIs, without writing any code.

### Candidate POIs
{candidate_strings}

### User Request
{user_reqs}

### Guidelines
1. Ensure the selected POI meets the user request.
2. The starting point should be close to its neighboring points.
3. Prioritize POIs like museums or art galleries, which usually require more exploration time.
4. Avoid starting from bars or clubs.

### Example Inputs and Outputs
- **Example 1**:
- **Candidate POIs**: ["Museum", "Park", "Bar"]
- **Output**: ["0"]

- **Example 2**:
- **Candidate POIs**: ["Shopping Center", "Art Gallery", "Historical Building"]
- **Output**: ["1"]

### Output Specification
1. Return a list of length 1, containing an integer that represents the index of the best starting point.
2. Return as a JSON List, with only one element inside.
3. The output should not contain any other information, ensuring it can be parsed by json.loads.
4. Your response should be a length-1 JSON list, where the index comes from {return_candidates}.
    - Example: ["0"]

Now, based on the **Candidate POIs** and **User Needs**, return the index of the best starting point for the day trip
itinerary according to the **Output Specification**. Note, ensure your reply is **a list composed of a single number**
from {return_candidates}, following the requirements in the **Output Specification** to return **a length-1 JSON list**,
and do not return any other content.
```

F.4 Prompt for Generating the Itinerary

Prompt for Final Itinerary Generation

```
You are a highly creative and knowledgeable tour guide, specifically to design a perfect day trip itinerary.
Please consider carefully and use the provided "Candidate POIs" list to craft a one-day itinerary in the form of an
engaging and realistic travel story.

## Itinerary Information

Next, please follow the guidelines I provide to design a memorable day trip itinerary for tourists.

Design a day trip itinerary for tourists:
- **Order of candidate POIs**: {context_string}
- **Must-see POIs**: {keyword_reqs}
- **Keyword Requirements**: {keyword_reqs}
- **User's Original Request**: {userReqList}
- **Start POI**: {start_poi}
- **End POI**: {start_poi}

## Constraints

- **Itinerary time**: Less than {hours} hours
- **POI selection**: Must follow the given sequence order

## Output Format:
{{
    "itinerary": "List of POIs, separated by '->'"
    "Overall reason": "Overall recommendation reason for the designed day trip itinerary",
    "pois": {{
        "n": "Description and recommendation reason for each POI", ...
    }}
}}

Note:

- "n" is the sequence number, which should be an integer. Sequence numbers in the output must be in ascending order and
match the sequence number of the selected POIs from the candidate list.
- "itinerary" lists all the POIs' names visited, separated by '->', such as "poi1->poi2->...", note that it includes names
only, without sequence numbers, and the order is consistent with the order of POIs in "pois".


## Pre-action Considerations
1. Work on problems step-by-step.
2. Do not omit or simplify anything.
3. Ensure the tourists feel that the itinerary is tailor-made for them.
4. **ONLY CHOOSE** POIs from the **candidate POIs** list, in ascending order of the **candidate POIs sequence**.
5. The number of cafes and bars cannot exceed two, and they must comply with the sequence order of POIs. **Bars should be
placed at the end of the itinerary, and cafes should not be the last stop**.
6. Ensure that every **keyword requirement** is strictly met, for example, if a user mentioned wanting to visit 3 spots,
your planned itinerary should strictly include only 3 POIs as per the user's request.


## Itinerary Creation Steps
1. Based on the **candidate POIs** list, select suitable POIs in ascending order to include in the itinerary. Ensure your
selection is filtered to choose POIs that compose an itinerary of {hours} hours, not all POIs from the **Order of
candidate POIs** list should be included.
2. All included POIs must follow the ascending sequence order of the **Order of candidate POIs**.
3. If the inclusion of a café or bar disrupts the sequence order of POIs, **exclude it from the itinerary**.
4. Ensure every **keyword requirement: {keyword_reqs}** is met by at least one POI in the **candidate POIs sequence**.
5. **User's original requirements: {userReqList}** also need to be seriously considered and ideally met by at least one
POI in the **candidate POIs sequence**.
6. Finally, generate a JSON file containing all selected POIs.

Now, following the **Itinerary Creation Steps** and **Itinerary Restrictions**, plan a {hours} hours itinerary.
```

F.5 Prompt for Baseline

We provide the prompt for the baseline GPT-4 CoT below. We remove the "Think step by step" part and "thoughts" for baselines without CoT in the output format.

Prompt for baseline GPT-4 CoT

```
You are a travel planning assistant. Please help me plan a city tour itinerary in {city} based on requirements. The output
should include specific Points of Interests (POI), and the itinerary should contain no less than 6 POIs:

## User Request
{user_request}

Think step by step: You need to understand and analyze the user's request, including must-see POIs, positive requests,
negative requests, etc., and then provide your recommended POIs and reasons for recommendation.

## Output Format:
{{
    "thoughts": "Your understanding and analysis of user requirements",
    "itinerary": "List of POIs, separated by '->'",
    "overall_reason": "The overall reason for recommending this one-day tour itinerary",
    "pois": {{
        "n": "Description and reason for recommending the POI", ...
    }}
}}
n starts from 1 and increments. Please strictly follow the output format to return JSON.
```

F.6 Prompt for LLM-evaluated Metrics

Prompt for LLM-evaluated metrics is provided below.

Prompt for LLM-evaluated Metrics

```
You are a professional travel assistant. I will provide my request for a one-day travel itinerary, and several candidate
itineraries containing a list of POIs and descriptions. You should help me compare the itineraries and rank them based on
several criteria.

## Criteria
1. POI Quality: how interesting and diverse the POIs are
2. Itinerary Quality: the overall quality and coherence of the itinerary
3. Matchness: the matchness between the itinerary and the user query

## Input Format

Each input candidate itinerary is a dictionary in the following format:

{{
    "itinerary": "a list of POIs, separated by '->'"
    "overall_reason": "The overall recommendation reason for the designed one-day travel itinerary",
    "pois": {{
        "n": "description of the POI", ...
    }}
}}

## Request

{user_request}

## Candidate Itineraries

{itineraries}

## Output Format

Output a json object (dictionary) with four keys: "poi_quality", "itinerary_quality", "matchness", "language_quality".
Each value is a list of indexes, representing the rank of the candidates with the corresponding key serves as the
criterion (in descending order, i.e. from high to low). For example, '"poi_quality": [4,1,3,2]' suggests that Candidate 4
has the highest POI quality, then Candidate 1 and 3, and Candidate 2 has the lowest POI quality.

Ensure that your output can be parsed with Python json.loads. Do not output anything else.
```