ChinaTravel: A Real-World Benchmark for Language Agents in Chinese Travel Planning

Anonymous ACL submission

Abstract

Recent advances in LLMs, particularly in language reasoning and tool integration, have rapidly sparked the real-world development of *Language Agents*. Among these, travel planning represents a prominent domain, combining academic challenges with practical value due to its complexity and market demand. However, existing benchmarks fail to reflect the diverse, real-world requirements crucial for deployment. To address this gap, we introduce *ChinaTravel*, a benchmark specifically designed for authentic Chinese travel planning scenarios. We collect the travel requirements from questionnaires and propose a compositionally generalizable domain-specific language that enables a scalable evaluation process, covering feasibility, constraint satisfaction, and preference comparison. Empirical studies reveal the potential of neuro-symbolic agents in travel planning, achieving a constraint satisfaction rate of 27.9%, significantly surpassing purely neural models at 2.6%. Moreover, we identify key challenges in real-world travel planning deployments, including open language reasoning and unseen concept composition. These findings highlight the significance of ChinaTravel as a pivotal milestone for advancing language agents in complex, real-world planning scenarios.

1 Introduction

A long-standing goal in AI is to build planning agents that are reliable and general, able to assist humans in real-world environments. Recently, Large Language Models (LLMs) (Brown et al., 2020; Ouyang et al., 2022; Achiam et al., 2023) have demonstrated remarkable potential in achieving human-level understanding and planning capabilities. This has sparked the rapid development of a field called *Language Agents*, employing LLMs to perceive the surroundings, reason the solutions, and take appropriate actions, ultimately building an autonomous planning agent (Shinn et al., 2024; Yao et al., 2023; Xi et al., 2023). Equipping LLMs born from web-scale corpora, language agents demonstrate a proficient ability to understand general natural language instructions and collect domain-specific information via tools (Yao et al., 2022; Xie et al., 2023; Jimenez et al., 2024). It alleviates the need for intensive domain-specific goal definition and model deployment with traditional rule-based or reinforcement-learning-based agents, showing few-shot generalization across various domains. This presents a solid step toward the goal of building general artificial intelligence.

Travel planning stands out as a significant domain, presenting both academic challenges and practical value due to its inherent complexity and real-world relevance. However, LLMs are still not able to accurately solve complex combinatorial optimization problems and tend to provide infeasible plans in travel planning. In a recently proposed U.S. domestic benchmark TravelPlanner (Xie et al., 2024) with intercity itinerary planning, the advanced LLM, GPT-4, only achieves a success rate of 0.6%. This result is disappointing and might make one pessimistic about the capabilities of Language Agents in travel planning. However, a few months later, Hao et al. (2024) introduced a neural-symbolic solution, which incorporates formal verification tools into language agents and achieved a 97% success rate on the LLM-synthesized queries from TravelPlanner benchmark. Despite this progress, travel queries posed by humans present significantly greater challenges than synthesized queries. The open-ended expression styles of humans, characterized by ambiguous phrasing and context-dependent meanings, make understanding these requirements difficult for LLMs. Furthermore, the diverse nature of user needs renders constraint verification based on predefined concepts hard to scale, limiting its applicability to evaluating human queries.

In this work, we introduce ChinaTravel, tailored to authentic Chinese travel requirements. It

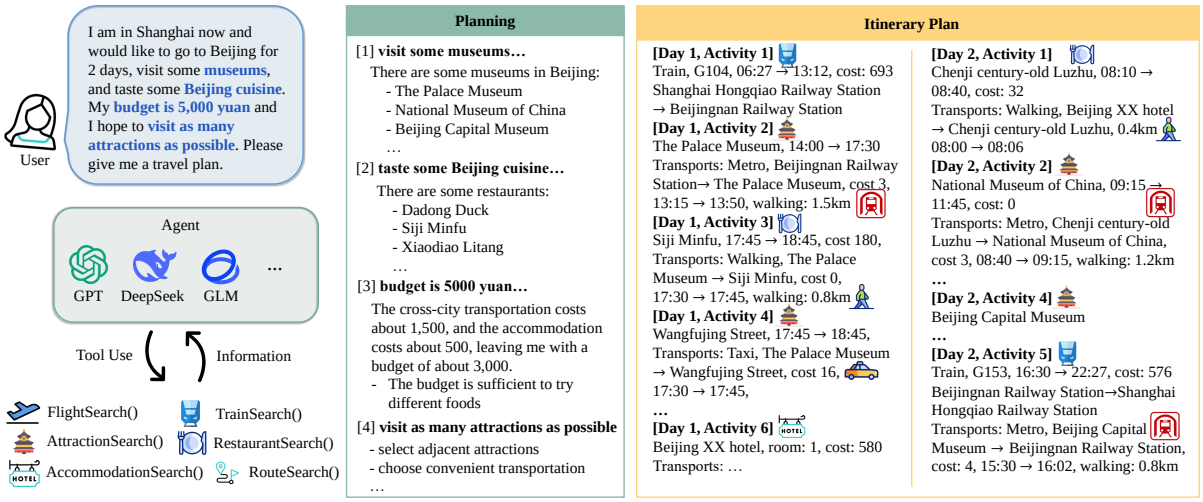


Figure 1: Overview of ChinaTravel. Given a query, language agents employ various tools to gather information and plan a multi-day multi-POI itinerary. The agents are expected to provide a feasible and reasonable plan while satisfying the hard logical constraints and soft preference requirements. To provide convenience for global researchers, we provide an English translation of the original Chinese information here.

concentrates on multi-point-of-interest (multi-POI) itineraries within specified cities (as illustrated in Fig. 1), which are in higher demand compared to the intercity itineraries provided by TravelPlanner. The main contributions of this work are as follows:

1. Comprehensive Evaluation Framework: It provides a sandbox enriched with authentic travel data, a domain-specific language for scalable requirements definition and automated evaluation, and diverse metrics covering feasibility, constraint satisfaction, and preference ranking.

2. Integration of Synthetic and Human Queries: The benchmark includes both LLM-generated and human-derived queries, offering a realistic and open testbed for evaluating agents' capabilities in addressing diverse travel requirements.

3. Empirical Neuro-Symbolic Insights: Our experiments reveal that neuro-symbolic agents significantly outperform pure LLM-based solutions, achieving a constraint satisfaction rate of 27.9% compared to 2.60% by purely neural methods, thus highlighting their promise for travel planning.

4. Identified Challenges for Future Research: We pinpoint key challenges of open-ended requirements: open language reasoning, and unseen concept composition, providing a foundation for advancing agents toward real-world applicability.

Overall, ChinaTravel provides a challenging yet meaningful testbed for evaluating language agents in travel planning, serving as a critical bridge between academic research and practical applications.

2 ChinaTravel Benchmark

Motivated by the significant travel demand in China, this benchmark offers a sandbox environment for generating multi-day, multi-POI itineraries for specified cities. ChinaTravel is designed to serve as a comprehensive and scalable benchmark for evaluating language agents in travel planning, including arrangements for attractions, restaurants, accommodations, and transportation between events.

2.1 Environment Information

ChinaTravel provides a sandbox with real-world travel information. We collect information from 10 of the most popular cities in China. It includes 720 airplanes and 5,770 trains connecting these cities, with records detailing departure and arrival times, origins, destinations, and ticket prices. Additionally, the dataset contains 3,413 attractions, 4,655 restaurants, and 4,124 hotels, each annotated with name, location, opening hours, and per-person prices. Type annotations for these POIs are included to meet user needs. Fig. 2 has demonstrated the travel information from Beijing and Nanjing, two of the most popular cities in China. For a more realistic interaction, we simulate the API interface of real market applications to query real-time information. The detailed designs of the sandbox are available in App. B.1. Environmental constraints act as a feasibility metric, ensuring that the generated plans are both valid and effective. For example, POIs in the plan must exist in the desig-

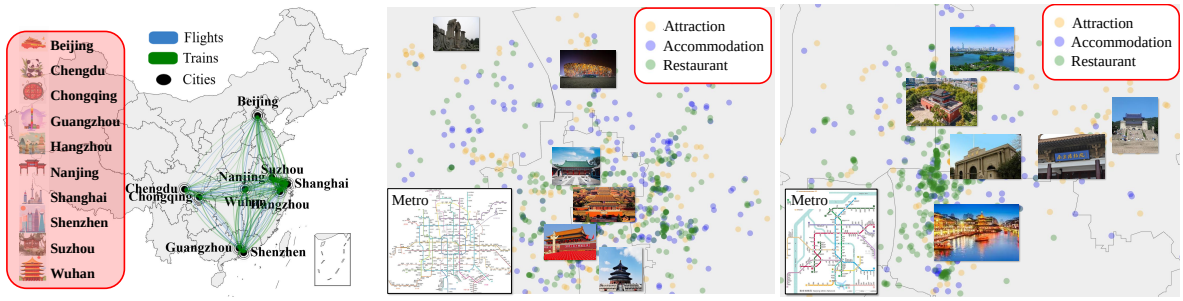


Figure 2: Overview of **ChinaTravel Sandbox Environment**. Our sandbox incorporates travel information from 10 of the most popular cities in China, offering comprehensive information on attractions, accommodations, and restaurants essential for travel planning. Here is the visualization of information from Beijing and Nanjing.

Evaluation Metrics	**Environment Constraints**
Cross-city Transportation	Available Trains or Airplanes across cities.
	Correct information of cost and schedule.
Inner-city Transportation	Available Metro, Taxi or Walking between different positions.
	Correct information of cost, distance and duration
Attractions	Available Attractions in the target city, visiting in their open time.
	Attraction choices should not be repeated throughout the trip.
	Correct information of cost.
Restaurants	Available Restruants in the target city, visiting in their open time.
	Restaurant choices should not be repeated throughout the trip.
	Breakfast, lunch, and dinner are served at their designated meal times.
	Correct information of cost.
Accommodation	Available Accommodation in the target city.
	Room information to meet headcounts.
Time	The given activity events occur in chronological order.
Space	Events at different positions should provide transport information.

Table 1: Descriptions of **Environment Constraints** for two benchmarks. Constraints in black are common in both TravelPlanner and ChinaTravel. Metrics in brown are the metrics only in our benchmark.

nated city, transportation options must be viable, and time information must remain accurate. Tab. 1 summarizes the environmental constraints.

2.2 Logical Constraint

A crucial ability for travel planning is to effectively satisfy personalized user needs. We extend the logical constraints from TravelPlanner (Xie et al., 2024) and present a Domain-Specific Language (DSL) to support general reasoning in logical constraints. ChinaTravel's DSL is a general set of pre-defined concept functions with built-in implementations and is listed in Tab. 2. TravelPlanner relies on 5 pre-defined concepts {total budget, room rules, room types, cuisines, and transportation types}, to evaluate the logical constraints, where each concept is equivalent to a specific logical requirement. We find that this approach limits the ability to validate diverse logical needs in an open-world context. For example, such an evaluation cannot express that the dining expenses should be within 1000 yuan or that arriving in Shanghai should be before 6 PM on the second day, despite the generated plan already including the expenses for each activity and time information of the return flight. Each new logical requirement necessitates human intervention for definition. To address this issue, our approach is grounded in a DSL-based solution that leverages basic concept functions and syntax to express and fulfill various logical requirements.

```
# Dining expenses <= 1000 CNY.
dining_cost = 0
for act_i in allactivities(plan):
  typ = activity_type(act_i)
  if typ=="breakfast" or typ=="lunch" or
       typ=="dinner": dining_cost =
      dining_cost + activity_cost(act_i)
return dining_cost <= 1000
```

Name	Syntax	Description
variables	$x, y, z, \cdots$	Variables that refer to activities in the travel planning domain.
not	$not\ expr$	The negation of an Boolean-valued expression.
and,or	$expr_1\ and\ expr_2$	The conjunction/disjunction of an Boolean-valued expression.
$<, >, ==$	$expr_1 < expr_2$	Return an expression with built-in number comparison functions.
$+, -, *, /$	$expr_1 + expr_2$	Return an expression with built-in number calculation functions.
attributes	$cost(var)$	A function that takes activities as inputs and returns the attributes, such as cost, type or time.
relation	$dist(expr_1, expr_2)$	A function that takes locations as inputs and returns the distance.
effect	$var = expr$	An assignment affects a variable var with the expression $expr$.
union, inter, diff	$uni(\{var\}_1, \{var\}_2)$	Return a set with the built-in union/intersection/difference operations of given two sets.
enumerate	$for\ var\ in\ \{var\}$	Enumerate all variables in the collection $\{var\}$.
when	$if\ expr : effect$	The conditional effect takes a Boolean-valued condition of the expression $expr$, and the effect $effect$.

Table 2: ChinaTravel's Domain-Specific Language (DSL) for logical constraints.

```
# Arriving in Shanghai should be before
    6 PM on the second day.
return_time = 0
for act_i in day_activities(plan, 2):
  typ = activity_type(act_i)
  dest = transport_destination(act_i)
  if (typ=="train" or typ=="airplane")
      and des=="Shanghai": return_time
      == activity_endtime(act_i)
return return_time < "18:00"
```

The DSL can represent varying requirements through concept composition in a Python format, and perform automated validation of plans using a Python compiler. This strategy maximizes the evaluation capability of the ChinaTravel benchmark. The App. B.2 provides a more detailed definition and implementation of concept functions.

2.3 Preference Requirement

Travel requirements encompass not only hard logical constraints but also soft preferences. The term "soft" implies that these preferences cannot be addressed as boolean constraint satisfaction problems, instead, they involve quantitative comparisons based on continuous values. This distinction highlights the unique nature of preference-based requirements compared to logical constraints. Common preferences identified through surveys include maximizing the number of attractions visited, minimizing travel time between destinations, and visiting positions near the specific POI, among others. In ChinaTravel, we formalize such preferences as minimization or maximization objectives via our DSL, thereby providing an automated evaluation.

```
# The number of attractions visited
count = 0
for act_i in all_activities(plan):
  if activity_type(act_i)=="attraction":
      count = count + 1
return count
```

2.4 Benchmark Construction

ChinaTravel provides user queries reflecting diverse requirements through a four-stage process that integrates LLM-based generation with questionnaires.

Stage I: Manual design of database and APIs. We collect travel information for multi-day, multi-POI itineraries across attractions, accommodations, and transportation. We define essential POI features, such as cuisine types and hotel characteristics, to construct the database from public information. APIs are designed to support agent queries via regular expressions and modeled after commercial APIs to ensure realism. See App. B.1 for details.

Stage II: Automatic data generation with LLMs. We define common travel information (e.g., origin, destination, days, number of people) and logical constraints to model travel tasks. To enable scalable queries, query skeletons are randomly constructed from this information and transformed into natural language queries using advanced LLMs. The generated queries are categorized into two difficulty levels: *Easy*, with 1 logical requirement beyond basic constraints like people number and trip duration, and *Medium*, with 3–5 additional logical requirements. We encourage the LLM to generate

diverse, human-like expressions, such as turning "Taste Beijing cuisine" into "Try local food in Beijing." See App. B.3 for an example snippet and more details.

Stage III: Quality control and auto-validation. To ensure data quality, we manually check whether the generated queries conform to symbolic skeletons, and re-calibrate natural language descriptions that contain ambiguities. Based on the symbolic skeletons of queries, we could verify whether the plan can pass the required logical constraints by executing the DSL code via Python compiler. Building on this, we ensure that each query has at least one solution that satisfies the logical constraints by implementing a heuristic search algorithm.

Stage IV: Open requirements from humans. After the first round of closed-loop development with LLM, including data generation and annotation, baseline development, and evaluation, we further collected travel requirements from more than 250 humans through questionnaires. Based on a new round of quality control on these data, a more challenging set with 154 queries is constructed. These queries even include unseen logical constraints in the deployment process, such as 'departure time' and 'dining cost', reflecting the real challenges of neural-symbolic systems in travel planning. We carefully annotate the required logical constraints for each query based on the DSL, enabling the automated evaluation of these challenging samples and forming the *Human* level dataset.

To support global research on travel planning, we provide an English version of all queries in ChinaTravel. However, we recommend that researchers primarily use the Chinese version, as it better captures the expression from native speakers.

3 Empirical Study

LLMs. We test both state-of-the-art proprietary and open LLMs: OpenAI GPT-4o, DeepSeek-V2.5, as well as Qwen-2.5-7B (Bai et al., 2023). The first two models are chosen for their strong performance, while the latter is selected for their Chinese language capabilities and ability to perform inference with limited local computational resources.

Metrics. We examine the Delivery Rate (DR), Environmental Pass Rate (CPR), Logical Pass Rate (LPR), and Final Pass Rate (FPR) from TravelPlan (Xie et al., 2024). Furthermore, we design a novel metric, Conditional Logical Pass Rate (C-LPR), evaluating the success rate of plans that first fulfill environmental constraints prior to logical constraints. It ensures that logical requirements are met within a realistic travel context, eliminating cases where unrealistic or incorrect information might lead to shortcutting logical constraints, such as misreporting costs to fit budget requirements. By introducing C-LPR, we aim to enhance the feasibility and meaningfulness of constraint satisfaction.

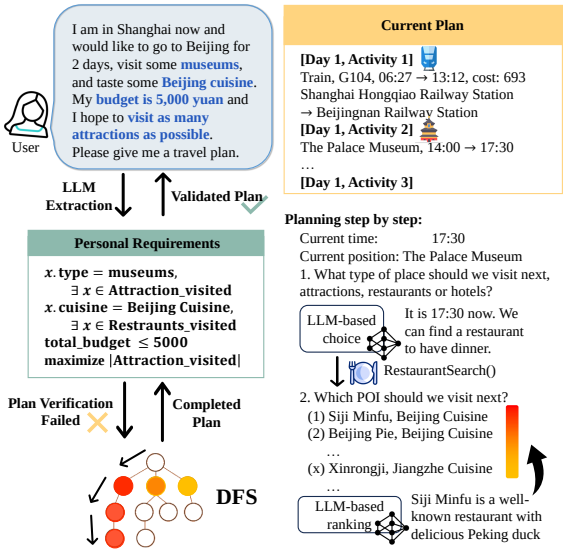


Figure 3: NeSy Planning with depth-first-search solver.

$$C\text{-}LPR = \frac{\sum_{p \in P} \mathbb{1}_{passed(Env,p)} \cdot \sum_{c \in C_p} \mathbb{1}_{passed(C_p,p)}}{\sum_{p \in P} \in |C_p|}$$

P is the plan set, C_p is the set of constraints for plan p, and passed(c, p) indicates whether p satisfies c.

Methods. We evaluate the performance of both pure-LLM-based and neuro-symbolic solutions on the ChinaTravel benchmark. For the former, we primarily test the well-known method, ReAct (Yao et al., 2023), and its Act-only ablation. We exclude Reflexion (Shinn et al., 2024) due to its performance being similar to ReAct on the TravelPlanner (Xie et al., 2024) and the high economic overhead associated with the larger input token size. For the latter, we adapt existing neuro-symbolic pipelines (Hao et al., 2024; Pan et al., 2023; Deng et al., 2024) using our proposed DSL to handle the complexities of multi-day, multi-POI itineraries.

3.1 Neuro-Symbolic Planning

This subsection presents a neuro-symbolic solution as a preliminary baseline for ChinaTravel. This solution consists of two stages. Stage 1: *NL2DSL translation* translates natural language queries into

	LLMs	DR	EPR		LPR		C-LPR	FPR
			Micro	Macro	Micro	Macro		
***Easy* (#300)**								
Act		70.4	49.9	0	64.6	30.8	0	0
		97.5	70.8	0.0	86.8	68.8	0.0	0.0
ReAct		43.3	40.8	0.0	41.9	19.6	0.0	0.0
		95.4	48.2	0.0	71.3	32.9	0.0	0.0
ReAct (one-shot)		77.5	68.3	6.25	74.1	52.5	5.77	5.42
		94.2	68.1	0	**89.4**	**70.8**	0	0
NeSy Planning		78.6	**75.9**	50.6	79.7	64.6	48.6	48.0
		75.0	73.6	**64.0**	73.5	63.3	**61.7**	**60.6**
		72.3	67.0	34.0	70.4	49.6	32.6	28.3
NeSy Planning* (Oracle Translation)		**82.6**	**81.7**	**75.0**	**82.2**	**75.3**	**75.0**	**74.0**
		66.6	66.7	66.0	64.6	63.6	64.6	62.6
		69.3	69.3	59.3	70.2	59.6	59.3	57.9
***Medium* (#150)**								
Act		72.7	52.3	0	63.5	15.3	0	0
		97.4	70.5	0	89.3	55.3	0	0
ReAct		41.3	35.2	0	37.6	4.0	0	0
		92.0	54.8	0	78.6	22.7	0	0
ReAct (one-shot)		82.7	**77.1**	3.33	82.6	48.7	2.95	1.33
		94.7	69.2	0.67	**91.8**	**64.0**	0.53	0
NeSy Planning		71.3	71.9	**69.3**	69.4	50.0	**69.3**	**46.7**
		68.0	68.0	68.0	64.1	46.6	64.1	**46.7**
		53.3	45.9	16.0	49.2	33.3	14.8	8.50
NeSy Planning* (Oracle Translation)		**68.6**	**65.4**	54.0	**66.2**	**61.3**	52.5	54.0
		60.8	59.4	**54.9**	60.3	58.2	**60.3**	**56.9**
		53.3	51.3	36.6	51.9	43.3	34.8	34.6
***Human* (#154)**								
ReAct		36.4	29.5	0.65	35.2	16.2	0.38	0
		96.1	50.5	0	**72.4**	32.5	0	0
ReAct (one-shot)		55.2	**57.3**	2.60	64.6	44.2	1.71	2.60
		69.5	46.3	0	63.6	46.8	0	0
NeSy Planning		45.4	46.6	40.9	40.9	**33.1**	35.3	**27.9**
		45.4	50.1	**45.4**	40.9	29.8	**38.5**	**27.9**
		42.8	47.4	42.2	36.2	27.2	34.4	25.3
NeSy Planning* (Oracle Translation)		50.6	**48.9**	36.3	45.9	40.2	32.0	35.0
		52.6	46.9	**42.9**	**47.6**	**40.9**	**43.9**	**40.9**
		41.5	41.1	31.1	36.5	33.7	25.0	28.5

Table 3: Main results of different LLMs and planning strategies on the ChinaTravel benchmark. LLMs: : DeepSeek-V2.5, : GPT-4o-2024-08-06, :Qwen2.5-7B.

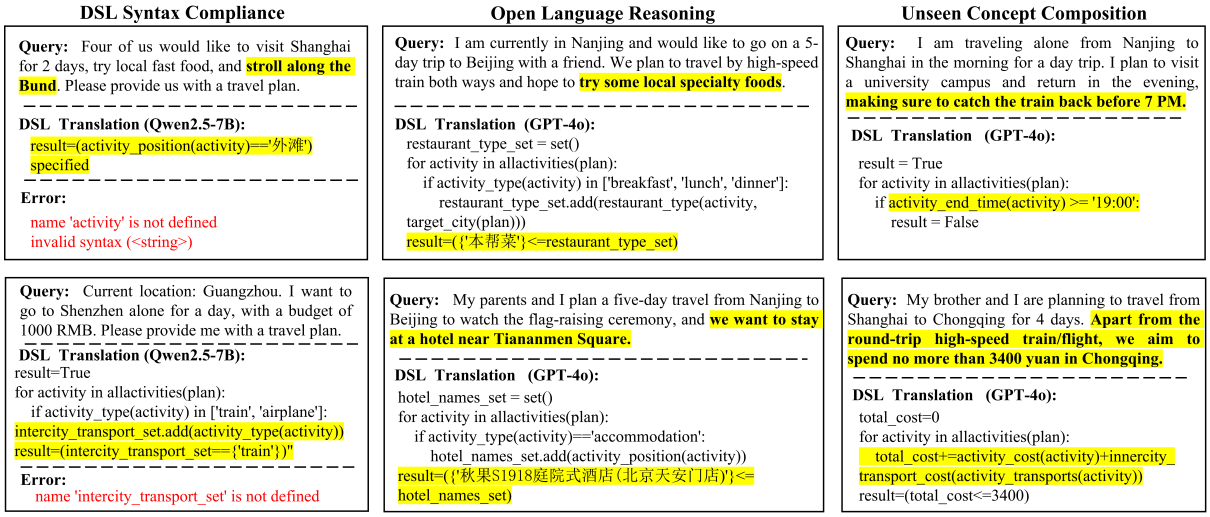


Figure 4: Challenges in the Neuro-Symbolic Planning.

logical, preference-based DSL requirements. We use Reflexion (Shinn et al., 2024) and a DSL syntax checker to iteratively assist the LLM (5 rounds in experiments). Stage 2: *Interactive search* uses a neuro-symbolic solver to sequentially arrange activities, guided by a symbolic sketch and LLM-driven POI recommendations, generating a multi-day itinerary with DSL validation. If constraints are violated, the process backtracks until a feasible solution is found. To ensure fairness, the symbolic sketch search is limited to 5 minutes per query, excluding LLM inference time. To observe the performance across the two stages, we also evaluated the planning results based on the Oracle DSL. App. C includes pseudo-code and LLM prompts.

3.2 Main Results

Based on the results presented in Table 3, we have the following observations and analyses:

Pure LLMs struggle in ChinaTravel. The DR evaluates an agent's ability to generate valid JSON plans (see Fig. 1). While high DRs indicate that advanced LLMs can produce structured outputs for travel planning, the near-zero EPR (Environmental Constraints Pass Rate) reveals their inability to gather and strictly adhere to required information. The sole exception is the DeepSeek model, which achieves the 5% EPR and 4.33% FPR, likely due to its strong capability to follow Chinese requirements. ReAct (one-shot, GPT-4o) excels in Macro LPR but achieves no FPR, suggesting it circumvents constraints via shortcuts. Our proposed C-LPR metric offers a more reliable measure of logical constraints, serving as a supplement to FPR.

Nesy Planning provides a promising solution.

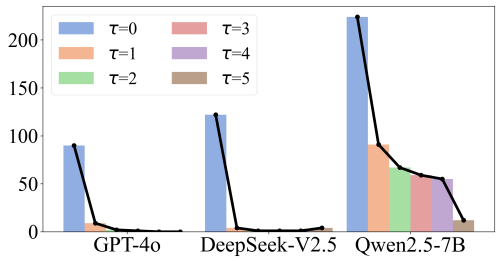


Figure 5: Syntax errors across reflexion rounds τ.

Our NeSy Planning framework integrates symbolic programs to orchestrate travel planning and tool management while utilizing LLMs to extract language-based requirements and prioritize POIs. By separating planning (flexible natural language handling) from grounding (precise execution), the framework enhances adaptability and ensures compliance with constraints. Across all data subsets, NeSy methods outperform pure-LLM approaches. With GPT-4o as the backend, it achieves FPRs of 60.6%, 46.6%, and 27.9% on three subsets, highlighting the effectiveness of NeSy solutions for travel planning with complex constraints.

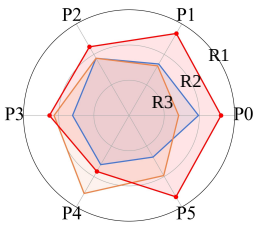
Challenges Persist for Nesy Planning. The performance gap between standard and oracle modes underscores the importance of DSL translation in NeSy planning. Inadequate translations may result in plan searches failing to meet user requirements, while incorrect translations can misguide the search, making feasible solutions unattainable. Among the three LLMs, GPT-4o performs the best, with minimal gaps between modes, indicating its relatively accurate DSL generation effectively supports the search process. We conclude with three challenges and provide the corresponding cases in the Fig. 4.

(1) DSL Syntax Compliance: As shown in Fig. 5, while the reflexion process with syntax checker significantly reduces syntax errors, the Qwen-7B model demonstrates weaker compliance than GPT-4o and DeepSeek, directly resulting in its lower performance in the Tab. 3. **(2) Open Language Reasoning**: Although GPT-4o exhibits relatively fewer syntax errors in translation, it still struggles with diverse queries and context-dependent meanings. For instance, when a user requests "local cuisine," GPT-4o maps it to 本帮菜, ignoring the logical connection that in Beijing, it should align with 北京菜. **(3) Unseen Concept Composition**: Real-world requirements derived from human data are inherently diverse and complex, making expecting models to encounter all possible needs during development impractical. A more feasible way is to emulate human reasoning by generalizing existing knowledge to novel problems. Based on our DSL design, LLMs can express new logical requirements through combinations of concept functions. However, compositional reasoning remains a challenge. For example, GPT-4o misinterpreted a return time constraint as applying to all activities instead of correctly limiting only the return train's departure time to before 19:00.

In summary, ChinaTravel poses significant challenges for current agents. Neuro-symbolic agents outperform pure-LLM approaches in constraint satisfaction, showing strong potential for real-world travel planning. With realistic queries and a versatile DSL for constraint validation, we highlight the critical challenges while providing a foundation for advancing neuro-symbolic systems in practice.

3.3 Ablation Study with Preference

The comparison of preferences should be conducted under the premise that both environmental and logical constraints are satisfied. Given the limited FPR achieved by existing methods on the challenging ChinaTravel, we perform a separate analysis of preference optimization in this section. Specifically, we sampled 50 queries from the easy subset that NeSy-DeepSeek-Oracle successfully passed as seed samples. Based on these, six subsets were created by introducing common preferences identified from user surveys. Three comparative scenarios were designed to explore the roles of LLMs and symbolic search in optimizing preferences during NeSy Planning: (1) Baseline Query (BQ): Results obtained by directly querying the seed samples without preference requirements. (2) Preference-Enhanced Query



NeSy Planning	BQ	PEQ	PDS
Daily attractions ↑	0.75	0.79	**1.63**
Transport time ↓	38.5	41.9	**26.5**
Transport time to rest. ↓	**22.2**	26.8	22.8
Food cost ratio ↑	0.19	0.29	**0.32**
Hotel cost ↓	1350	559	**519**
Distance to POI ↓	30.9	30.6	**26.1**

Figure 6: Ablation on preference ranking.

(PEQ): Results based on seed samples augmented with natural language preference expressions (e.g., "visit more attractions"), evaluating whether embedding preferences into POI recommendations via LLMs improves outcomes. (3) Preference-Driven Search (PDS): Results using both natural language and DSL-based expressions, where the agent, within the 5-minute search time limit, computes the preference concept for solutions that pass environmental and logical constraints and retains plans that maximize or minimize the preference objective. The results are provided in Fig. 6.

From the results(Fig. 6, where ↑ indicates maximization), PEQ outperforms BQ in preference optimization. This ablation demonstrates that LLMs can effectively capture natural language needs during the POI ranking stage, contributing to preference improvements. However, on P2, PEQ underperforms BQ, indicating that LLMs can sometimes have a negative impact. This may be due to the complexity of the preference in P2, which involves minimizing transport time to restaurants, leading to misinterpretation. PDS achieves more significant improvements in preference optimization, relying on DSL-based preference calculations that filter plans more effectively over extended search times. This supports the scalability of DSL in preference optimization but also highlights the pressing need for more efficient algorithms.

4 Conclusion

We present ChinaTravel, a benchmark for multi-day multi-POI travel planning focused on authentic Chinese needs. We address the limitations of previous benchmarks by incorporating open-ended and diverse human queries, capturing real-world user needs. Additionally, we propose a scalable evaluation framework based on DSL, enabling comprehensive assessments of feasibility, constraint satisfaction, and preference comparison. These advancements provide a foundation for developing language agents capable of meeting diverse user requirements and delivering reliable travel solutions.

5 Limitations

Our research represents a significant step forward in evaluating the travel planning capabilities of language agents, but it is not without challenges. One limitation lies in its focus on Chinese travel planning. Due to the inherent differences in natural language, the translated versions of queries may fail to fully capture the challenges of understanding requirements in Chinese queries, potentially limiting its applicability in a global context. However, given the substantial demand within China's travel market, we believe a benchmark tailored to Chinese travel planning is both necessary and socially valuable. Although our benchmark is comprehensive, it may not encompass the full range of requirements encountered in real-world scenarios. The high cost of collecting authentic data has limited the number of human queries in our study. To address this, future work will focus on combining LLMs with real user queries to automate the generation of a wider variety of human-like queries. Continuous refinement and expansion of our benchmark are crucial for more accurately reflecting the realistic travel planning needs.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, pages 1877–1901.

Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. 2002. Deep blue. *Artificial intelligence*, 134(1-2):57–83.

Aili Chen, Xuyang Ge, Ziquan Fu, Yanghua Xiao, and Jiangjie Chen. 2024. TravelAgent: An AI assistant for personalized travel planning. *arXiv preprint arXiv:2409.08069*.

Wang-Zhou Dai, Qiu-Ling Xu, Yang Yu, and Zhi-Hua Zhou. 2019. Bridging machine learning and logical reasoning by abductive learning. In *Advances in Neural Information Processing Systems*, pages 2811–2822.

Shujie Deng, Honghua Dong, and Xujie Si. 2024. Enhancing and evaluating logical reasoning abilities of large language models. In *Proceedings of the ICLR 2024 Workshop on Secure and Trustworthy Large Language Models*.

Tanmay Gupta and Aniruddha Kembhavi. 2023. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962.

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.

Yilun Hao, Yongchao Chen, Yang Zhang, and Chuchu Fan. 2024. Large language models can plan your travels rigorously with formal verification tools. *CoRR*, abs/2404.11891.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *Proceedings of the 12th International Conference on Learning Representations*.

Weiyu Liu, Geng Chen, Joy Hsu, Jiayuan Mao, and Jiajun Wu. 2024. Learning planning abstractions from language. In *Proceedings of the 12th International Conference on Learning Representations*.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. 2018. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems*, pages 3753–3763.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, pages 27730–27744.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 3806–3824.

Vibhor Sharma, Monika Goyal, and Drishti Malik. 2017. An intelligent behaviour shown by chatbot system. *International Journal of New Technology and Research*, 3(4):263312.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359.

Yihong Tang, Zhaokai Wang, Ao Qu, Yihao Yan, Kebing Hou, Dingyi Zhuang, Xiaotong Guo, Jinhua Zhao, Zhan Zhao, and Wei Ma. 2024. Synergizing spatial optimization with large language models for open-domain urban itinerary planning. *CoRR*, abs/2402.07204.

Po-Wei Wang, Priya L. Donti, Bryan Wilder, and J. Zico Kolter. 2019. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6545–6554.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. 2023. The rise and potential of large language model based agents: A survey. *CoRR*, abs/2309.07864.

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

Tianbao Xie, Fan Zhou, Zhoujun Cheng, Peng Shi, Luoxuan Weng, Yitao Liu, Toh Jing Hua, Junning Zhao, Qian Liu, Che Liu, Leo Z. Liu, Yiheng Xu, Hongjin Su, Dongchan Shin, Caiming Xiong, and Tao Yu. 2023. Openagents: An open platform for language agents in the wild. *CoRR*, abs/2310.10634.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, pages 20744–20757.

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 *Proceedings of the 11th International Conference on Learning Representations*.

Hongbo Zhang, Junying Chen, Feng Jiang, Fei Yu, Zhihong Chen, Guiming Chen, Jianquan Li, Xiangbo Wu, Zhiyi Zhang, Qingying Xiao, Xiang Wan, Benyou Wang, and Haizhou Li. 2023. Huatuogpt, towards taming language model to be a doctor. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 10859–10885.

A Discussion with Related Work

LLM-based Agents have demonstrated significant capability in understanding complex instructions and employing domain-specific tools to complete tasks, showcasing their potential in fields such as visual reasoning (Gupta and Kembhavi, 2023), healthcare (Zhang et al., 2023) and robotics (Liu et al., 2024). This reduces the reliance of previous agents on domain-specific efforts, that is, either mainly following domain-specific rules to plan (rule-based agents, such as DeepBlue (Campbell et al., 2002) and Eliza (Sharma et al., 2017)) or mainly learning from domain-specific data to plan (reinforcement-learning-based agents, such as AlphaGo (Silver et al., 2017) and Atari DQN (Mnih et al., 2013)). While the language agents have shown promising results in some domains, most of their planning scenarios are limited to simple tasks with single objective function and fail in the travel planning benchmark with complex logical constraints on the results.

Neuro-Symbolic Learning explores to combine traditional symbolic reasoning with learning to enhance the reliability (Manhaeve et al., 2018; Wang et al., 2019; Dai et al., 2019). In the era of large language models, Pan et al. (2023) presents the LogicLM integrates LLMs with separate symbolic solvers for various logical reasoning tasks. They

first utilize LLMs to translate a natural language problem into a symbolic formulation. Afterward, a deterministic symbolic solver performs inference on the formulated problem to ensure the correctness of the results. Deng et al. (2024) supplement LogicLM with a Self-Refinement Module to enhance the reliability of LLM translation. In the travel planning domain, Hao et al. (2024) presents a framework with a similar pipeline. It first extracts the logical constraints from natural language queries and then formalizes them into SMT code. Thanks to SMT solvers being sound and complete, this neuro-symbolic solution guarantees the generated plans are correct and has basically solved the TravelPlanner benchmark with a 97% pass rate.

Travel Planning is a time-consuming task even for humans, encompassing travel-related information gathering, POI selection, route mapping, and customization to meet diverse user needs (Halder et al., 2024). Natural languages are one of the most common ways for users to express their travel requirements. However, the ambiguity and complexity of travel requirements make it still challenging for LLMs to generate accurate and reliable travel plans. Xie et al. (2024) presents the TravelPlanner benchmark for cross-city travel planning and reveals the inadequacies of pure-LLM-driven agents. TravelPlanner generates user queries through LLMs and provides a rigorous evaluation mechanism to verify whether the provided plans can meet the logical constraints in the queries. It has become a pivotal benchmark for language agents in real-world travel planning. Tang et al. (2024) study the open-domain urban itinerary planning where a single-day multi-POI plan is required. They integrates spatial optimization with large language models and present a system ITTNERA, to provide customized urban itineraries based on user needs. A concurrent work, TravelAgent (Chen et al., 2024), also considers a multi-day multi-POI travel planning problem for the specified city. It constructs an LLM-powered system to provide personalized plans. However, due to the high cost of collecting and annotating real travel needs, they evaluate the proposed TravelAgent in only 20 queries. This also demonstrates the necessity of introducing a new benchmark for travel planning.

B Detailed Design of ChinaTravel

B.1 Sandbox Information

We started collecting travel information with the motivation of planning a multi-day, multi-POI itinerary in four aspects: attractions, accommodation, activities, and transportation. Developers first determine the POI description information that needs to be obtained from the user's perspective, such as cuisine and hotel features. Based on this feature set, we collect public information to construct the database. For the design of APIs, we directly support queries based on the regular expressions from agents. At the same time, we expect the design of APIs to have similar features and characteristics to existing commercial APIs, enabling our dataset to be applicable to more realistic scenarios. The information our database contains is shown in Table 4 and the APIs we offer is in Table 5

B.2 Concept Function

We defined 35 concept functions. Their definition and implementation is in Table 6, 7, 8 and 9.

B.3 Query Synthesis

We designed common travel information (origin, destination, days, number of people) and logical constraints based on the nature of travel tasks. To facilitate scalable queries for ChinaTravel, we randomly constructed query skeletons from the aforementioned information and used advanced LLMs to generate natural language queries from these skeletons. The automatically generated data is categorized into two difficulty levels: In the *Easy* level, user inputs encompass a single logical requirement, sourced from categories such as transportation, restaurants, attractions, and accommodations. In the *Medium* level, user inputs involve 2 to 5 logical requirements, introducing more complex constraints. During the generation, we encourage the LLMs to provide varied and human-like expressions, necessitating a deeper understanding and processing to accurately interpret and fulfill the user's needs. For instance, the logical requirement "taste Beijing cuisine" could correspond to the natural language query: "Try local food in Beijing." We utilize prompt engineering to guide LLMs in refining natural language expressions to facilitate automated generation. One of the prompts is shown in Figure 7. Several examples of generated data is in Figure 8.

Tool	Information
Attractions	Name, Type, Latitude, Longitude, Opentime, Endtime, Price, Recommendmintime, Recommendmaxtime
Accommodations	Name, Name_en, Featurehoteltype, Latitude, Longitude, Price, Numbed
Restaurants	Name, Latitude, Longitude, Price, Cuisinetype, Opentime, Endtime, Recommendedfood
Transportation	Transportation in specific city including walk, metro and taxi
IntercityTransport	Flight: FlightID, From, To, BeginTime, EndTime, Duration, Cost Train: TrainID, TrainType, From, To, BeginTime, EndTime, Duration, Cost
Poi	Names of POIs(including intercity transportation hub) and their coordinates

Table 4: Sandbox Information

C NeSy Planning

Since the Z3 solver from (Hao et al., 2024) would restructure the tool API to return travel information expressed in specific Z3 variables, which may not be feasible given that APIs in the real world are typically black boxes that agents can only call. Following their two-stage solution, we first extract logical constraints from natural language. Based on these constraints, we implement a step-by-step plan generation process using depth-first search, mimicking how humans plan to travel by arranging activities one by one. As shown in Fig. 3, we first translate the natural languages to logical constraints through prompting. generate the next activity type based on the current plan, and then recursively generate the next activity until the goal is reached. The generated plan is then used to solve the problem. In the second step, we define the rule-based activity selection and score function. For example, if the current time is in the [10:30, 12:30] and there is no scheduled lunch in the current plan, then the agent should find a restaurant to have lunch at this time. If the current time is after 22:00 and there are no open-time attractions nearby, the agent should choose to return to the hotel. For the score function, we select the restaurants that satisfy the required cuisine and sort the candidates by the price if there a budget constraints in the constraints C. These ranking functions will help us to find a feasible solution as soon as possible. In ChinaTravel, the duration arrangement of activities is continuous and difficult to enumerate and search. We pre-define a meal or a visit to an attraction as 90 minutes, and when there are less than 90 minutes until closing time, the event continues until the closing time. Given these designs, we adapt the neural-symbolic solution into a multi-POI planning problem and evaluate it in the ChinaTravel benchmark.

Given that some queries are particularly challenging due to the limited number of feasible plans, we set the maximum runtime for the symbolic sketch from interactive search to 5 minutes per query, excluding the LLM inference time, to ensure a fair comparison across different models. If a plan satisfying the generated DSL validation is found within the time limit, it is returned directly. Otherwise, the program halts when the time limit is reached, and the plan that satisfies environmental constraints while achieving the highest number of validation code successes among all intermediate results is returned. In cases where no environment-compliant plan is identified, the partially completed plan generated up to that point is returned.

In the Figure 9, 10 and 11, we provide the prompts of the LLM POI-ranking phases.

Algorithm 1 Depth-First Greedy Search

Require: Constraints C, current plan p,
  if the least activity is an intercity-transport from destination to origin **then**
    return ConstraintValidation(p, C), p ▷ The plan p is finished, return the validation result.
  end if
  type = GetNextActivityType(p) ▷ Select the next type of activities, e.g. lunch, attraction.
  candidates = ToolUse(type) ▷ Collect the corresponding information for the activity type
  scores = LLMScore(candidates, p, C) ▷ Score candidates through constraints C.
  for activity in candidates **do**
    p.push(activity) ▷ Perform a greedy search with priority ranking.
    flag, p = Depth-FirstGreedySearch(C, p)
    if flag **then**
      return True, p ▷ Return the solution p if the validation is passed.
    end if
    p.pop(activity)
  end for
  return False, p ▷ Fail to find a solution with the given conditions.

Tool	API	Docs
Attractions	attractions_keys(city)	Return a list of (key, type) pairs of the attractions data.
	attractions_select(city, key, func)	Return a DataFrame with data filtered by the specified key with the specified function.
	attractions_id_is_open(city, id, time)	Return whether the attraction with the specified ID is open at the specified time.
	attractions_nearby(city, point, topk, dist)	Return the top K attractions within the specified distance of the location.
	attractions_types	Return a list of unique attraction types.
Accommodations	accommodations_keys(city)	Return a list of (key, type) pairs of the accommodations data.
	accommodations_select(city, key, func)	Return a DataFrame with data filtered by the specified key with the specified function.
	accommodations_nearby(city, point, topk, dist)	Return the top K accommodations within the specified distance of the location.
Restaurants	restaurants_keys(city)	Return a list of (key, type) pairs of the restaurants data.
	restaurants_select(city, key, func)	Return a DataFrame with data filtered by the specified key with the specified function.
	restaurants_id_is_open(city, id, time)	Return whether the restaurant with the specified ID is open at the specified time.
	restaurants_nearby(city, point, topk, dist)	Return the top K restaurants within the specified distance of the location.
	restaurants_with_recommended_food(city, food)	Return all restaurants with the specified food in their recommended dishes.
	restaurants_cuisine(city)	Return a list of unique restaurant cuisines.
Transportation	goto(city, start, end, start_time, transport_type)	Return a list of transportation options between two locations with the specified departure time and transportation mode.
IntercityTransport	intercity_transport_select(start_city, end_city, intercity_type, earliest_leave_time)	Return the intercity transportation information between two cities.
Others	notedown(description, content)	Write the specified content to the notebook
	plan(query)	Generates a plan based on the notebook content and query and report the plan is done.
	next_page()	Get the next page of the latest Result history if it exists. Because of the length limited, all returned DataFrame information is split into 10 rows per page.

Table 5: APIs

An Example of Prompts for Data Generation

你是一个用户，你想请ai制定一个旅行规划，请根据以下的例子构建一些自然语言的询问，并提供对应的逻辑约束表达。注意tickets和people_number一样。
例子：
JSON：
{
"start_city": "北京",
"target_city": "南京",
"hard_logic": [
"days==2",
"people_number==1",
"tickets==1",
"{'南京大排档'} <= restaurant_names",
],
"nature_language": "当前位置北京。我一个人想去南京玩2天，想吃南京大排档，请给我一个旅行规划。"
}
使用如下的餐饮。
店名：{}
即要求restaurant_names包含这个店。
注意，餐饮不一定完全按照提供的特征的名字来，可以使用近义词，比如如果提供的是泳池，可以使用想在酒店游泳这样的自然语言询问
注意，你现在的出发地点为{},目标地点为{}。人数{},天数{}
现在请给一个json询问,
JSON：

```
# You are a user who wants to ask an AI agent to help you plan a
    trip. Please construct some natural language inquiries based
    on the following example and provide the corresponding
   logical constraint expressions. Note that "tickets" and "
   people_number" are the same.
# Example:
# JSON:
# {}
# Use the following restaurants.
# Restaurant name: {}
# This means that "restaurant_names" should include this
   restaurant.
# The dining options may not always be exactly as described by
   the provided features; synonyms can be used. For example, if
   the hotel's feature is a pool, you could ask naturally in
   language like "I want to swim in the hotel pool."
# Now, your departure location is {}, and your destination is
   {}. The number of people is {}, and the number of days is {}.
# Now please provide a JSON inquiry.
# JSON:
```

Figure 7: An example of prompts for data generation. This example is about restaurant_name. By replacing this with other constraints or combining multiple constraints, we can generate data with different levels of difficulty based on different constraints.

Examples of Generated Data

Example 1

```
{
    "start_city": "杭州",
    "target_city": "上海",
    "hard_logic": [
        "days==2",
        "people_number==1",
        "tickets==1",
        "{'本帮菜'} <= food_type"
    ],
    "nature_language": "当前位置杭州。我一个人想去上海玩2天，想尝试当地的特色菜，请给我一个旅行规划。"
}
```

Example 2

```
{
    "start_city": "深圳",
    "target_city": "北京",
    "hard_logic": [
        "days==2",
        "people_number==3",
        "intercity_transport=={'airplane'}",
        "tickets==3",
        "rooms==3",
        "room_type==1"
    ],
    "nature_language": "当前位置深圳。我们三个人计划去北京玩两天，选择飞机出行，开三间大床房。请给我一个旅行规划。"
}
```

Example 3

```
{
    "start_city": "重庆",
    "target_city": "苏州",
    "hard_logic": [
        "days==3",
        "people_number==3",
        "cost<=7300",
        "{'日本料理'} <= food_type",
        "intercity_transport=={'train'}",
         "tickets==3",
        "rooms==2",
        "room_type==2"
    ],
    "nature_language": "当前位置重庆。我们三个人计划去苏州玩三天，选择火车出行，想吃日本料理，预算7300元，开两间双床房。请给我一个旅行规划。"
}
```

Figure 8: Examples of Generated Data

Function Name	Meaning	Implementation
day_count	total days in the plan	`def day_count(plan):` `  return len(plan["itinerary"])`
people_count	number of people in the trip	`def people_count(plan):` `  return plan["people_number"]`
start_city	start city of the plan	`def start_city(plan):` `  return plan["start_city"]`
target_city	target city of the plan	`def target_city(plan):` `  return plan["target_city"]`

allactivities	all the activities in the plan	`def allactivities(plan):`
		`  activity_list = []`
		`  for day_activity in plan["itinerary"]:`
		`  for act in day_activity["activities"]:`
		`  activity_list.append(act)`
		`  return activity_list`

allactivities_count	the number of activities in the plan	`def allactivities_count(plan):`
		`  count = 0`
		`  for day_activity in plan["itinerary"]:`
		`  count += \`
		`  len(day_activity["activities"])`
		`  return count`

dayactivities	all the activities in the specific day [1, 2, 3, ...]	`def dayactivities(plan, day):`
		`  activity_list = []`
		`  for act in plan["itinerary"]\`
		`  [day - 1]["activities"]:`
		`  activity_list.append(act)`
		`  return activity_list`

activity_cost	the cost of specific activity without transport cost	`def activity_cost(activity):` `  return activity.get("cost", 0)`
activity_position	the position name of specific activity	`def activity_position(activity):` `  return activity.get("position", "")`
activity_price	the price of specific activity	`def activity_price(activity):` `  return activity.get("price", 0)`
activity_type	the type of specific activity	`def activity_type(activity):` `  return activity.get("type", "")`
activity_tickets	the number of tickets needed for specific activity	`def activity_tickets(activity):` `  return activity.get("tickets", 0)`
activity_transports	the transport information of specific activity	`def activity_transports(activity):` `  return activity.get("transports", [])`
activity_start_time	the start time of specific activity	`def activity_start_time(activity):` `  return activity.get("start_time")`
activity_end_time	the end time of specific activity	`def activity_end_time(activity):` `  return activity.get("end_time")`

Table 6: Concept Function

Function Name	Meaning	Implementation

activity_time	the duration of specific activity	`def activity_time(activity):`
		`  start_time = activity.get("start_time")`
		`  end_time = activity.get("end_time")`
		`  if start_time and end_time:`
		`  st_h, st_m = \`
		`  map(int, start_time.split(":"))`
		`  ed_h, ed_m = \`
		`  map(int, end_time.split(":"))`
		`  return \`
		`  (ed_m - st_m) + (ed_h - st_h) * 60`
		`  return -1`

poi_recommend_time	the recommend time of specific poi(attraction) in the city	`def poi_recommend_time(city, poi):`
		`  select = Attractions().select`
		`  attrction_info = \`
		`  select(city, key="name",`
		`  func=lambda x: x == poi).iloc[0]`
		`  recommend_time = \`
		`  (attrction_info["recommendmintime"]) \`
		`  * 60`
		`  return recommend_time`

poi_distance	the distance between two POIs in the city	`def poi_distance(city, poi1, poi2):`
		`  start_time="00:00"`
		`  transport_type="walk"`
		`  goto = Transportation().goto`
		`  return goto(city, poi1, poi2, start_time,`
		`  transport_type)[0]["distance"]`

innercity_transport_cost	the total cost of specific innercity transport	`def innercity_transport_cost(transports, mode):`
		`  cost = 0`
		`  for transport in transports:`
		`  if node is None or \`
		`  transport.get("type") == node:`
		`  cost += transport.get("cost", 0)`
		`  return cost`

innercity_transport_price	the price of innercity transport	`def innercity_transport_price(transports):`
		`  price = 0`
		`  for transport in transports:`
		`  price += transport["price"]`
		`  return price`

innercity_transport_distance	the distance of innercity transport	`def innercity_transport_distance\`
		`  (transports, mode=None):`
		`  distance = 0`
		`  for transport in transports:`
		`  if mode is None or \`
		`  transport.get("type") == mode:`
		`  distance += \`
		`  transport.get("distance", 0)`
		`  return distance`

innercity_transport_time	the duration of innercity transport	`def innercity_transport_time(transports):`
		`  def calc_time_delta(end_time, start_time):`
		`  hour1, minu1 = \`
		`  int(end_time.split(":")[0]), \`
		`  int(end_time.split(":")[1])`
		`  hour2, minu2 = \`
		`  int(start_time.split(":")[0]), \`
		`  int(start_time.split(":")[1])`
		`  return (hour1 - hour2) * 60\`
		`  + (minu1 - minu2)`

Table 7: Concept Function

Function Name	Meaning	Implementation
metro_tickets	the number of metro tickets if the type of transport is metro	`def metro_tickets(transports):` `  return transports[1]["tickets"]`
taxi_cars	the number of taxi cars if the type of transport is taxi	`def taxi_cars(transports):` `  return transports[0]["cars"]`
room_count	the number of rooms of accommodation	`def room_count(activity):` `  return activity.get("rooms", 0)`
room_count	the number of rooms of accommodation	`def room_count(activity):` `  return activity.get("rooms", 0)`
room_type	the type of room of accommodation	`def room_type(activity):` `  return activity.get("room_type", 0)`

restaurant_type	the type of restaurant's cuisine in the target city	`def restaurant_type(activity, target_city):`
		`  restaurants = Restaurants()`
		`  select_food_type = \`
		`  restaurants.select(`
		`  target_city, key="name",`
		`  func=lambda x: x == activity["position"]`
		`  )["cuisine"]`
		`  if not select_food_type.empty:`
		`  return select_food_type.iloc[0]`
		`  return ""`

attraction_type	the type of attraction in the target city	`def attraction_type(activity, target_city):`
		`  attractions = Attractions()`
		`  select_attr_type = \`
		`  attractions.select(`
		`  target_city, key="name",`
		`  func=lambda x: x == activity["position"]`
		`  )["type"]`
		`  if not select_attr_type.empty:`
		`  return select_attr_type.iloc[0]`
		`  return ""`

accommodation_type	the feature of accommodation in the target city	`def accommodation_type(activity, target_city):`
		`  accommodations = Accommodations()`
		`  select_hotel_type = \`
		`  accommodations.select(`
		`  target_city, key="name",`
		`  func=lambda x: x == activity["position"]`
		`  )["featurehoteltype"]`
		`  if not select_hotel_type.empty:`
		`  return select_hotel_type.iloc[0]`
		`  return ""`

innercity_transport_type	the type of innercity transport	`def innercity_transport_type(transports):`
		`  if len(transports) == 3:`
		`  return transports[1]["mode"]`
		`  elif len(transports) == 1:`
		`  return transports[0]["mode"]`
		`  return ""`

intercity_transport_type	the type of intercity transport	`def intercity_transport_type(activity):` `  return activity.get("type", "")`

Table 8: Concept Function

Function Name	Meaning	Implementation
innercity_-transport_-start_time	the start time of innercity transport	`def innercity_transport_start_time(transports):` `  return transports[0]["start_time"]`
innercity_-transport_-end_time	the end time of innercity transport	`def intercity_transport_end_time(transports):` `  return transports[-1]["end_time"]`

intercity_-transport_-origin	the origin city of intercity transport	`def intercity_transport_origin(activity):`
		`  if "start" in activity:`
		`  for city in city_list:`
		`  if city in activity["start"]:`
		`  return city`
		`  return ""`

intercity_-transport_-destination	tthe destination city of intercity transport	`def intercity_transport_destination(activity):`
		`  if "end" in activity:`
		`  for city in city_list:`
		`  if city in activity["end"]:`
		`  return city`
		`  return ""`

Table 9: Concept Function

Prompts for POI recommendation

```
NEXT_POI_TYPE_INSTRUCTION = """
   You are a travel planning assistant.
   The user's requirements are: {}.
   Current travel plans are: {}.
   Today is {}, current time is {}, current location is {}, and
      POI_type_list is {}.
   Select the next POI type based on the user's needs and the
      current itinerary.
   Please answer in the following format.
   Thought: [Your reason]
   Type: [type in POI_type_list]
    """
```

Figure 9: Prompts for next-POI-type recommendation

Prompts for restaurants recommendation

```
RESTAURANT_RANKING_INSTRUCTION = """
    You are a travel planning assistant.
    The user's requirements are: {user_requirements}.
    The restaurant info is:
    {restaurant_info}
    The past cost for intercity transportation and hotel
       accommodations is: {past_cost}.

    Your task is to select and rank restaurants based on the
       user's needs and the provided restaurant information.
       Consider the following factors:
    1. Restaurant name
    2. Cuisine type
    3. Price range
    4. Recommended food

    Additionally, keep in mind that the user's budget is
       allocated across multiple expenses, including intercity
       transportation and hotel accommodations. Ensure that the
       restaurant recommendations fit within the remaining
       budget constraints after accounting for the past cost.
    Note that the price range provided for each restaurant is
       the average cost per person per meal, the remaining
       budget must cover the cost of three meals per day for {
       days} days.

    For each day, recommend at least 6 restaurants, combining
       restaurants for all days together.

    Your response should follow this format:

    Thought: [Your reasoning for ranking the restaurants]
    RestaurantNameList: [List of restaurant names ranked by
       preference, formatted as a Python list]
    """
```

Figure 10: Prompts for restaurant recommendation

Prompts for attractions recommendation

```
ATTRACTION_RANKING_INSTRUCTION = """
    You are a travel planning assistant.
    The user's requirements are: {user_requirements}.
    The attraction info is:
    {attraction_info}
    The past cost for intercity transportation and hotel
       accommodations is: {past_cost}.

    Your task is to select and rank attractions based on the
       user's needs and the provided attraction information.
       Consider the following factors:
    1. Attraction name
    2. Attraction type
    3. Location
    4. Recommended duration

    Additionally, keep in mind that the user's budget is
       allocated across multiple expenses, including intercity
       transportation and hotel accommodations. Ensure that the
       attraction recommendations fit within the remaining
       budget constraints after accounting for the past cost.

    For each day, recommend at least 8 attractions, combining
       attractions for all days together. To ensure a
       comprehensive list, consider a larger pool of candidates
       and prioritize diversity in attraction type and location.

    Your response should follow this format:

    Thought: [Your reasoning for ranking the attractions]
    AttractionNameList: [List of attraction names ranked by
       preference, formatted as a Python list]

    Example:
    Thought: Based on the user's preference for historical sites
        and natural attractions, the attractions are ranked as
       follows:
    AttractionNameList: ["Attraction1", "Attraction2", ...]
    """
```

Figure 11: Prompts for attraction recommendation