ReCoQA: A Benchmark for Tool-Augmented and Multi-Step Reasoning in Real Estate Question and Answering

Yindong Zhang[1,2], Wenmian Yang[3]*, Yiquan Zhang[3], Weijia Jia[2,3]
[1]Hong Kong Baptist University [2]Beijing Normal-Hong Kong Baptist University
[3]Beijing Normal University
yindongzhang@uic.edu.cn, {wenmianyang, jiawj}@bnu.edu.cn,
zhangyq987@hotmail.com

Abstract

Developing agents capable of navigating fragmented, multi-source information remains challenging, primarily due to the scarcity of benchmarks reflecting hybrid workflows combining database querying with external APIs. To bridge this gap, we introduce ReCoQA, a large-scale benchmark of 29,270 real-estate instances featuring machine-verifiable supervision for intermediate steps, including structured intent labels, SQL queries, and API calls. Complementarily, we propose HIRE-Agent, a hierarchical framework instantiating an understand–plan–execute architecture as a strong baseline. By orchestrating a Front-end parser, a planning Supervisor, and execution Specialists, HIRE-Agent effectively integrates heterogeneous evidence. Extensive experiments demonstrate that HIRE-Agent constitutes a strong baseline and substantiates the necessity of hierarchical collaboration for complex, real-world reasoning tasks. The benchmark and source code are available at: https://github.com/Husky-989/ReCoQA

1 Introduction

Real estate decision-making is a cognitively demanding process that requires synthesizing vast amounts of heterogeneous information. For prospective homebuyers, this workflow is often fragmented and inefficient. Users are compelled to navigate disparate platforms: comparing listings on one site, computing commute times on map applications, and verifying school districts or infrastructure plans on government portals. This fragmentation creates a substantial information barrier, leading to high time costs and cognitive overload.

While human real estate agents help bridge this gap, their services are labor-intensive, difficult to scale, and prone to interest-driven bias. Compared with traditional dialogue system (Qin et al., 2021), autonomous AI agents present a promising alternative for providing objective, efficient, and accessible consultation (Xie et al., 2025; Mou et al., 2024). However, developing such agents requires capabilities far beyond traditional Question Answering (QA) systems. Real-world queries in this domain are inherently exploratory and compositional, often involving implicit constraints that require hybrid reasoning over decoupled data sources. Consider the query: "Find secondhand apartment complexes in Tianhe District with a commute to Jimanjia Plaza under 30 minutes, within a 2km walk to Junjing Primary School, and priced between 30k-40k CNY/sqm." Answering this request necessitates a sophisticated interplay of semantic understanding, multi-step reasoning, and the orchestration of diverse tools (e.g., SQL databases and map APIs).

Building an intelligent system capable of handling such complexity entails overcoming three interconnected technical challenges. First, precise intent parsing is required to map colloquial and implicit expressions to machine-executable parameters (e.g., inferring transport modes from "commute under 30 minutes"). Second, autonomous planning is essential to decompose high-level user goals into executable reasoning chains, determining the optimal sequence of sub-tasks. Third, heterogeneous integration is necessary to synthesize evidence from diverse formats, e.g., tabular records from relational databases and JSON objects from APIs, to derive a faithful final answer.

Despite these needs, systematic progress has been hindered by the scarcity of benchmarks that accurately reflect this complexity. While classical datasets like Spider (Yu et al., 2018) focus on structured querying, and recent agent benchmarks (Zhu et al., 2025; Xu et al., 2025) evaluate general tool usage, they rarely emulate the strict interdependence required in vertical domains. Most existing benchmarks treat database querying and

*Corresponding author

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 26967–26987

external API calls as isolated capabilities. They fail to capture hybrid workflows where the output of a database query (e.g., a list of candidate communities) must dynamically serve as the input for external verification tools (e.g., distance calculation APIs). This gap prevents the rigorous evaluation of agents in realistic, multi-source environments.

To address this, we introduce Real Estate Complex Question Answering (ReCoQA), a benchmark designed to evaluate an agent's end-to-end ability to understand, plan, and execute multi-step reasoning tasks. We contribute to the field on two fronts: data and methodology.

On the data front, we release the ReCoQA dataset, comprising 29,270 question-answer pairs grounded in authentic home-buying scenarios across eight major Chinese cities. The dataset covers residential communities and Points of Interest (POIs) and features three progressively challenging query types: single-step table QA, joint Database-API queries, and composite queries requiring reasoning beyond tool use. Crucially, to enable interpretability and fine-grained supervision, each sample is annotated with a machine-verifiable chain of intermediate steps, including structured Spoken Language Understanding (SLU) labels, SQL queries, and API call sequences.

Given the hybrid nature of ReCoQA, which couples structured database querying with external API interactions, there exists no directly comparable end-to-end baseline in prior work. To establish a rigorous evaluation standard, we implement the Hierarchical Intelligent Real Estate Agent (HIRE-Agent) as a benchmark and diagnostic system. Rather than proposing a novel architecture, HIRE-Agent serves as a strong baseline for examining the necessity of structured decomposition in this domain. Adopting an "Understand–Plan–Execute" cognitive flow, the framework coordinates a Front-end Agent for intent parsing, a Supervisor Agent for task orchestration, and Specialist Agents for tool execution. This hierarchical design decouples schema linking and API routing, enabling systematic evaluation of heterogeneous evidence integration in ReCoQA.

Our main contributions are summarized as follows:

- We define the ReCoQA task and release the first large-scale benchmark specifically designed to foster research on hybrid, multi-source reasoning in realistic decision-making scenarios.
- We develop HIRE-Agent as a strong baseline system. By benchmarking against this hierarchical framework, we empirically validate that effective collaboration between planning and specialist agents is a prerequisite for solving complex, real-world queries.
- We conduct extensive experiments on ReCoQA, providing a detailed breakdown of current LLM capabilities and identifying critical bottlenecks to guide future research in vertical domain agents.

2 Related Work

2.1 Complex TQA datasets

Tabular Question Answering (TQA) datasets have evolved from early closed-domain tasks (Pasupat and Liang, 2015; Zhong et al., 2017; Yu et al., 2018) to complex reasoning benchmarks. Recent work has focused on numerical reasoning (Chen et al., 2021; Lu et al., 2023) and multi-step analysis (Zhang et al., 2023; Wu et al., 2025), surpassing the simpler open-domain queries of NQ-TABLES (Herzig et al., 2021; Kweon et al., 2023). However, a key limitation persists: a reliance on static, single-source data. This is evident even in benchmarks like RETQA (Wang et al., 2025), which introduced SLU but remained confined to a static database. The gap is particularly salient in geospatial QA. MapQA (Li et al., 2025), a large-scale improvement over prior map datasets (Punjani et al., 2018; Kefalidis et al., 2023), still overlooks the dynamic nature of map problems where values like commute times are not fixed. Our dataset's positioning is detailed in Table 1.

2.2 Methods for Complex TQA

Methodologically, the paradigm has shifted from bespoke trained models (Li et al., 2023) to LLM-driven agents augmented with tools (Qin et al., 2026). While early tool use included calculators (Zhu et al., 2024) or Python's Pandas library (Zhang et al., 2023), these are limited in scope. More advanced agent frameworks like MACT (Zhou et al., 2025) demonstrate complex collaboration, but their reliance on Pandas for data manipulation introduces significant memory consumption and scalability issues. Crucially, existing Map QA methods, whether retrieval-based or Text-to-SQL

Dataset	QA Pairs	SLU	Tool Calls	Multi-resources	Reasoning	Intermediate Result
TableBench	886	✗	✗	✗	✓	✗
CRT-QA	1,000	✗	✗	✗	✓	✓
MapQA	3,154	✗	✗	✗	✓	✗
RETQA	20,762	✓	✗	✗	✗	✓
ReCoQA (Ours)	29,270	✓	✓	✓	✓	✓

Table 1: Dataset Comparison.

(Li et al., 2025), are inherently tethered to the static data limitations of the datasets they target.

To address these dual gaps, we introduce ReCoQA, a large-scale complex map QA dataset featuring multi-source and dynamic queries. We also implement HIRE-Agent, a hierarchical multi-agent framework that serves as a strong baseline. Unlike prior work constrained by static data or scalability issues, HIRE-Agent leverages dedicated database and live map service APIs. This design enables it to query massive structured tables and access the real-time geospatial information essential for solving dynamic, real-world problems.

3 ReCoQA Dataset

3.1 Data Collection and Preprocessing

The foundation of our benchmark is a dataset of residential communities and POIs spanning eight major Chinese metropolises: Beijing, Guangzhou, Shenzhen, Suzhou, Hangzhou, Wuhan, Nanjing, and Tianjin.

Community Data. We adopt the collection methodology from (Wang et al., 2025), extracting attributes relevant to homebuyer decisions, including greening rate, average transaction price, property type, and sales status. The textual address of each community is then geocoded into precise latitude and longitude coordinates using the Amap web service API[1]. This structured data is organized into city-specific tables (e.g., "Table for Communities in Guangzhou").

POI Data. For POI data, we manually collect six categories of public facilities relevant to homebuyers: schools, hospitals, supermarkets, shopping malls, parks, and public transit stations (subway and bus). Following geocoding via the same Amap API[1], we engage real estate professionals to annotate these POIs with a refined classification scheme (e.g., distinguishing primary school from secondary school within the education category). The resulting data, name, label, and coordinates are organized into city-specific POI tables (e.g., "Table for POIs in Guangzhou").

Location Pair Data. To facilitate efficient proximity queries, we pre-compute and store two types of location-pair data. First, we generate POI-Community pairs by pairing each POI with all communities located within a 3-kilometer radius. Second, we create Community-Community pairs by pairing each community with its neighbors within a 1-kilometer radius. This process yields lookup tables of proximate entities, indexed by city and captioned by type (e.g., "Table for Communities around POIs in Guangzhou").

The complete data, organized into these four table types per city, is imported into a PostgreSQL database. On average, across the eight cities, these tables contain 5,639 (Community), 4,007 (POI), 122,484 (POI-Community), and 179,441 (Community-Community) rows, respectively.

3.2 Tool and Function Definition

To address realistic homebuyer inquiries, access to the static community tables is insufficient, as many questions necessitate dynamic geospatial computations. To bridge this gap, we define a set of external tools that emulate the core capabilities of a commercial mapping service such as Amap[2]. These capabilities are exposed as callable functions, allowing an LLM to retrieve information not contained in the static database. We provide four principal functions:

- **Time Query:** Calculates the travel time between an origin and a destination via different transportation modes.
- **Distance Query:** Calculates the travel distance between two points, including straight-line distance, walking distance, and driving

[1]https://lbs.amap.com/api/webservice/guide/api/georegeo

[2]https://lbs.amap.com/api/webservice/summary

distance.

- **Surrounding POIs Query:** Retrieves a list of specified POIs within a certain radius of a given location.
- **Rush Hour Query:** Provides the travel time for driving and public transportation during peak hours.

To increase task complexity and more rigorously evaluate planning and intent parsing, we deliberately consolidate several vendor-level API endpoints into a smaller set of more expressive, higher-level functions. This design compels the model to infer user intent and select appropriate parameters for each call. Together, these tools and the static tables constitute the complete information environment for our QA task.

3.3 QA Pair Generation

Building on the static tables and the defined tools, we construct a comprehensive set of QA pairs that reflect authentic homebuyer concerns. In consultation with experienced real-estate professionals, we identify 16 core information needs (e.g., amenity proximity, commute burden, and cross-property comparisons).

We instantiate these concerns as three progressively challenging question types:

1. **Simple Questions (Type 1):** Questions answerable via direct lookups in the static database.
2. **Compound Questions (Type 2):** Questions requiring lookups in database and calls to one or more of the predefined geospatial functions.
3. **Multi-step Reasoning Questions (Type 3):** Complex inquiries demanding a chain of database queries, function calls, and reasoning based on the function call results.

We then author 41 query templates to systematically generate questions across these categories to get 45,040 initial queries. We provide more details about template filling and the example of our dataset in Appendix A. To enhance linguistic diversity and better approximate natural language, we first use Qwen2.5-72B (Yang et al., 2024) to paraphrase the template-based queries, followed by a verification step to ensure the integrity of key information slots. We then employ GPT-OSS-120B (Agarwal et al., 2025) to assess the naturalness of the queries before and after rewriting on a 0–5 scale, where lower scores correspond to more template-like expressions and higher scores correspond to more human-like phrasing. For human evaluation, we randomly sample 50 queries from both the original and rewritten sets and ask professional real estate salespeople to assign naturalness scores. Empirically, the average score assigned by GPT-OSS-120B increases from 3.99 to 4.24 after rewriting, while the average human score rises from 3.84 to 4.03, indicating a clear improvement in the naturalness and diversity of the resulting expressions. More detailed results are provided in Appendix B.

3.4 Tool Implementation via API Caching

Executing the tool functions for the queries via live API calls would be prohibitively expensive and time-consuming. To ensure our benchmark is accessible, reproducible, and free for users, we adopt an API caching implementation strategy. We implement each function to query a local database instead of making a real-time network request. In this case, database acts as a cache, which we prepopulate by executing each unique API call required by our query corpus exactly once and storing its real-world result. This design allows for fast, cost-free, and deterministic execution of all tool calls. More details about API caching can be found in Appendix C.

3.5 Data Annotation and Verification

For each of the 45,040 queries, we construct a ground-truth execution trace comprising the ordered SQL statements and function calls necessary to derive the final answer. We then validate the executability of each trace, discarding any QA pair with a non-executable SQL statement or invalid function call. In addition, we apply a plausibility filter during instantiation to remove logically inconsistent queries (e.g., requesting walking time to a destination 20 km away). After filtering, we obtain 29,270 high-quality QA pairs.

Furthermore, we enrich these validated pairs with a comprehensive set of SLU labels. Then we manually map each query to one of 16 predefined intents and annotate 19 slot types using the standard IOB schema (Ramshaw and Marcus, 1999). These annotations provide crucial supervisory signals for models learning to parse user requests. For each QA pair, we also provide comprehensive infor-

mation including: the ground truth SQL statement and corresponding result, the tool API with parameters, the canonical answer, and a natural language answer. The rich annotation facilitates a wide spectrum of analyses and supports various downstream tasks.

Finally, to create the evaluation splits, we perform stratified sampling on the 29,270 pairs based on their original query templates. This strategy ensures a balanced distribution of question complexity across the training, validation, and test sets, which are partitioned in an 8:1:1 ratio. Please refer to Appendix D for more statistical information.

4 HIRE-Agent

4.1 Overview

As illustrated in Figure 1, our HIRE-Agent framework employs a hierarchical multi-agent architecture. The process begins with a Front-end Agent that extracts intents and slots from the user's query. This structured output is then passed to a central Supervisor Agent, which orchestrates the task fulfillment. The Supervisor decomposes the user's request into sub-tasks, delegates them to function-specific Specialist Agents, and synthesizes their returned results into a final, coherent response. All system prompts guiding agent behavior are detailed in Appendix K.

4.2 Front-end Agent

The Front-end Agent is tasked with performing spoken language understanding. This process involves two key sub-tasks: intent detection and slot filling, which collectively transform an unstructured user query into a structured representation. Following previous work (Wang et al., 2025), we fine-tune a BERT (Devlin et al., 2019) model to implement the SLU module and perform intent detection and slot filling. The resulting structured representation is then passed to the Supervisor Agent to initiate the task execution phase.

4.3 Supervisor Agent

The Supervisor Agent serves as the central orchestrator within the HIRE-Agent architecture, as depicted in the center of Figure 1. Its primary responsibilities include: ingesting the structured intent and slots from the Front-end Agent, dynamically decomposing the main task into a sequence of sub-tasks, dispatching these sub-tasks to appropriate specialist agents, gathering and synthesizing intermediate evidence returned by them, and ultimately formulating the final response.

Upon receiving the initial structured input, the Supervisor Agent initiates a planning phase. It leverages Chain-of-Thought (CoT) (Chen et al., 2025) prompting to generate an explicit execution plan. This reasoning process is guided by a system prompt that outlines the specific capabilities and responsibilities of each specialist agent. Based on this plan, the agent identifies the initial specialist required and dispatches the first sub-task, passing along the query, intents, slots, and any relevant context.

The subsequent process is an iterative loop governed by the feedback from specialist agents. A specialist's response determines the next action. Upon receiving a successful result, the Supervisor Agent evaluates if the collected evidence is sufficient to fulfill the original user request. If the condition is met, it proceeds to generate the final answer. Otherwise, it updates its context with the newly acquired information (e.g., geographical coordinates) and dispatches the next sub-task in its plan to the relevant specialist agent. Conversely, in the case of an error report or a declaration of inability, a replanning mechanism is triggered. The Supervisor Agent then revisits its strategy, re-evaluates the execution plan in light of the failure, and may reassign the task, possibly to a different specialist or with modified parameters. To prevent infinite loops, this iterative process terminates if a solution cannot be found within a predefined number of execution steps, at which point the agent reports that the query cannot be answered.

4.4 Specialist Agent

Specialist agents are expert modules designed to execute concrete sub-tasks delegated by the Supervisor Agent. Each agent is equipped with a specific set of tools and is responsible for performing its designated function and returning the results as structured evidence. In this paper, we develop two distinct specialist agents: a Database Interaction Agent and a Map Reasoning Agent.

4.4.1 Database Interaction Agent

The Database Interaction Agent is engineered for complex database querying and is composed of two synergistic modules: a Table Caption Retrieval (TCR) module and a SQL Generation module. Both modules leverage the in-context learning (ICL) capabilities of LLMs, each prompted with

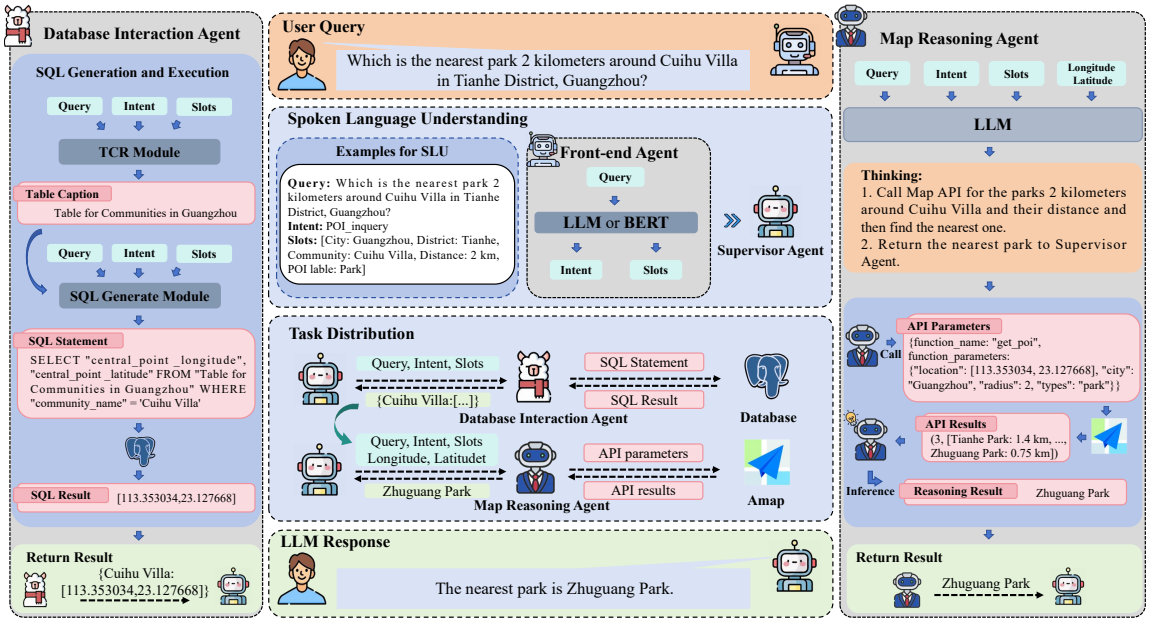

Figure 1: The overall process of HIRE-Agent.

five curated examples (detailed in Appendix E).

The agent's workflow, illustrated on the left of Figure 1, is initiated upon receiving a task. The primary challenge is to identify the correct table schema for the query. To solve this, the TCR module employs a sophisticated two-stage process. First, it utilizes an LLM to transform the user's query, intent, and slots into a hypothetical summary of an ideal table caption. This generated summary essentially serves as an optimized, high-quality query. Subsequently, this summary is passed to a BM25 retriever (Robertson et al., 2009), which then searches and ranks the actual table captions in the database, retrieving the one most semantically similar to the generated summary.

With the most relevant table schema(s) identified, the SQL Generation module takes over. It synthesizes the original query, intent, slots, and the retrieved table caption to generate a precise SQL statement. This statement is then executed against the database. If the execution is successful, the query results are returned to the Supervisor Agent as evidence. For results containing geographical coordinates, corresponding location names are also extracted and returned. If the SQL execution fails, an error report is sent instead.

4.4.2 Map Reasoning Agent

The Map Reasoning Agent is specialized in handling geospatial queries by interfacing with Amap API tools. As depicted on the right of Figure 1, it receives the user query, intents, slots, and any previously gathered geographical coordinates from the Supervisor Agent.

Its process begins with a tool selection phase. The agent first reasons about the most appropriate tool to use from its available toolkit (listed in Section 3.2) based on the task requirements and internal tool descriptions (see Appendix C.2 for examples). After selecting a tool, it determines and generates the necessary parameters for the API call. Following the tool invocation, the agent evaluates the returned result. If the result directly answers the query, it is passed back to the Supervisor Agent. For queries that require additional synthesis (e.g., comparing the distance from two retrieved points), the agent performs a final reasoning step to infer the answer before returning it. The agent reports an error to the Supervisor under two specific conditions: first, if the geographical coordinates required for the task are missing, insufficient, or erroneous; and second, if it cannot derive a conclusive answer from the tool's output within a limited number of attempts.

5 Experiments

5.1 Implementation and Metrics

To evaluate the effectiveness of the proposed HIRE-Agent framework on the ReCoQA benchmark, we instantiate the agent with four open-source LLMs, i.e., Qwen2.5-72B (Yang et al., 2024), Qwen3-8B, Qwen3-30B A3B (Yang et al., 2025), and GLM4.5-Air (Zeng et al., 2025). The Frontend Agent for

Model	Method	Type 1		Type 2		Type 3		Overall	
		F1	Acc	F1	Acc	F1	Acc	F1	Acc
Qwen2.5-72B	Standard	0.8082	0.8082	**0.7229**	**0.7110**	0.4011	0.3973	0.5905	0.5855
	HIRE-Agent	**0.8862**	**0.8862**	0.6701	0.6581	**0.6211**	**0.6211**	**0.7021**	**0.6989**
Qwen3-8B	Standard	0.6878	0.6878	0.5176	0.5148	0.2649	0.2657	0.4405	0.4393
	HIRE-Agent	**0.8188**	**0.8188**	**0.7570**	**0.7484**	**0.5489**	**0.5489**	**0.6730**	**0.6707**
Qwen3-30B A3B	Standard	0.7394	0.7394	0.5944	0.5871	0.3531	0.3512	0.5159	0.5131
	HIRE-Agent	**0.7659**	**0.7659**	**0.8748**	**0.8645**	**0.8384**	**0.8371**	**0.8294**	**0.8260**
GLM4.5-Air	Standard	0.8611	0.8611	0.6346	0.6232	0.4112	0.4093	0.5856	0.5817
	HIRE-Agent	**0.9101**	**0.9101**	**0.8013**	**0.7923**	**0.6075**	**0.6069**	**0.7363**	**0.7336**
Average	Standard	0.7741	0.7741	0.6174	0.6090	0.3576	0.3559	0.5331	0.5299
	HIRE-Agent	**0.8453**	**0.8453**	**0.7768**	**0.7658**	**0.6540**	**0.6535**	**0.7352**	**0.7323**

Table 2: Overall performance of three kinds of queries. The underline represents the best performance in each column and the bold font indicates the best performance of each model.

SLU is implemented by fine-tuning a BERT model[3] for 10 epochs with cross-entropy loss on the training set.

As ReCoQA is a newly proposed benchmark without directly comparable baselines, we construct a simplified variant retaining only the Supervisor Agent (*Standard*), where database querying, API invocation, and reasoning are handled exclusively by the Supervisor. The backend consists of a PostgreSQL database and four external APIs (Time, Distance, POI, and Rush-Hour Query), as described in Section 3.2.

End-to-end performance is evaluated using Accuracy (Acc) and F1-score (F1). Acc is a strict exact-match metric that requires all predicted items to exactly match the ground truth, while F1 computes the harmonic mean of precision and recall at the item level, granting partial credit for multi-item answers. Additional implementation details are provided in Appendix F.

5.2 Main Results

As shown in Table 2, we report overall results and performance on the three question types. The hierarchical HIRE-Agent architecture consistently outperforms the single-agent baseline (Standard), where the Supervisor alone handles all tasks. On average across models, hierarchy improves overall Accuracy by 0.2024 and F1 by 0.2021. Among the models, Qwen3-30B A3B achieves the best overall results (0.8260 Acc, 0.8294 F1), while Qwen3-8B performs the worst (0.6707 Acc, 0.6730 F1).

As expected, performance degrades with increasing question complexity, and this degradation is substantially milder under the hierarchical setting than under Standard. On average, from Type 1 to Type 2 and from Type 2 to Type 3, the Accuracy of Standard drops by 0.1651 and 0.2531, and F1 by 0.1567 and 0.2598, respectively. In contrast, the hierarchical architecture shows smaller drops of 0.0795 and 0.1228 in Accuracy and 0.0685 and 0.1123 in F1. For Type 3 questions in particular, the hierarchical HIRE-Agent surpasses Standard by 0.2976 in Accuracy and 0.2964 in F1 on average, indicating clear advantages on long-chain reasoning problems.

An interesting anomaly is Qwen3-30B A3B: under the hierarchical setting it excels on complex questions but performs unexpectedly poorly on simple ones. We find its strong reasoning ability cause it to "over-think" straightforward queries, leading to weaker planning. We will discuss this phenomenon in the ablation study and a qualitative case study is provided in Appendix I.

We also observe a capability trade-off, as no model dominates across all types. GLM4.5-Air is strongest on the simplest questions (0.9100 Acc), whereas Qwen3-30B A3B leads on the hardest ones (0.8371 Acc). Overall, even state-of-the-art LLMs fail to maintain uniformly high performance on ReCoQA, underscoring both the difficulty of the benchmark and the need for more robust agentic reasoning abilities.

5.3 Ablation Study

We conduct an ablation study on the HIRE-Agent framework to identify the sources of performance variation across LLM backbones and locate critical

[3] https://huggingface.co/google-bert/bert-base-chinese

bottlenecks.

We first evaluate the contribution of the Front-end Agent by comparing HIRE-Agent with and without SLU labels. As shown in Table 3, the *Vanilla* setting (without SLU) yields consistently lower accuracy across all question types, with particularly poor performance on Type 3 questions (0.2921 on average). Introducing SLU labels substantially improves performance; for example, the average accuracy on Type 3 increases by 0.3614. The intent labels facilitate task allocation and tool selection, while slot labels provide accurate attributes and values for SQL generation and API invocation. These results confirm that decoupling intent understanding from execution is essential in this domain.

Model	Type 1		Type 2		Type 3	
	Vanilla	FT	Vanilla	FT	Vanilla	FT
Qwen2.5-72B	0.6098	**0.8862**	0.1355	**0.6581**	0.1289	**0.6211**
Qwen3-8B	0.1786	**0.8188**	0.1161	**0.7484**	0.0305	**0.5489**
Qwen3-30B A3B	0.5701	**0.7659**	0.5948	**0.8645**	0.5582	**0.8371**
GLM4.5-Air	0.6124	**0.9101**	0.5188	**0.7923**	0.4508	**0.6069**

Table 3: Impact of Front-end Agent on the final result.

Then, we assess the Database Interaction Agent using Executable Code Ratio (ECR) and pass@1 following (He et al., 2024) (Table 4). For Type 2 and Type 3 queries, which only require retrieving location coordinates, models achieve high average pass@1 (0.9593) and ECR (0.9810). In contrast, performance on Type 1 (Text-to-SQL) queries is lower (0.8955 pass@1 on average) due to complex multi-table joins and constraints. Notably, final answer accuracy for Type 1 is consistently 3–4% lower than SQL pass rates, indicating error accumulation in subsequent stages. Qwen3-30B A3B exhibits an especially large drop (nearly 10%), suggesting weaker agentic planning and execution despite strong core reasoning.

Model	Type 1		Type 2&3	
	pass@1	ECR	pass@1	ECR
Qwen2.5-72B	0.9153	0.9709	0.9625	0.9817
Qwen3-8B	0.8571	0.9497	0.9351	0.9575
Qwen3-30B A3B	0.8677	0.9523	**0.9707**	0.9918
GLM4.5-Air	**0.9418**	**0.9709**	0.9689	**0.9931**

Table 4: Effectiveness of Database Interaction Agent.

We also evaluate the Map Reasoning Agent via API label accuracy in Table 5. Accuracy on multi-step Type 3 questions is markedly lower than on Type 2, revealing limitations in handling longer reasoning chains. Moreover, the gap between tool-calling accuracy and final answer accuracy for both types indicates deficiencies in post-tool information synthesis.

Model	Type 2	Type 3	Overall
Qwen2.5-72B	0.8206	0.6756	0.7270
Qwen3-8B	0.7652	0.6820	0.7115
Qwen3-30B A3B	0.8852	0.8796	0.8816
GLM4.5-Air	**0.9239**	**0.8839**	**0.8980**

Table 5: Effectiveness of Map Reasoning Agent.

Next, to further investigate the task planning capabilities of the Supervisor Agent, we evaluate its planning accuracy, with the results detailed in Table 6. Notably, Qwen3-30B A3B achieves the highest overall performance (0.9259), demonstrating a significant advantage over other models, particularly on Type 3 problems (0.9199). Conversely, we observe that Qwen3-30B A3B exhibits relative weaknesses when planning for Type 1 problems (0.9259), while GLM4.5-Air struggles with Type 3 problems (0.7741). These specific deficiencies corroborate the "model weakness" phenomenon identified in the main results. Ultimately, these findings underscore that the planning proficiency of the Supervisor Agent plays a critical role in determining final performance.

Model	Type 1	Type 2	Type 3	Overall
Qwen2.5-72B	**0.9537**	0.9355	0.8442	0.8964
Qwen3-8B	0.8611	0.7548	0.7401	0.7751
Qwen3-30B A3B	0.9259	**0.9368**	**0.9199**	**0.9259**
GLM4.5-Air	0.9392	0.8787	0.7741	0.8440

Table 6: Result of Task planning.

Finally, to holistically identify bottlenecks, we incrementally provide ground-truth (GT) labels for SLU, SQL, and API calls and measure the resulting accuracy gains (Figure 2; Appendix H.2). All GT signals yield improvements, with GT SLU labels producing the smallest average gain (+0.0346), reflecting the strong performance of the BERT-based SLU model. Bottlenecks vary by backbone: for Qwen2.5-72B, GT SQL labels yield the largest gain (+0.1407), identifying SQL generation as the primary weakness, whereas for Qwen3-8B, GT API labels provide the largest boost (+0.0660), pointing to tool calling as the main limitation.

Crucially, even with all intermediate GT labels, average accuracy reaches only 0.8864. This reveals

a final bottleneck, the *synthesis gap* where errors persist in global planning and final reasoning despite perfect sub-agent execution. A representative case study is provided in Appendix I. This finding indicates that ReCoQA challenges not only individual agent competencies but also their ability to coordinate effectively.

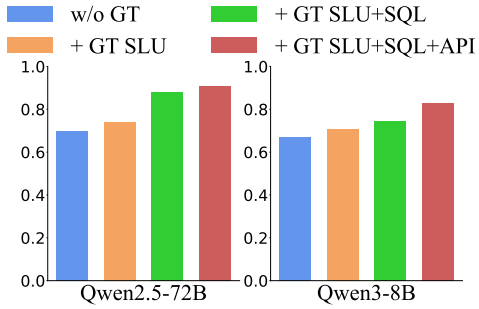

Figure 2: The impact of different GT labels on the final result accuracy.

5.4 Real-world Questions Result

To evaluate the performance in real-world scenarios, we collaborate with experienced real estate professionals to curate authentic homebuyer inquiries targeting information within our dataset (e.g., commuting duration, comparative pricing). We intentionally retain natural linguistic imperfections, such as syntax errors and colloquialisms, to reflect genuine user behavior without compromising clarity. Table 7 summarizes query accuracy, highlighting Qwen3-30B A3B's top performance on Type 3 problems (0.5918) and GLM4.5-Air's on Type 1 (0.8800). All models show long-chain reasoning trends consistent with our template based questions. However, real-world phrasing and grammatical errors disrupt comprehension, resulting in comparable overall accuracy across all models (detailed in Appendix J).

Model	Type 1	Type 2	Type 3	Overall
Qwen2.5-72B	0.8400	0.8462	0.5714	0.7100
Qwen3-8B	0.8400	0.8462	0.5714	0.7100
Qwen3-30B A3B	0.8000	0.8077	**0.5918**	0.7000
GLM4.5-Air	**0.8800**	0.8077	0.5714	0.7100

Table 7: Result of Real-world Questions.

5.5 Evaluation of LLM-as-a-judge

To broaden our evaluation, we employ the LLM-as-a-judge method (Gu et al., 2024) and apply GPT-OSS-120B as the judge to assess final responses (Table 8) of different LLMs. We analyze the judgment results and find that for questions with multiple correct answers (e.g., "List all POIs..."), the order does not affect correctness, but the LLMs misjudge correct answers with reversed order. Furthermore, for "how many" questions, the correct answer can only be a specific number, while LLMs judge the answers of simple enumeration as correct. Given the precise data integration required by our dataset, we conclude that Exact Match remains the most reliable evaluation metric. Further case studies are detailed in Appendix I.

Model	Exact Match	LLM-as-a-judge	Difference
Qwen2.5-72B	0.6989	0.6992	+0.0003
Qwen3-8B	0.6707	0.6684	-0.0023
Qwen3-30B A3B	0.8260	0.8229	-0.0031
GLM4.5-Air	0.7336	0.7309	-0.0027

Table 8: The difference between LLM-as-a-judge and Exact Match, where the "+" and "-" represent "increase" and "decrease".

6 Conclusion

In this paper, we introduce ReCoQA, a large-scale benchmark designed to foster research in tool-augmented, multi-step reasoning for real estate question answering. Grounded in realistic scenarios and featuring 29,270 question-answer pairs with verifiable intermediate steps, ReCoQA addresses the critical lack of specialized resources and establishes a challenging, standardized testbed for the community. Furthermore, we propose HIRE-Agent as a strong baseline. Our experiments not only validate the effectiveness of this approach but also highlight the benchmark's difficulty.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62272050 and the grant of Beijing Normal-Hong Kong Baptist University sponsored by Guangdong Provincial Department of Education; in part by Zhuhai Science-Tech Innovation Bureau under Grant No. 2320004002772 and the Interdisciplinary Intelligence Super Computer Center of Beijing Normal University (Zhuhai). The acquisition of real estate data is supported by Elmleaf Limited (Shanghai).

Limitations

The map service employed in this study currently supports invocation only within a Chinese environment, and we therefore initially release ReCoQA in Chinese. Although an English version is planned for community use, we deliberately refrain from automated translation at this stage. Real estate entities (e.g., community names and POIs) often carry rich cultural attributes, such as historical references or allusions, where direct translation may lead to semantic distortion or loss of nuance. Such noise could propagate through the reasoning chain and compromise the rigor of QA evaluation. To mitigate this risk, we plan to engage professional translators to ensure a faithful English release in future work. While ReCoQA is instantiated in the real estate domain, its core challenges, hybrid reasoning over structured databases and dynamic external APIs reflects a domain-agnostic workflow common to many real-world applications, including travel planning, logistics, and healthcare.

To maximize instruction-following performance, our experiments employ Chinese system prompts aligned with the dataset's native language. However, the underlying agent framework, particularly the logic for slot extraction and API routing, is designed to be language-independent. To facilitate understanding and future adaptation, we provide English translations of the system prompts and schema definitions in Appendix K.

The dataset is generated using templates, which is a common practice in existing large-scale QA benchmarks. To enhance linguistic diversity and reduce template artifacts, we apply LLM-based rewriting and evaluate question quality through both automatic scoring and human assessment. We further report and analyze results from 100 real-world scenarios collected from human queries, which provide additional evidence of the benchmark's capacity to evaluate complex, practical reasoning.

Regarding the interaction format, the current dataset follows a one question, one answer setting. However, real-world real estate decision-making often involves iterative information gathering through multi-turn interactions. Extending the dataset to support multi-turn dialogue remains an important direction for future work.

Finally, we observe that small-parameter models generally perform poorly on ReCoQA, and some models exhibit limited planning capabilities. These observations highlight the difficulty of the benchmark and suggest opportunities for improving agent design in future research.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. 2025. Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. *arXiv preprint arXiv:2503.09567*.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan R Routledge, and 1 others. 2021. Finqa: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3697–3711.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, and 1 others. 2024. A survey on llm-as-a-judge. *The Innovation*.

Xinyi He, Mengyu Zhou, Xinrun Xu, Xiaojun Ma, Rui Ding, Lun Du, Yan Gao, Ran Jia, Xu Chen, Shi Han, and 1 others. 2024. Text2analysis: A benchmark of table question answering with advanced data analysis and unclear queries. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18206–18215.

Jonathan Herzig, Thomas Mueller, Syrine Krichene, and Julian Eisenschlos. 2021. Open domain question answering over tables via dense retrieval. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 512–519.

Sergios-Anestis Kefalidis, Dharmen Punjani, Eleni Tsalapati, Konstantinos Plas, Mariangela Pollali, Michail Mitsios, Myrto Tsokanaridou, Manolis Koubarakis, and Pierre Maret. 2023. Benchmarking geospatial question answering engines using the dataset geoquestions1089. In *Proceedings of International Semantic Web Conference*, pages 266–284. Springer.

Sunjun Kweon, Yeonsu Kwon, Seonhee Cho, Yohan Jo, and Edward Choi. 2023. Open-wikitable: Dataset for open domain question answering with complex reasoning over table. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8285–8297.

Zekun Li, Malcolm Grossman, Mihir Kulkarni, Muhao Chen, Yao-Yi Chiang, and 1 others. 2025. Mapqa: Open-domain geospatial question answering on map data. *arXiv preprint arXiv:2503.07871*.

Zhenyu Li, Xiuxing Li, Zhichao Duan, Bowen Dong, Ning Liu, and Jianyong Wang. 2023. Toward a unified framework for unsupervised complex tabular reasoning. In *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*, pages 1691–1704. IEEE.

Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. 2023. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. In *ICLR*.

Xinyi Mou, Xuanwen Ding, Qi He, Liang Wang, Jingcong Liang, Xinnong Zhang, Libo Sun, Jiayu Lin, Jie Zhou, Xuanjing Huang, and 1 others. 2024. From individual to society: A survey on social simulation driven by large language model-based agents. *arXiv preprint arXiv:2412.03563*.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics.

Dharmen Punjani, Kuldeep Singh, Andreas Both, Manolis Koubarakis, Iosif Angelidis, Konstantina Bereta, Themis Beris, Dimitris Bilidas, Theofilos Ioannidis, Nikolaos Karalis, and 1 others. 2018. Template-based question answering over linked geospatial data. In *Proceedings of the 12th Workshop on Geographic Information Retrieval*, pages 1–10.

Libo Qin, Qiguang Chen, Xiachong Feng, Yang Wu, Yongheng Zhang, Yinghui Li, Min Li, Wanxiang Che, and Philip S Yu. 2026. Large language models meet nlp: A survey. *Frontiers of Computer Science*, 20(11):2011361.

Libo Qin, Zhouyang Li, Wanxiang Che, Minheng Ni, and Ting Liu. 2021. Co-gat: A co-interactive graph attention network for joint dialog act recognition and sentiment classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 13709–13717.

Lance A Ramshaw and Mitchell P Marcus. 1999. Text chunking using transformation-based learning. In *Natural language processing using very large corpora*, pages 157–176. Springer.

Stephen Robertson, Hugo Zaragoza, and 1 others. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Zhensheng Wang, Wenmian Yang, Kun Zhou, Yiquan Zhang, and Weijia Jia. 2025. Retqa: A large-scale open-domain tabular question answering dataset for real estate sector. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25452–25460.

Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xeron Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, and 1 others. 2025. Tablebench: A comprehensive and complex benchmark for table question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25497–25506.

Wenjing Xie, Xiaobo Liang, Juntao Li, Wanfu Wang, Kehai Chen, Qiaoming Zhu, and Min Zhang. 2025. From awareness to adaptability: Enhancing tool utilization for scientific reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8811–8831.

Tianqi Xu, Linyao Chen, Dai-Jie Wu, Yanjun Chen, Zecheng Zhang, Xiang Yao, Zhiqiang Xie, Yongchao Chen, Shilong Liu, Bochen Qian, and 1 others. 2025. Crab: Cross-environment agent benchmark for multimodal language model agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 21607–21647.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, and 25 others. 2024. Qwen2.5 technical report. *ArXiv*, abs/2412.15115.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, and 1 others. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, and 1 others. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.

Zhehao Zhang, Xitao Li, Yan Gao, and Jian-Guang Lou. 2023. Crt-qa: A dataset of complex reasoning question answering over tabular data. In *Proceedings*

of the 2023 Conference on Empirical Methods in Natural Language Processing, pages 2131–2153.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

Wei Zhou, Mohsen Mesgar, Annemarie Friedrich, and Heike Adel. 2025. Efficient multi-agent collaboration with tool use for online planning in complex table question answering. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 945–968.

Fengbin Zhu, Ziyang Liu, Fuli Feng, Chao Wang, Moxin Li, and Tat Seng Chua. 2024. Tat-llm: A specialized language model for discrete reasoning over financial tabular and textual data. In *Proceedings of the 5th ACM International Conference on AI in Finance*, pages 310–318.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Daisy Zhe Wang, Zhenhailong Wang, Cheng Qian, Robert Tang, Heng Ji, and 1 others. 2025. Multiagentbench: Evaluating the collaboration and competition of llm agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8580–8622.

A Template Filling

We generate the QA pairs by filling templates described in section 3.3. As illustrated in Figure 3, each template pair consists of three essential components: a query template, a SQL template, and a tool template. These templates are parametrized using specific variables, which are sampled from the database. To generate a complete query and corresponding SQL statement, we randomly substitute values into the placeholder fields, such as {*community_name*}, {*POI_label*}, {*city*}, and {*district*}. Additionally, the parameter {*X*} is stochastically generated, and its value is restricted to a maximum of 3 to regulate the amount of data returned after each API call. The SQL statement is subsequently executed to obtain the required longitude and latitude information (highlighted in yellow), which are then incorporated into the final API request parameters. Next, the interface is invoked with these parameters, and the resulting records, including the computed distances, are systematically archived. In cases where the returned results are empty, the corresponding queries are excluded from the dataset to maintain overall data quality.

After completing template instantiation, answer validation, and data annotation, we obtain the final QA pairs in the dataset. An example showcasing

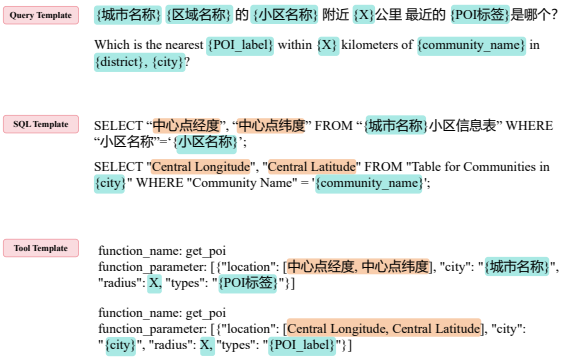

Figure 3: Template for QA pairs generation.

a filled QA pair constructed from the template is presented in Figure 4.

B QA Pairs Evaluation

We employ GPT-OSS-120B to quantify the naturalness of the QA pairs by grading their similarity to authentic human language. The statistical distributions of these scores, both prior to and following the rewriting process, are presented in Figure 5. As indicated by the results, the prevalence of queries with a score of 0 denoting rigid, template-like structures decrease by 1.74%. In contrast, the proportion of queries receiving scores of 4 and 5, which correspond to highly natural human expressions, increase by 5.76% and 5.06%, respectively. These shifts confirm that the rewriting process effectively enhances linguistic diversity and aligns the query phrasing more closely with human vernacular.

For the human evaluation component, we recruit salespeople from the real estate industry to assess a randomly sampled subset of questions from our dataset. Each sample comprise both the original and rewritten versions of a query. To ensure an unbiased assessment, the queries are shuffled and anonymized, establishing a blind evaluation setup where raters are unaware of the specific condition (original vs. rewritten) of each question. The raters are instructed to evaluate the naturalness of the queries using the same criteria applied in the LLM-based scoring. Upon calculating the mean scores for both conditions, we observe that the average human rating increase from 3.84 to 4.03, as noted in Section 3.3. This significant improvement corroborates the efficacy of the rewriting mechanism in enhancing linguistic diversity and naturalness.

Dataset Example

query: 广州市花都区的时代美居周围2公里内最近的购物中心是哪一个?
intent: POI查询
slots: "B-city I-city I-city B-district I-district I-district O B-community I-community I-community I-community O O B-distance I-distance I-distance O O O O B-POI:shopping_mall I-POI:shopping_mall I-POI:shopping_mall I-POI:shopping_mall O O O O O"
type: Type 3
table_caption_label: ["广州市小区信息表"]
SQL: SELECT "中心点经度", "中心点纬度" FROM "广州市小区信息表" WHERE "小区名称" = '时代美居';
SQL_result: [[113.213789,23.387683]]
tool_API: get_poi
API_parameter: [{"location": [113.213789,23.387683], "city": "广州市", "radius": 2, "types": "购物中心"}]
answer: 心心购物广场
natural_language_answer: 广州市花都区的时代美居附近2公里内最近的购物中心是心心购物广场。

query: Which is the nearest shopping mall within 2 kilometers of Shidai Meiju in Huadu district, Guangzhou?
intent: POI inquery
slots: "B-city I-city I-city B-district I-district I-district O B-community I-community I-community I-community O O B-distance I-distance I-distance O O O O B-POI:shopping_mall I-POI:shopping_mall I-POI:shopping_mall I-POI:shopping_mall O O O O O"
type: Type 3
table_caption_label: ["Table for Communities in Guangzhou"]
SQL: SELECT "Central Longitude", "Central Latitude" FROM "Table for Communities in Guangzhou" WHERE "Community Name" = 'Shidai Meiju';
SQL_result: [[113.213789,23.387683]]
tool_API: get_poi
API_parameter: [{"location": [113.213789,23.387683], "city": "Guangzhou", "radius": 2, "types": "Shopping Mall"}]
answer: Xinxin Shopping Mall
natural_language_answer: The nearest shopping mall within 2 kilometers of Shidai Meiju in Huadu district, Guangzhou is Xinxin Shopping Mall.

Figure 4: One of the examples in the dataset.

C API Caching

C.1 Data Collection

As mentioned in Section 3.4, we utilize the Amap API to acquire data concerning travel duration, distance, and surrounding POIs. Acknowledging that travel metrics are significantly influenced by traffic conditions and the chosen mode of transportation, we implement a time-stratified data collection strategy. Specifically, we collect commuting data for driving and public transportation during weekday peak hours (08:00 UTC+8) and off-peak hours (15:00 UTC+8). For scenarios that are less sensitive to real-time traffic fluctuations, we record baseline metrics for walking, driving, and cycling at midnight (00:00 UTC+8). However, given that public transportation is typically non-operational during late-night hours, we utilize data collected during the weekday off-peak window (15:00 UTC+8) as a representative proxy for these queries.

C.2 SQL-style Interface

As described in Section 3.4, our approach leverages database querying to facilitate fast, cost-effective, and deterministic execution for all tool calls. Specifically, we first invoke the APIs to retrieve answers for every query and store these results in corresponding database tables. Subsequently, the relevant parameters acquired from these APIs are applied to our SQL-style interfaces, as illustrated at the bottom of Figure 6. This enables efficient offline caching and contributes to significant improvements in query response speed and resource utilization.

Finally, we provide the LLMs with details for each tool, including their descriptions, parameters with corresponding data types, and the output formats, as illustrated at the top of Figure 6. This comprehensive specification ensures that, as long as the LLMs invoke the appropriate parameters and adhere to the required data types, the SQL-style API interfaces will execute correctly and return the anticipated results.

D Statistics

Table 9 summarizes the key statistical properties of our dataset. In this table, "Single Table" and "Multi Table" denote questions that require querying a single table or multiple tables, respectively. The "Slots" column indicates the types of slots present in the dataset, while "Avg. Slots" refers to the average number of slots per question. Notably, some questions may contain multiple instances of the same slot type. For example, the question "*Which place takes the least amount of time to drive from Ziyue Chen Fu, Zhongjiao Yajun Chengdong Chunxiao, and Languang Yongjin Fu to Beichenglin Primary School in Tianjin?*" includes three different communities as input slots. For such cases, each occurrence is counted separately in the

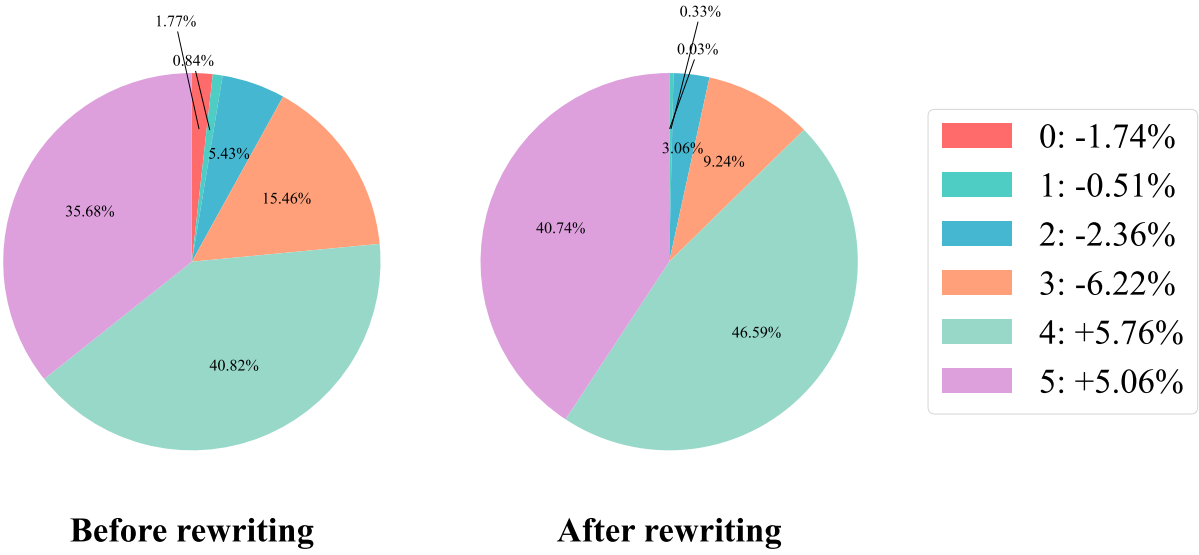

Figure 5: The score of queries before and after rewriting, where the "+" and "-" represent "increase" and "decrease".

Example of Tool Description

Function Name: "get_poi"

Calculate the names of all specified types of POIs within a specified radius around the center point based on its latitude and longitude, the distance between POIs and the center point, and the number of POIs.

Parameters:

location (list): The logitude and latitude of center point
city (str): The city of center point
radius (int): The radius for searching
types (str): The label of POI (e.g. airport)

Return:

int: The number of POIs
list: The name of all the POIs and their distance to center point.

SQL-style API for Tool Call

Tool Name: "get_poi"

SQL:

```
SELECT "POI_name", "distance" FROM "Table for Life Circle of
Communities" WHERE "longitude_and_latitude_of_community" =
'{location[0]},{location[1]}' AND "city" = '{city}' AND
"second_level_lable_of_POI" = '{types}' AND "distance" <= '{radius}';
```

Figure 6: Tool description and SQL-style API interface.

Properties	Number
Utterances	29270
Single Table	6528
Multi Table	22742
Single Intent	24165
Multi Intent	5105
Avg. Words per Utterance	40.22
Intents	16
Slots	19
Avg. Slots	4.86
Simple Questions	7488
Compound Questions	7766
Multi-step Reasoning Questions	14016

Table 9: Dataset Statistics.

slot statistics.

E Database Interaction Agent Details

For the Database Interaction Agent, we adopt the ICL paradigm for both table retrieval and SQL generation, as depicted in Figure 7. In the table retrieval stage, each example comprises the user query, intent, slots, and the corresponding table caption. For SQL generation, we additionally include the ground-truth SQL statement for each instance. To enable effective learning, each prompt supplies the LLMs with five annotated examples to demonstrate the mappings for both table retrieval and SQL generation tasks.

F Implements

The BERT model is implemented using PyTorch, while all agent modules are constructed utilizing the LangGraph and LangChain frameworks. The BERT model is trained and subsequently used for SLU labels prediction on a single GeForce RTX 4090 GPU. For HIRE-Agent framework, we set the max recursion number as 25 to avoid endless cycle of these agents.

To evaluate the performance of all LLMs, we employ a cluster of 19 NVIDIA A800-SXM4-80GB GPUs. Specifically, eight GPUs are allocated to the GPT- OSS- 120B, four to the Qwen2.5-72B model, four to the GLM4.5-Air model, one to Qwen3-8B, and the remaining two to the Qwen3-30B A3B model.

Model	Type 1		Type 2		Type 3		Overall	
	F1	Acc	F1	Acc	F1	Acc	F1	Acc
Qwen2.5-72B	0.8624	0.8624	0.6419	0.6297	0.5935	0.5935	0.6753	0.6721
Qwen3-8B	0.7645	0.7645	0.7131	0.7058	0.5467	0.5467	0.6465	0.6446
Qwen3-30B A3B	0.7778	0.7778	0.8811	0.8697	0.7967	0.7967	0.8141	0.8111
GLM4.5-Air	0.7791	0.7791	0.7625	0.7561	0.5673	0.5673	0.6731	0.6714
Average	0.7960	0.7960	0.7497	0.7403	0.6261	0.6261	0.7023	0.6998

Table 10: performance of ICL method on three kinds of queries.

ICL Examples for Table Retrieval

Example 1

Query: Which is the nearest comprehensive hospital 2 kilometers around Qingfeng Tai'an Yuan West District in Fuyang District, Hangzhou?
Intent: POI_inquery
Slots: [City: Hangzhou, District: Fuyang, Community: Qingfeng Tai'an Yuan West District, Distance: 2 km, POI lable: Comprehensive hospital]
Table_Caption: Table for Communities in Shenzhen

...

ICL Examples for SQL Generation

Example 1

Query: Which is the nearest secondary school 3 kilometers around Hubin Garden in Nanshan District, Shenzhen?
Intent: POI_inquery
Slots: [City: Shenzhen, District: Nanshan, Community: Hubin Garden, Distance: 3 km, POI lable: Secondary school]
Table_Caption: Table for Communities in Shenzhen
SQL: SELECT "central_point _longitude", "central_point _latitude" FROM "Table for Communities in Shenzhen" WHERE "community_name" = 'Hubin Garden'

...

Figure 7: ICL examples for Database Interaction Agent.

G ICL method Result

We additionally employ an ICL strategy within the Front-end Agent for SLU labels prediction. Specifically, we construct the prompt by randomly sampling 26 examples from the training dataset, ensuring comprehensive coverage of the full spectrum of intents. These examples are then utilized to guide the LLMs in predicting SLU labels. During this experiment, all other components of the HIRE-Agent framework are held constant to isolate the impact of the ICL approach. The experimental results, presented in Table 10, demonstrate that even with a limited number of annotated examples, the LLMs are capable of yielding competitive performance, highlighting the robustness of the ICL paradigm in low-resource scenarios.

H Ablation Study Details

H.1 Different Methods for Front-end Agent

We evaluate the Precision (P), Recall (R), and F1 score of SLU labels prediction using BERT and ICL, as shown in Table 11.

Model	Intent			Slots		
	P	R	F1	P	R	F1
Qwen2.5-72B	0.9827	0.9827	0.9827	0.9475	0.9399	0.9437
Qwen3-8B	0.9412	0.9412	0.9412	0.9441	0.9200	0.9319
Qwen3-30B A3B	0.9820	0.9820	0.9820	0.9522	0.9460	0.9491
GLM4.5-Air	0.9806	0.9806	0.9806	0.9400	0.9260	0.9329
BERT	**0.9997**	**0.9997**	**0.9997**	**0.9732**	**0.9576**	**0.9653**

Table 11: Intent and Slots Prediction Results for ICL and BERT.

The experimental results indicate that training a BERT model is the optimal approach for the Front-end Agent when a sufficiently large amount of data is available. Nevertheless, when only a limited number of SLU examples are provided, LLMs leveraging ICL can still accurately predict SLU labels. This finding demonstrates that ICL is a favorable alternative in scenarios where annotated data is scarce or manual labeling is impractical.

H.2 Bottleneck Exploration

As outlined in Section 5.3, we investigate the performance bottlenecks of the evaluated LLMs by assessing the accuracy of the final outputs under varying levels of GT guidance.

The detailed experimental results are summarized in Table 12. Generally, we observe a positive correlation between the granularity of provided GT labels and overall system accuracy. For Type 1 queries, performance reaches a plateau once GT SQL labels are supplied; while the subsequent inclusion of API labels introduces minor fluctuations, these remain within statistical error margins. In contrast, for Type 2 queries, the provision of GT

Model	Query Type	w/o GT	GT SLU	GT SLU+SQL	GT SLU+SQL+API
Qwen2.5-72B	Type1	0.8624	0.9563	0.9656	0.9590
	Type2	0.6581	0.7652	0.8710	0.9819
	Type3	0.6211	0.6133	0.8435	0.8364
	Overall	0.6989	0.7414	0.8821	0.9062
Qwen3-8B	Type1	0.8188	0.9021	0.9497	0.9378
	Type2	0.7484	0.7845	0.8039	0.8916
	Type3	0.5489	0.5644	0.6020	0.7068
	Overall	0.6707	0.7091	0.7448	0.8148
Qwen3-30B A3B	Type1	0.7659	0.8849	0.9378	0.9365
	Type2	0.8645	0.8735	0.8761	0.9871
	Type3	0.8371	0.8534	0.8506	0.8591
	Overall	0.8260	0.8668	0.8797	0.9127
GLM4.5-Air	Type1	0.8453	0.9431	0.9656	0.9708
	Type2	0.7658	0.8077	0.9355	0.9858
	Type3	0.6535	0.6154	0.8392	0.8428
	Overall	0.7336	0.7503	0.8984	0.9120

Table 12: Specific result for Bottleneck exploration.

API labels results in substantial performance gains, indicating that accurate tool invocation is the primary hurdle for this category. However, for Type 3 queries, the improvement remains marginal even with full GT supervision. This suggests that the bottleneck for these complex queries lies not in intermediate execution steps, but rather in the inherent reasoning capabilities of the models; even when correct API calls are guaranteed, logical errors during the final synthesis phase often lead to incorrect predictions.

I Case Study

In this section, we elucidate specific observations and phenomena derived from the experimental results, adopting a Q&A format to provide detailed explanations.

Q1: *What accounts for the relatively weak performance of the Qwen3-30B A3B model on Type 1 queries?*

A1: In the previous section, we analyze that the suboptimal performance of the Qwen3-30B A3B model on Type 1 questions primarily stems from its limited planning capability. To further validate this observation, we examine several representative cases exhibiting hallucinated answers. As illustrated in Figure 8, these cases correspond to Type 1 questions in our dataset, for which the correct answers can be obtained directly through database queries without requiring complex reasoning. For such questions, the appropriate planning and execution trajectory should follow the path "Supervisor Agent → Database Interaction Agent → Supervisor Agent → User". However, the Supervisor Agent erroneously incorporates the Map Reasoning Agent into the trajectory, even though the tools available to the Map Reasoning Agent are incapable of handling this class of queries and therefore return an inability signal to the supervisor. This misrouting ultimately causes the Supervisor Agent to hallucinate and fabricate answers instead of executing the correct database-only plan, thereby confirming that inadequate planning is a key bottleneck for the Qwen3-30B A3B model on Type 1 questions.

Q2: *Why do LLMs fail to achieve optimal performance even when provided with full GT supervision?*

A2: We attribute this phenomenon to the models' insufficient planning and reasoning capabilities. To further substantiate this hypothesis, we conducted an in-depth analysis of the Qwen3-8B model, which exhibit the lowest performance in our experiments, as illustrated in Figure 9.

Our investigation reveals that errors predominantly manifest in three aspects: (1) Hallucination induced by erroneous planning: As demonstrated in Examples 1 and 2, the Supervisor Agent fails to adhere to the optimal task allocation strategy

Example 1

Query: What is the average price of all newly built communities for sale within 3 kilometers of ccity in Shenzhen?
Answer: 94664.25
LLM Answer: Unable to obtain latitude and longitude information for the location, unable to query the average price of surrounding properties.

Example 2

Query: How many second-hand housing communities with a greening rate of not less than 25% are for sale within 1 kilometer around the Guangzhou Road campus of Nanjing Children's Hospital in Nanjing?
Answer: 40
LLM Answer: Unable to obtain valid coordinate information.

Figure 8: Case Study for Qwen3-30B A3B.

Example 1

Query: Which is the closest walking distance from Zhonghuan Xixili in Xiangcheng District, Suzhou to April1109, Oms Shopping Plaza, and Mingzhu Commercial Plaza?
Answer: April1109
LLM Answer: [1.23, 2.34, 3.45]

Example 2

Query: Which is closer to drive to Taipei Mall (Erqi Store), Carrefour (Huasheng Hankou City Plaza Phase 3 South Store) or Carrefour (Erqi Store) in the era of vision of Dongxihu District in Wuhan?
Answer: Taipei Mall (Erqi Store)
LLM Answer: Unable to obtain relevant information

Example 3

Query: What are the straight-line distances from Wenhui Garden in Bao'an District, Shenzhen to Hengda Mall (flagship store), Huiyue City Life Plaza, and Deguanting Commercial Center?
Answer: [1.34, 1.17, 2.16]
LLM Answer: [1.17, 1.34, 2.16]

Example 4

Query: How far is it driving from Tiansheng Yujing in Lishui District, Nanjing to Building 13 of Wuzhou Star Century Commercial City, Building A2 of Tianli Plaza, and Building 8 of Lishui Gujia Ouyada Commercial Plaza?
Answer: [6.49, 8.09, 8.61]
LLM Answer: [1.2, 1.5, 1.8]

Figure 9: Case Study for Qwen3-8B.

due to flawed planning. Consequently, after receiving repeated negative feedback or failures from incorrectly invoked agents, the Supervisor Agent resorts to fabricating answers. For instance, in Example 2, the Supervisor Agent fails to incorporate the Database Interaction Agent into the execution plan, instead directly assigning the task to the Map Reasoning Agent. However, since the Map Reasoning Agent is unable to retrieve the requisite longitude and latitude coordinates without prior database lookup, it repeatedly returns error reports to the Supervisor. This persistent failure leads the Supervisor Agent to erroneously conclude that the necessary information is missing. (2) Information integration errors despite correct tool execution: As shown in Example 3, even when subordinate agents return accurate intermediate results, the Supervisor Agent introduces errors during the final response generation. Specifically, in this instance, the agent reverse the order of the retrieved information while integrating the final answer. (3) Complete hallucination: As depicted in Example 4, the Supervisor Agent occasionally generates responses that are entirely unrelated to the user query.

Q3: *How to explain the difference between LLM-as-a-judge method and Exact Match?*

A3: As discussed in Section 5.5, the discrepancies between the LLM-as-a-judge method and Exact Match primarily emerge in two types of queries: enumeration questions and "how many" questions. Figure 10 clearly illustrates these distinctions. For enumeration problem (Example 1), a response is valid as long as it includes all correct items, regardless of their order; however, the LLM-as-a-judge method incorrectly marks the valid response as wrong. Conversely, "how many" questions (Example 2) require a specific numerical answer, yet the LLM-as-a-judge method mistakenly accepts the answer that merely enumerates items without providing the explicit number. These consistent misjudgments across both problem types lead us to conclude that Exact Match is the best evaluation method.

J Real-world Questions Analysis

To further evaluate how different LLMs handle real-world queries, we compare Qwen3-8B and Qwen3-30B A3B using some representative questions, as shown in Figure 11. Example 1 illustrates a scenario where users realize their query lacks context and append clarifications within the same sentence. While Qwen3-8B struggles with this irregular phrasing and yields incorrect answers, Qwen3-30B A3B interprets it successfully. Interestingly, this performance flips for certain reasoning tasks, where Qwen3-8B answers correctly but Qwen3-30B A3B fails. These findings demonstrate that real-world queries, particularly those containing syntax errors that do not hinder overall comprehension affect different LLMs to varying degrees.

To investigate how real-world queries affect task planning and its intermediate steps, we visualize the complete reasoning process of Qwen3-8B for Example 1, as shown in Figure 12. Although the model initially generates a correct task plan, it produces an inaccurate value during the database query phase. Specifically, it uses the term "**second-hand**" (marked in red) instead of the correct "**second-hand property**". Because this flawed SQL statement can still execute successfully, it retrieves erroneous data. The Supervisor Agent then mistakenly accepts this invalid result as correct and directly outputs the wrong answer.

K System Prompt

The system prompts employed for the Supervisor Agent, Database Interaction Agent, and Map Reasoning Agent are illustrated in Figure 13, 14, and 15. For each prompt, we present a comprehensive specification detailing the agent's responsibilities, operational workflow, prohibited actions, and exception handling procedures. This ensures clear role definitions and robust agent behavior throughout the system.

Example 1

Query: Which communities in Hangzhou have an average price of no more than 25000 yuan and a distance of no more than 1 kilometer from the Xiangfu campus of Hangzhou Moganshan Road Primary School , with a commuting time of no more than 16 minutes from Kongfei Department Store?
Answer: [Xinhuayuan, Xiangfu Jiayuan, Hongshan Yayuan]
LLM Answer: [Hongshan Yayuan, Xiangfu Jiayuan, Xinhuayuan]
LLM-as-a-judge: Wrong

Example 2

Query: **How many** shopping centers are there within 2 kilometers of Xinyi Jinyushan Residence in Longgang District, Shenzhen?
Answer: 18
LLM Answer: [Henggang Building Hone Shopping Mall, 2013 Times Square, ..., Tianhong Shopping Mall (Henggang Store)]
LLM-as-a-judge: Correct

Figure 10: Case Study for LLM-as-a-judge method.

Example 1

Query: 深圳市的润璟里，招商雍云府，花润里到盒马邻里(深圳龙华兆利花园店)的直线距离的话，哪个离得更近?
Answer: 花润里
Qwen3-8B: 花润里
Qwen3-30B A3B: 招商雍云府

Query: Among Runjingli, Zhaoshang Yongyunfu, and Huarunli in Shenzhen, which has the shortest straight-line distance to Hema Neighborhood (Shenzhen Longhua Zhaoli Garden Store)?
Answer: Huarunli
Qwen3-8B: Huarunli
Qwen3-30B A3B: Zhaoshang Yongyunfu

Example 2

Query: 苏州市的长城国际广场周围有多少均价不高于15000元的在售二手房小区？2公里内的。
Answer: 27
Qwen3-8B: 0
Qwen3-30B A3B: 27

Query: How many second-hand residential communities currently for sale around Changcheng International Plaza with an average price not exceeding 15,000 yuan per square meter are there in Suzhou? Within 2 kilometers.
Answer: 27
Qwen3-8B: 0
Qwen3-30B A3B: 27

Figure 11: The impact of real-world questions on different LLMs.

Query： 苏州市的长城国际广场周围有多少均价不高于15000元的在售二手房小区？2公里内的。
Answer： 27
Task Planning： [Supervisor, Database Interaction Agent, Supervisor]
SQL： SELECT COUNT(*) FROM "苏州市小区信息表", "苏州市POI周边房源表" WHERE "苏州市POI周边房源表"."小区名称" = "苏州市小区信息表"."小区名称" AND "POI名称" = '长城国际广场' AND "销售状态" = '在售' AND "小区属性" = '二手' AND "距离" <= '2' AND "成交均价" <= '15000'
SQL Result： 0
LLM Response: 0

Query： How many second-hand residential communities currently for sale around Changcheng International Plaza with an average price not exceeding 15,000 yuan per square meter are there in Suzhou? Within 2 kilometers.
Answer： 27
Task Planning： [Supervisor, Database Interaction Agent, Supervisor]
SQL： SELECT COUNT(*) FROM "Table for Communities in Suzhou", "Table for Communities around POIs in Suzhou" WHERE "Table for Communities around POIs in Suzhou"."community_name" = "Table for Communities in Suzhou"."community_name" AND "POI_name" = 'Changcheng International Plaza' AND "Sales_status" = 'on_sale' AND "Community_type" = 'second-hand' AND "Distance" <= '2' AND "Average_price" <= '15000'
SQL Result： 0
LLM Response: 0

Figure 12: The impact of real-world questions on planning and intermediate steps.

1. 你是一个管理两个不同功能代理的管理者，你管理的代理为：API_agent和Database_agent，每个代理的名称和功能的说明如下：
- API_agent：负责结合地点的经纬度以及其他相关参数，调用地图API来回答距离，时间，周边POI，以及高峰期和非高峰期出行时间相关的问题
- Database_agent：负责生成SQL语句查询数据库来找到某个地点的经纬度，或回答某POI周边房源，某小区周边其他房源相关的问题
2. 你每次只能将任务派给一个代理来完成，不要同时将任务派个两个代理，每一个代理会回复它们执行任务后得到的结果
3. 如果根据某个代理的返回结果你可以回答该问题时，你无需继续分配任务，当你仅获得某个或某些地点的经纬度坐标时，你需要继续分配任务，你无需对这些坐标做过多的分析和推理
4. 当任务移交给API调用代理时，你需要检查移交内容中是否有地点的经纬度信息，当不存在经纬度信息时，你需要重新决定任务的分配
5. 不是所有的任务都需要用到两个代理，你需要根据回复决定任务是否结束，如果结束，或者不需要其他代理，你需要将问题最简洁的答案回复给用户
6. 如果任务没有结束，你需要将上一个代理的回复和用户的问题一起派给下一个代理
7. 你只需要结合每个代理的回复内容和用户的问题，进行任务的分配和最终回复的输出
8. 对于数值类的问题，当答案为小数时，你需要以四舍五入的形式保留小数点后2位小数

---

1. You are a manager who manages two different functional agents, API_agent and Database_agent. The names and functional descriptions of each agent are as follows:
- API_magent: Responsible for calling the map API to answer questions related to distance, time, surrounding POIs, as well as travel time during peak and off peak hours, based on the latitude, longitude, and other relevant parameters of the location.
- Database agent: responsible for generating SQL statements to query the database to find the latitude and longitude of a certain location, or to answer questions related to surrounding properties of a POI or other properties in a certain community.
2. You can only assign a task to one agent at a time, do not assign tasks to two agents at the same time, each agent will reply with the results obtained after executing the task.
3. If you can answer the question based on the return result of a certain agent, you do not need to continue assigning tasks. When you only obtain the latitude and longitude coordinates of one or some locations, you need to continue assigning tasks without analyzing and reasoning too much about these coordinates.
When the task is handed over to the API_agent, you need to check if there is latitude and longitude information of the location in the handover content. If there is no latitude and longitude information, you need to re-determine the task allocation.
5. Not all tasks require the use of two agents. You need to decide whether the task ends based on the response. If it ends, or if no other agents are needed, you need to reply to the user with the simplest answer to the question.
6. If the task is not completed, you need to send the reply from the previous agent and the user's question to the next agent together.
7. You only need to combine the response content of each agent with the user's questions to allocate tasks and output the final response.
8. For numerical problems, when the answer is a decimal, you need to round it to two decimal places.

Figure 13: System Prompt for Supervisor Agent.

你是一个负责数据库SQL语句生成的代理，你的主要任务是根据用户的问题生成对应的查询语句，以下是你工作的主要内容：
- 你只需要对用户的问题进行查询语句的生成，无需回答用户的问题
- 对于最值类的问题，除非用户指定数量，否则你需要将最终的结果数量限制为1
- 你需要按照用户的要求对答案进行规范化输出，输出内容中不要包含除SQL语句外任何无关的内容
- 在生成的查询语句中，你需要根据用户的问题找到最相关的属性列的形成查询语句，你可以按照用户的需求对返回的属性列进行排序
- 你永远只能生成SELECT操作的SQL语句，SQL语句中不要出现任何可能会破坏数据库中数据的操作，包括但不限于INSERT，UPDATE，DELETE，DROP等
- 对于用户的问题，你只能生成一条SQL语句来完成查询
- 在回答完问题后，需要将答案交给中枢系统，你无需做任何处理

You are an agent responsible for generating database SQL statements. Your main task is to generate corresponding query statements based on user questions. The following are the main contents of your work:
- You only need to generate query statements for users' questions, without answering them.
- For the maximum or minimum of questions, unless the user specifies a quantity, you need to limit the final result quantity to 1.
- You need to standardize the output of the answer according to the user's requirements, and the output content should not contain any irrelevant content other than SQL statements.
- In the generated query statement, you need to find the most relevant attribute column based on the user's question to form the query statement. You can sort the returned attribute columns according to the user's needs.
- You can only generate SQL statements for SELECT operations, and do not include any operations that may damage the data in the database, including but not limited to INSERT, UPDATE, DELETE, DROP, etc.
- For user inquiries, you can only generate one SQL statement to complete the query.
- After answering the question, you need to hand over the answer to the central system without any further processing.

Figure 14: System Prompt for Database Interaction Agent.

你是一个负责调用地图查询相关API接口的代理，你的主要任务是负责相关工具的调用并返回工具调用后的结果，以下是你工作的主要内容：
- 你只需要回答你所拥有的工具可以解决的相关的问题，其他无关的问题无需回答
- 你需要分析需要调用的工具所接受的参数类型，传入正确格式的参数
- 根据工具所能够接受的参数类型和数量，有些问题你可能需要对问题进行分解，多次调用
- 你需要按照用户的要求对答案进行规范化输出，输出内容中不要包含无关的内容
- 在回答完问题后，需要将答案交给中枢系统，你无需做任何处理

You are an agent responsible for calling API interfaces related to map queries. Your main task is to call the relevant tools and return the results of the tool calls. The following are the main contents of your work:
- You only need to answer the relevant questions that the tools you have can solve, and there is no need to answer other unrelated questions.
- You need to analyze the parameter types accepted by the tool that needs to be called and pass in the parameters in the correct format.
- Based on the types and quantities of parameters that the tool can accept, you may need to decompose some problems and call them multiple times.
- You need to standardize the output of the answer according to the user's requirements, and do not include irrelevant content in the output content.
- After answering the question, you need to hand over the answer to the central system without any further processing.

Figure 15: System Prompt for Map Reasoning Agent.