Text-to-TrajVis: Enabling Trajectory Data Visualizations from Natural Language Questions

Tian Bai[1,3], Huiyan Ying[2,3], Kailong Suo[1,3], Junqiu Wei[4], Tao Fan[5], Yuanfeng Song[*]
[1]College of Computer Science and Technology, Jilin University, Changchun, China
[2]College of Software, Jilin University, Changchun, China
[3]Key Laboratory of Symbolic Computation and Knowledge Engineering, Jilin University, Changchun, China
[4]Macau University of Science and Technology, Macau, China
[5]Hong Kong University of Science and Technology, Hong Kong, China

Abstract

This paper introduces the **Text-to-TrajVis** task, which aims to transform natural language questions into trajectory data visualizations, facilitating the development of natural language interfaces for trajectory visualization systems. As this is a novel task, there is currently no relevant dataset available in the community. To address this gap, we first devised a new visualization language called Trajectory Visualization Language (TVL) to facilitate querying trajectory data and generating visualizations. Building on this foundation, we further proposed a dataset construction method that integrates Large Language Models (LLMs) with human efforts to create high-quality data. Specifically, we devised a four-stage pipeline that begins with candidate extraction, proceeds through seed TVL generation and tree-based expansion, and concludes with LLM-driven question creation followed by human validation. This process results in the creation of the first large-scale Text-to-TrajVis dataset, named **TrajVL**[1], which contains 9,608 *(question, TVL)* pairs. We propose a framework called **TRCAT** for progressively converting natural language questions into TVLs. The framework incorporates TVL-RAG Chain Module and Area-Time Standardization Module, significantly enhancing the accuracy of LLMs in TVL generation. Based on the TrajVL dataset, we conduct a comprehensive evaluation of TRCAT's performance across several mainstream LLMs (e.g., GPT, Qwen, LLaMA, and Gemma). Furthermore, we established a benchmarking system for this task, providing a foundation for future research in structured trajectory language generation.

1 Introduction

The growing availability of trajectory datasets such as GeoLife+ (Amiri et al., 2024b), T-Drive (Yuan et al., 2010), and WorldTrace (Zhu et al., 2024b) provides rich spatio-temporal data for research. Visualization plays a key role in applications such as human mobility analysis (Toch et al., 2019; Zhu et al., 2024a), disease modeling (Amiri et al., 2024a; Kohn et al., 2023), and animal migration tracking (Konzack et al., 2019). However, existing methods for trajectory data querying and visualization often rely on specialized expertise, including database query languages, programming skills, and the configuration of visualization parameters. This high technical barrier makes it difficult for non-expert users to express their analytical needs intuitively, thereby limiting their ability to extract meaningful insights from trajectory data.

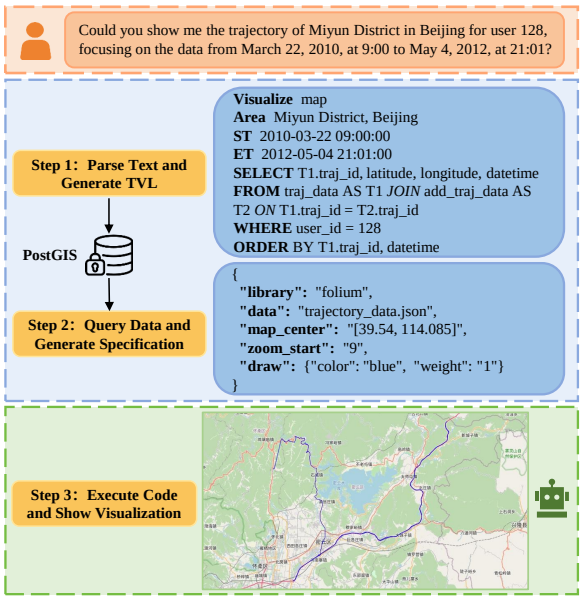


Figure 1: The process of transforming natural language questions into trajectory data visualizations.

Natural Language to Visualization (NL2VIS) technology focuses on the automatic translation of natural language questions (NLQs) into visualization specifications (Song et al., 2024, 2022). This capability enables non-expert users to more eas-

[*]Yuanfeng Song is the corresponding author.
[1]https://github.com/Text2Vis/dataset-TrajVL

Findings of the Association for Computational Linguistics: ACL 2026, pages 19456–19475

ily explore and visualize trajectory data (Lu et al., 2026; Song et al., 2026), providing new opportunities to address the challenges discussed above. In this work, we introduce the Text-to-TrajVis task. Under this setting, users express trajectory visualization requests in natural language, and the Text-to-TrajVis system parses these queries to extract key semantic elements, including geographic areas, temporal constraints, and query conditions. The NLQ is then transformed into the Trajectory Visualization Language (TVL), which serves as an intermediate representation for automatically generating executable visualization programs. These programs render the queried trajectory data as either map-based visualizations or statistical charts. Figure 1 illustrates an example in which a natural language query is converted into a map-based trajectory visualization, while the transformation processes for other chart types are detailed in Appendix A.1.

Large language models (LLMs) have shown remarkable generalization capabilities in the field of natural language processing. In addition to making breakthroughs in traditional tasks such as text generation and question-answering systems, LLMs have enabled the construction of knowledge indexing libraries through Retrieval-Augmented Generation (RAG) (Lewis et al., 2020), significantly improving the accuracy and domain adaptability of generation tasks. Furthermore, with task decomposition reasoning capabilities, LLMs can systematically break down complex tasks into well-defined subtasks (Khot et al., 2022) and efficiently organize and utilize retrieved knowledge. This structured approach significantly enhances both the comprehension of intricate queries and the quality of the final generated outputs. However, to the best of our knowledge, there is currently no dedicated NL2VIS dataset for trajectory data visualization that is suitable for benchmarking LLMs.

To address this gap, we introduce TrajVL, the first benchmark dataset for the Text-to-TrajVis task. We design a unified template and a systematic TVL generation pipeline with 3 stages. First, seed TVLs are created by instantiating the template with areas and time ranges. Second, we construct a constraint tree in which each leaf represents a specific query constraint or combination, and progressively augment TVLs to increase query diversity. Finally, we grouped 2–3 TVLs from the same geographic area into single units, treating each as a multi-trajectory visualization query. To annotate TVLs with natural language questions, we design two NLQ templates for general and complex queries involving geographic and temporal expressions. NLQs are automatically generated using an LLM and manually verified for semantic completeness and accuracy. For each multi-trajectory group, a single NLQ is assigned. The resulting benchmark contains 4,804 TVLs and 9,608 *(question, TVL)* pairs.

To enhance the accuracy of LLMs in generating TVL, we proposed a framework named TRCAT, which comprises two core modules: TVL-RAG Chain module and Area-Time Standardization module. The TVL-RAG Chain module incrementally transforms natural language questions into structured TVL through a four-stage pipeline: (i) visualization type recognition and geographic area analysis, (ii) SQL sketch construction with temporal structure, (iii) sketch parameter filling, and (iv) TVL structure integration and generation. The core of this module is the RAG mechanism, which retrieves semantically relevant examples based on input queries, and the decomposition step. This design significantly improves the accuracy and structural consistency of the generated TVL. To further improve robustness, TRCAT integrates the Area-Time Standardization module. This module employs a multi-stage resolution strategy that combines exact string matching and fuzzy semantic matching with a gazetteer. Additionally, the module leverages LLMs to parse diverse and ambiguous temporal expressions. Collectively, these components substantially improve the structural accuracy of the generated TVLs.

We evaluated TRCAT against the baseline method on TrajVL. Compared with the baseline approach, our method improves the accuracy of the best-performing model by 39.54%. Extensive experiments demonstrate that our approach significantly improves the performance of large language models on the Text-TrajVis task.

In summary, our contributions are as follows:

- We are the first to introduce the Text-to-TrajVis task, which aims to enable users to generate trajectory data visualizations directly from natural language questions.
- We construct TrajVL, the first benchmark dataset specifically designed for the Text-to-TrajVis task. It comprises 9,608 *(question, TVL)* pairs.
- We propose a novel framework called TRCAT,

the first framework for the Text-to-TrajVis task, which incrementally converts natural language questions into TVLs.

- We conducted extensive experiments on the proposed dataset, showing that TRCAT achieves state-of-the-art performance, outperforming baseline implementation with mainstream LLMs (e.g., GPT, Qwen, LLaMA, and Gemma) on the Text-to-TrajVis task.

2 Related Work

2.1 Trajectory Data Visualization

Basic trajectory visualization typically represents trajectories by connecting sampled points into polylines overlaid on map canvases, with GeoJSON (Butler et al., 2016) serving as the standard data format. Based on this representation, Folium (Suma et al., 2024) builds on Leaflet (Crickard III, 2014) to render GeoJSON as interactive web layers, while GeoPandas (Jordahl et al., 2021) abstracts geospatial processing by integrating Shapely for topology and Cartopy for map projections. Deep learning techniques have been introduced to support higher-level semantic analysis. For instance, DeepHL (Maekawa et al., 2020) applies attention-based models to automatically label anomalous trajectory segments and enables near real-time transformation of large-scale raw data into interactive 3D visualizations. Despite these advances, most trajectory visualization tools remain technically complex and difficult for non-expert users to access.

2.2 Natural Language to Data Visualization

The field of NL2VIS has received growing attention with the advancement of visualization technologies (Liu et al., 2021; Narechania et al., 2020; Luo et al., 2021b,a; Wan et al., 2025; Qin et al., 2025). Numerous studies leverage deep neural networks to learn mappings between natural language questions and visualization specifications (Dibia and Demiralp, 2019; Song et al., 2022; Liu et al., 2023; Li et al., 2023b; Xiang et al., 2023). To support this research, several datasets have been introduced—most notably NVBench (Luo et al., 2021a), which serves as a standard benchmark, and Dial-NVBench (Song et al., 2024), which extends the task to interactive scenarios. Recently, LLM-based methods have become the dominant paradigm (Gan et al., 2021; Dong et al., 2023; Li et al., 2023a; Pourreza and Rafiei, 2023; Gao et al., 2024), benefiting from the synergy of semantic parsing and code generation. Pre-trained LLMs like Chat2vis (Maddigan and Susnjak, 2023) and ChartGPT (Tian et al., 2024) demonstrate effective prompting and task-specific fine-tuning strategies (Chung et al., 2024). Emerging multi-agent LLM systems further improve NL2VIS by distributing tasks across specialized agents. MatPlotAgent (Yang et al., 2024b) and NVAgent (Ouyang et al., 2025) enhance accuracy and interpretability through collaborative workflows, while PlotGen (Goswami et al., 2025) and VisPath (Seo et al., 2025) integrate multimodal feedback to refine visual quality. However, the Text-to-TrajVis task remains unexplored. To address this gap, we propose the first systematic Text-to-TrajVis framework, accompanied by a dedicated dataset and a comprehensive evaluation on LLMs.

VISUALIZE ::= map | bar | line | pie | scatter | heat map | contour plot
AREA area ∈ ({geographical_name})
ST st ∈({null, timestamp})
ET et ∈({null, timestamp}, et > st (if st, et ≠ null))
SELECT C | CC | CCC | C...C (C ::= column)
FROM T
TRANSFORM ::= avg | count | sum | group by | binning | max | min
ORDER BY ::= asc C | desc C

Figure 2: Structure of Trajectory Visualization Language.

3 TrajVL Dataset

3.1 Trajectory Visualization Language

We designed a structured template for generating TVLs (Figure 2), which integrates four core components: (i) The VISUALIZE component defines the output visualization type and target rendering language (e.g., Python, Vega-Lite). (ii) Spatial dimensions are specified through the AREA parameter using geographical area names, while temporal dimensions employ a dual-anchor mechanism ([ST, ET]) to define closed intervals. (iii) The TRANSFORM operator implements trajectory-specific operations including temporal binning, spatial aggregation (GROUP BY), and basic statistics (AVG/-COUNT/SUM/MAX/MIN), formally expressed as $T(x) \rightarrow x'$. (iv) The ORDER BY extension enables sorting based on transformed variables.

3.2 Dataset Construction

Data Preparation The trajectory data used to construct TrajVL dataset is sourced from the open-source GPS trajectory dataset released by the GeoLife project (Zheng et al., 2011). This dataset comprises 17,621 trajectories collected from 182

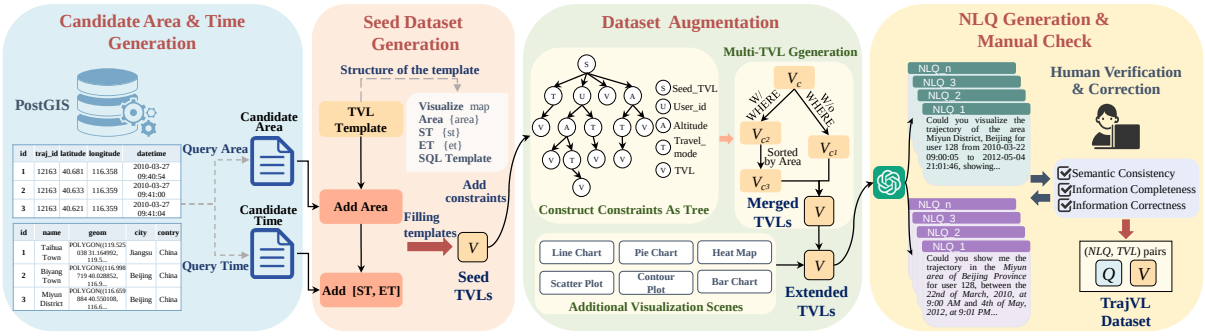


Figure 3: The pipeline of constructing the TrajVL dataset. The TrajVL dataset construction pipeline involves four main steps: (i) extraction of geographic areas and their associated temporal ranges from the trajectory database as candidates; (ii) generating representative TVLs as seed data by integrating geographic areas and temporal information into the structured template; (iii) enriching the dataset by expanding TVLs using a tree-structured augmentation approach, generating multi-trajectory queries and adding additional visualization scenarios; and (iv) generating multiple types of natural language questions using LLMs, with subsequent manual validation.

users, each trajectory as a time-ordered sequence of geospatial points with latitude, longitude, and altitude. The trajectories predominantly cover over 30 Chinese cities. For administrative boundary data of Chinese provinces at multiple levels, we employed Chinese area boundary data files provided by OpenStreetMap (Haklay and Weber, 2008) as the source. These boundary data were stored in a PostGIS-enabled spatial database for efficient geospatial queries.

Seed Dataset Generation The TrajVL construction pipeline is shown in Figure 3. As the trajectory data were stored in database, we first extracted all geographic areas within the trajectory data to validate them before incorporating them into the template. These collected geographic areas constituted the area candidates $G^* = \{g_1, g_2, \ldots, g_n\}$. Subsequently, we derived temporal ranges by identifying the earliest and latest timestamps of trajectories within each geographic area g_i. From each range, we selected 3-5 distinct time intervals to establish temporal candidates $T_i = \{\tau_1^{(i)}, \tau_2^{(i)}, \ldots, \tau_k^{(i)}\}$. The seed TVLs are generated via combinatorial selection of validated geographic areas and temporal intervals. This process can be represented as:

$$V_{\text{seed}} = \left\{ v_{ij} \mid g_i \in G^*,\ \tau_j^{(i)} \in T_i \right\} \tag{1}$$

Dataset Augmentation We designed a tree-structured constraint framework to enhance the expressiveness and diversity of TVLs. Beyond the fundamental latitude and longitude attributes in the primary table, the database contains auxiliary tables storing supplementary trajectory features. We systematically extracted and organized attributes including user ID, travel mode, and altitude to construct query constraints. These constraints form the hierarchy's leaf nodes $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_q\}$. By integrating these constraint nodes into TVL's SQL query framework, we generated an enriched dataset. This process can be represented as:

$$V_{\text{aug}} = V_{\text{seed}} \cup \left\{ v_{ij}^{\ell} = v_{ij} \wedge \ell \;\middle|\; v_{ij} \in V_{\text{seed}},\ \ell \in \mathcal{L} \right\} \tag{2}$$

Multi-Trajectory Generation Given a subset S, we proposed a composite TVL merging strategy $M(S)$ that combines multiple TVL queries into a single composite query containing 2-3 TVLs. This strategy follows the principle of merging only when a *WHERE* clause constraint w_i is present, ensuring that the merged natural language question retains the distinct semantics of each original trajectory. Formally, merging is performed only if all TVLs in S satisfy the following conditions: they share the same visualization type ν_i, correspond to the same geographic area $g_i = g^*$, and their corresponding SQL queries contain non-empty *WHERE* clauses. Conversely, any TVL with an empty *WHERE* clause is retained as an independent query and excluded from the merging process. This merging operation can be expressed as:

$$\begin{aligned} V_{\text{merge}} = &\left\{ M(S) \mid S \subseteq V_{\text{aug}},\ \nu_i = \textit{map},\ g_i = g^*, \right. \\ &\left. w_i \neq \emptyset,\ \forall v_i \in S \right\} \cup \left\{ v_j \in V_{\text{aug}} \mid w_j = \emptyset \right\} \end{aligned} \tag{3}$$

We found that complementary trajectory features, such as travel mode and altitude, can be utilized to construct novel visual query paradigms. Consequently, variations in travel mode and altitude within trajectories were selected as primary analytical dimensions and combined them with

geographic and temporal constraints to generate TVLs. This methodology was employed to generate queries for three common visualization types: pie charts, line charts, and bar charts, thereby expanding the dataset's coverage. This dataset can be extended to support other visualization types, including scatter plots and heat maps. Formally, let V_{feat} denote the set of those TVLs. The final TVL dataset is defined as the union of the previously merged set V_{merge} and this augmented set:

$$V^*_{\text{final}} = V_{\text{merge}} \cup V_{\text{feat}}. \quad (4)$$

Natural Language Questions Generation To reduce the cost and time overhead of manually constructing large-scale NLQs, we employ LLMs to automatically generate natural language expressions. Specifically, GPT-4o-mini is used with a temperature of 0.3 to balance generation diversity and consistency. Given the importance of geographic and temporal information in trajectory data, we adopt a sequential prompting strategy with two templates per TVL. These templates vary in area descriptions, temporal expressions, and sentence structures, enabling semantically diverse yet structurally faithful NLQs. For composite TVLs involving multiple trajectory subqueries, we design a specialized prompting strategy that explicitly extracts the geographic area, time range, and SQL filtering conditions of each subquery. These components are organized in parallel to preserve subquery independence and avoid predicate fusion, thereby ensuring that downstream systems can accurately reconstruct the original TVLs.

Manual Check To ensure the quality of NLQs generated from TVLs, we first categorized the dataset into three types based on query structure: single-trajectory, multi-trajectory, and other visualization types. A systematic human evaluation was subsequently conducted to assess the initial NLQ quality. We provide guidelines for human evaluation that focus on three key dimensions, namely semantic consistency, information completeness, and factual accuracy. Each instance was independently reviewed by two graduate students with expertise in data visualization and natural language processing. The inter-annotator agreement, measured by Cohen's Kappa (Vieira et al., 2010), reached 0.803, indicating substantial consensus. For defective NLQs identified during the manual check, we employed targeted prompting strategies (see Appendix A.4) to rectify them. A second round of manual verification was further performed to guarantee the correctness and clarity of the final NLQs.

Vis	TVL	TVL+	W/ Cons.	(NLQ, TVL)
Map	1,853	836	1,410	5,378
Bar	788	0	288	1,576
Line	727	0	727	1,454
Pie	600	0	268	1,200
Total	3,968	836	2,693	9,608

Table 1: Statistics of the TrajVL dataset. W/ Cons. denotes the TVLs with constraints, where TVL+ denotes the TVLs that support multi-trajectory query.

3.3 Dataset Analysis

The dataset includes four common visualization types: maps, bar charts, line charts, and pie charts, comprising a total of 4,804 visualization instances. Each instance is accompanied by two natural language questions, resulting in 9,608 *(question, TVL)* pairs (see Table 1). Maps account for 55.93% of the dataset, with single-trajectory queries representing 38.54% and multi-trajectory queries making up 17.39%, thus reflecting the predominance of maps in trajectory data visualization. In addition to maps, the dataset includes visual queries such as "Display the percentage of travel by each mode in Beijing over the past year" and "Display the change in altitude of the user's trajectory over a specified period". To capture such diversity, we expanded the number of bar charts, line charts, and pie charts, which account for 16.39%, 15.12%, and 12.48% of the dataset, respectively. Our dataset covers 9 provinces and includes 638 regions, with the majority situated in Hebei and Beijing.

4 TRCAT Framework

4.1 Pipeline of TRCAT

For a given NLQ, TRCAT encodes the query into an embedding vector using the *all-mpnet-base-v2* model and computes cosine similarities with precomputed embeddings of all stored NLQs. The top-K most relevant NLQs and their corresponding TVLs are retrieved to construct structured prior knowledge. This structured knowledge, along with the database schema, is subsequently passed to the LLM to generate a preliminary TVL. The Area-Time Standardization module further refines geographic area-related fields via a hybrid strategy that combines exact and fuzzy matching with a curated geographic gazetteer. It also employs LLMs to extract and normalize temporal expressions. The standardized geographic area name and temporal

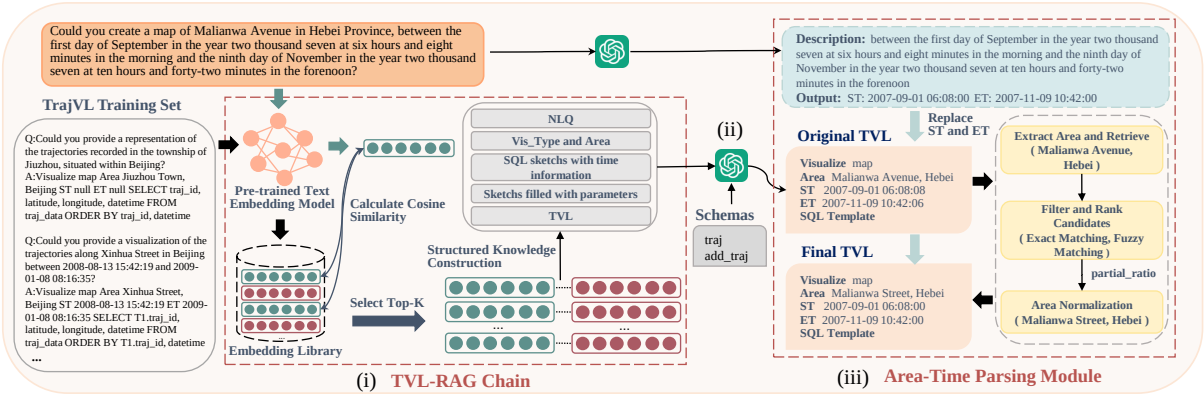


Figure 4: The working pipeline of our approach, which includes three steps: (i) Input NLQ into the retriever to obtain top-K NLQs and their associated TVLs, and generate structured knowledge based on these instances; (ii) Input structured knowledge and database schema into the LLM to generate TVL; (iii) Extract geographic area and time from the generated TVL, resolve them using Area-Time Standardization module to produce the final output.

constraint are reintegrated to form the final TVL. Figure 4 illustrates the TRCAT pipeline.

TVL-RAG Chain This module incorporates a RAG mechanism. For each input query, the system retrieves a set of semantically similar annotated examples from the knowledge base. Rather than directly reusing these examples, the TVL-RAG Chain module extracts structured knowledge aligned with the TVL format from the retrieved items, including visualization type, geographic area, abstract SQL sketches with temporal structure, parameter-filled sketches, and the complete TVL. These components are subsequently organized into a dynamic sequence of intermediate decomposition prompts, which serve as contextual demonstrations for the language model (Appendix A.5 provides complete implementation details). By leveraging these structurally consistent examples, the TVL-RAG Chain module guides LLMs to generate logically coherent and semantically faithful outputs that align with the user's query intent.

Area-Time Standardization Module To ensure accurate interpretation of geographic area names and temporal expressions in natural language queries, we introduce an Area–Time Standardization module integrated into the TVL-RAG Chain. This module operates after TVL generation to normalize geographic and temporal fields in the structured output. Specifically, the geographic area name is first extracted from the generated TVL. It is subsequently passed to a dedicated toponym resolver, which employs both exact string matching and fuzzy semantic similarity techniques against a curated geographic gazetteer. The extracted candidate toponyms are further filtered and ranked to identify the most plausible standardized toponym. The module also supports extracting temporal descriptions from NLQs and converting them into standardized start and end timestamps. It can identify and normalize time-related expressions, including implicit and ambiguous temporal ones. The standardized geographic and temporal information is then reintegrated into the final TVL. This post-processing step improves the robustness and executability of TVLs, particularly for vague or non-standard geographic and temporal expressions.

5 Experimental Setup

Baseline To comprehensively compare the performance of various large language models on the TVL generation task, we selected four representative models: GPT-4o-mini (Achiam et al., 2023), LLaMA3-8B (Grattafiori et al., 2024), Qwen2.5-7B (Yang et al., 2024a), and Gemma-7B (Team et al., 2024). Among these, GPT-4o-mini is a proprietary model, while LLaMA3-8B, Qwen2.5-7B, and Gemma-7B are open-source models. All baseline models were evaluated using their respective optimal few-shot prompting strategies (Brown et al., 2020). Appendix B.1 provides specifications of the models.

Metrics To evaluate LLM performance on TVL generation, we use three common NL2VIS metrics (Luo et al., 2021a): *Vis Accuracy (Vis.Acc)*, *Axis Accuracy (Axis.Acc)*, and *Data Accuracy (Datas.Acc)*. We also introduce *TVL Accuracy (TVL.Acc)* to evaluate TVL quality. Considering the geographic and temporal nature of trajectory data, Data Accuracy is further subdivided into Area Accuracy and Time Accuracy to evaluate geographic area and tempo-

Test Set	Model	Vis.Acc	Axis.Acc	Data.Acc			TVL.Acc	TVL.F1
				Area	Time	SQL		
$\text{TrajVL}_{\text{normal}}$	Gemma-7B	77.21%	76.12%	74.52%	76.06%	41.59%	39.41%	44.24%
	LLaMA3-8B	94.22%	93.65%	92.04%	92.30%	71.12%	68.87%	72.32%
	Qwen2.5-7B	95.70%	94.09%	94.29%	94.29%	70.80%	69.19%	72.74%
	GPT-4o-mini	97.05%	95.25%	95.70%	94.74%	77.34%	74.52%	77.32%
	Gemma-7B + Ours	86.65%	85.17%	86.20%	85.82%	67.01%	64.63%	70.85%
	LLaMA3-8B + Ours	98.10%	**97.37%**	97.75%	97.43%	81.51%	79.08%	81.41%
	Qwen2.5-7B + Ours	94.87%	93.13%	94.42%	94.16%	79.14%	76.77%	79.99%
	GPT-4o-mini + Ours	**98.14%**	95.83%	**97.82%**	**97.50%**	**83.76%**	**81.39%**	**83.41%**
$\text{TrajVL}_{\text{complex}}$	Gemma-7B	77.27%	71.31%	17.91%	61.68%	37.87%	7.25%	8.26%
	LLaMA3-8B	88.90%	88.32%	24.26%	78.75%	63.41%	15.21%	16.06%
	Qwen2.5-7B	94.35%	92.49%	26.12%	83.57%	69.13%	16.56%	18.23%
	GPT-4o-mini	95.89%	94.03%	26.83%	86.01%	74.07%	18.55%	17.83%
	Gemma-7B + Ours	83.76%	82.03%	57.64%	75.87%	64.57%	38.51%	42.59%
	LLaMA3-8B + Ours	97.43%	**96.73%**	**76.32%**	88.51%	80.36%	**54.75%**	**57.08%**
	Qwen2.5-7B + Ours	96.85%	94.87%	67.33%	87.87%	78.69%	48.27%	50.83%
	GPT-4o-mini + Ours	**97.95%**	95.51%	70.67%	**89.28%**	**83.25%**	53.59%	55.16%

Table 2: Performance comparison of various LLMs under different settings on the TrajVL dataset.

ral dimensions. For the generated multi-trajectory TVL queries, as changes in TVL order may lead to misjudgments in exact matching methods, we adopt *F1-score (TVL.F1)* as the primary evaluation metric to reflect the model's performance in multi-trajectory semantic generation tasks. Detailed metric definitions are provided in Appendix B.2.

Dataset Splitting TrajVL is split into 1,746 TVLs for training, 1,500 for validation, and 1,558 for testing. The validation set is used for few-shot parameter tuning. The test set is constructed via stratified sampling over visualization types and constraint patterns to ensure semantic and structural diversity while reducing bias toward specific query forms. To assess generalization over geographic entities, all geographic area names appearing in the test set are excluded from the training set. For each test TVL, two semantically distinct NLQs are generated, resulting in two evaluation subsets: a standard test set ($\text{TrajVL}_{\text{normal}}$) and a Spatiotemporal complexity set ($\text{TrajVL}_{\text{complex}}$), with the latter emphasizing queries involving challenging geographic and temporal expressions.

Implementation Details We employed three open-source large language models (LLaMA3-8B (Grattafiori et al., 2024), Qwen2.5-7B (Yang et al., 2024a), and Gemma-7B (Team et al., 2024)) as well as one proprietary model (GPT-4o-mini (Achiam et al., 2023)). During the inference stage, the temperature parameter for all LLMs was set to 0.1 and the max_length was set to 2048 tokens.

6 Results and Discussion

6.1 Performance Comparison

To comprehensively evaluate the performance of four LLMs and our proposed method on the TrajVL dataset, we conducted experiments on two test sets: $\text{TrajVL}_{\text{normal}}$ and $\text{TrajVL}_{\text{complex}}$. The results are presented in Table 2.

On $\text{TrajVL}_{\text{normal}}$, the baseline GPT-4o-mini achieved the highest TVL accuracy of 74.52%. Among open-source models, Qwen2.5 slightly outperformed LLaMA3, with accuracies of 69.19% and 68.87%, respectively, while Gemma performed substantially worse at 39.41%. Performance degraded markedly on the more challenging $\text{TrajVL}_{\text{complex}}$, which contains complex spatial and temporal expressions. TVL accuracy dropped by 55.97 percentage points for GPT-4o-mini, 53.66 for LLaMA3, and 32.16 for Gemma, highlighting the increased difficulty introduced by ambiguous geographic and temporal descriptions.

Applying the proposed TRCAT framework consistently improved performance across all models. On $\text{TrajVL}_{\text{normal}}$, GPT-4o-mini with TRCAT reached 81.39% TVL accuracy, a 6.87 percentage point gain over the baseline, while all open-source models also showed clear improvements. Notably, TRCAT enabled LLaMA3 to surpass Qwen2.5, and boosted Gemma to 64.63%, although it remained behind other models.

The gains from TRCAT were more pronounced on $\text{TrajVL}_{\text{complex}}$. GPT-4o-mini improved by 35.04 percentage points to 53.59% TVL accuracy.

LLaMA3, Qwen2.5, and Gemma achieved improvements of 39.54, 31.71, and 31.26 percentage points, respectively. Under this setting, LLaMA3 outperformed Qwen2.5 by 6.48 percentage points and even surpassed GPT-4o-mini. This suggests that TRCAT effectively enhances LLaMA3's ability to interpret implicit multi-trajectory structures in NLQs, leading to more accurate generation of multiple TVLs. Overall, these results demonstrate TRCAT's robustness in handling vague and complex geographic and temporal expressions.

Beyond TVL accuracy, TRCAT outperformed all baselines on Vis.Acc, Axis.Acc, and Data.Acc, indicating strong robustness in identifying visualization types and layout components. Axis.Acc was consistently higher on $\text{TrajVL}_{\text{normal}}$ than on $\text{TrajVL}_{\text{complex}}$, due to the strong dependence of map-based visualizations on precise spatial descriptions. In addition, TRCAT improved SQL.Acc by leveraging the TVL-RAG Chain, which decomposes generation into sketch construction and parameter filling, enabling LLMs to better capture implicit SQL structures in NLQs.

6.2 Parameter Study

To examine how the number of retrieved exemplars affects the performance of TRCAT and baselines, we conducted experiments using TVL.Acc as the evaluation metric. Under the baseline setting on $\text{TrajVL}_{\text{complex}}$, GPT-4o-mini, Qwen2.5, LLaMA3, and Gemma achieved their best accuracies with 6-shot or 7-shot configuration. Under the TRCAT setting, GPT-4o-mini and LLaMA3 performed best with 5-shot or 6-shot configuration, Qwen2.5 and Gemma again with 3-shot configuration. Increasing the number of exemplars beyond these optimal values led to performance degradation, likely due to excessive context length hindering reasoning. All reported results correspond to each model's best-performing few-shot setting. Accuracy curves are provided in Appendix B.3.

6.3 Ablation Study

We conduct an ablation study to assess the contribution of each component in TRCAT. The full framework with all modules enabled is first evaluated, followed by configurations with individual components removed: (i) without the TVL-RAG Chain and Area-Time Standardization modules (w/o TRC&AT); (ii) without the TVL-RAG Chain module (w/o TRC); and (iii) without the Area-Time Standardization module (w/o AT). All configurations use the same number of demonstration examples as the full TRCAT to ensure fair comparison.

Model	Setting	Area	TVL	TVL+
Gemma	TRCAT	**57.64%**	**38.51%**	**22.14%**
	- w/o TRC&AT	17.91%	7.25%	4.80%
	- w/o TRC	41.34%	17.59%	6.27%
	- w/o AT	32.80%	20.47%	15.50%
LLaMA3	TRCAT	**76.32%**	**54.75%**	**53.14%**
	- w/o TRC&AT	24.26%	15.21%	16.97%
	- w/o TRC	54.81%	34.85%	25.83%
	- w/o AT	54.04%	37.87%	43.91%
Qwen2.5	TRCAT	**67.33%**	**48.27%**	**42.07%**
	- w/o TRC&AT	26.12%	16.56%	18.08%
	- w/o TRC	43.26%	27.54%	20.3%
	- w/o AT	32.86%	23.36%	28.41%
GPT-4o-mini	TRCAT	**70.67%**	**53.59%**	**55.35%**
	- w/o TRC&AT	26.83%	18.55%	26.57%
	- w/o TRC	59.69%	40.82%	39.85%
	- w/o AT	40.31%	31.19%	40.96%

Table 3: Ablation study results on $\text{TrajVL}_{\text{complex}}$.

The results, presented in Table 3, validate the importance of both modules in the proposed framework. We observe that the TVL-RAG Chain module plays a critical role in the Text-to-TrajVis task, as it guides LLMs to align their output structure with the TVL format, thereby enhancing the accuracy of the generated TVLs. It also enhances the LLM's ability to interpret the implicit multi-trajectory semantics embedded in the NLQ, improving multi-TVL generation accuracy. Furthermore, we observe that the decomposition step enables LLMs to better identify the underlying SQL structure within NLQ, thus improving the accuracy of SQL generation. The Area-Time Standardization module is essential for normalizing geographic and temporal entities. It standardizes geographic area names that LLMs may fail to recognize and converts complex temporal expressions into precise timestamps, thereby improving the accuracy of area and time parameters in TVLs.

7 Conclusion

We introduce TrajVL, the first benchmark dataset for the Text-to-TrajVis task, and design TVL, a visualization language for trajectory data querying and visualization. TrajVL is constructed through template-based seed TVL generation, tree-structured constraint augmentation, and multi-trajectory query grouping. Using LLMs for NLQ generation with human verification, we obtained 9,608 *(question, TVL)* pairs. To enhance TVL generation, we propose TRCAT, a framework inte-

grating TVL-RAG Chain module and Area-Time Standardization module, which enhances accuracy and structural consistency. Experimental results demonstrate that TRCAT-enhanced LLMs outperform baseline models, especially for complex geographic and temporal expressions. Future work will aim to improve end-to-end TVL generation under complex geographic and temporal constraints, further advancing trajectory visualization research.

Limitations

We propose a dataset generation framework that integrates LLMs with human intervention, and construct the first large-scale Text-to-TrajVis benchmark dataset (dubbed TrajVL) to address the scarcity of benchmark datasets in this domain. However, the trajectory data used to construct the dataset primarily derives from the GeoLife Project Dataset. Although the dataset was effectively expanded through a systematic approach, the insufficient volume of auxiliary information associated with the trajectory data limited the complexity of constraint trees. Thus, future research should incorporate more comprehensive auxiliary information related to trajectories and construct more complex constraint trees to expand the TrajVL dataset. Additionally, more diverse and complex spatiotemporal descriptions of real-world trajectories should be incorporated into the NLQ generation process to enhance the dataset's difficulty.

Acknowledgments

This work is supported by the National Natural Science Foundation of China [62576149]. The research of Victor Junqiu WEI was supported in part by the Guangdong Basic and Applied Basic Research Foundation (Grant No.: GDST23EG32, Project No.: 2024A1515011667), Macau Science and Technology Development Fund (FDCT-25-074-SCSE, Project No.: 0094/2025/ITP2) and Faculty Research Grant of MUST (FRG-25-080-FIE).

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Hossein Amiri, Ruochen Kong, and Andreas Züfle. 2024a. Urban anomalies: A simulated human mobility dataset with injected anomalies. In *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Geospatial Anomaly Detection*, pages 1–11.

Hossein Amiri, Richard Yang, and Andreas Züfle. 2024b. Geolife+: Large-scale simulated trajectory datasets calibrated to the geolife dataset. In *Proceedings of the 7th ACM SIGSPATIAL International Workshop on GeoSpatial Simulation*, pages 25–28.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Howard Butler, Martin Daly, Allan Doyle, Sean Gillies, Stefan Hagen, and Tim Schaub. 2016. The geojson format. Technical report.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, and 1 others. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

Paul Crickard III. 2014. *Leaflet. js essentials*. Packt Publishing Ltd.

Victor Dibia and Çağatay Demiralp. 2019. Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks. *IEEE computer graphics and applications*, 39(5):33–46.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, and 1 others. 2023. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306*.

Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R Woodward, John Drake, and Qiaofu Zhang. 2021. Natural sql: Making sql easier to infer from natural language specifications. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2030–2042.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-sql empowered by large language models: A benchmark evaluation. *Proceedings of the VLDB Endowment*, 17(5):1132–1145.

Kanika Goswami, Puneet Mathur, Ryan Rossi, and Franck Dernoncourt. 2025. Plotgen: Multi-agent llm-based scientific data visualization via multimodal retrieval feedback. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 1672–1676.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Mordechai Haklay and Patrick Weber. 2008. Openstreetmap: User-generated street maps. *IEEE Pervasive computing*, 7(4):12–18.

Kelsey Jordahl, Joris Van den Bossche, Jacob Wasserman, James McBride, Jeffrey Gerard, Jeff Tratner, Matthew Perry, and Carson Farmer. 2021. geopandas/geopandas: v0. 5.0. *Zenodo*.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*.

Will Kohn, Hossein Amiri, and Andreas Züfle. 2023. Epipol: An epidemiological patterns of life simulation (demonstration paper). In *Proceedings of the 4th ACM SIGSPATIAL International Workshop on Spatial Computing for Epidemiology*, pages 13–16.

Maximilian Konzack, Pieter Gijsbers, Ferry Timmers, Emiel van Loon, Michel A Westenberg, and Kevin Buchin. 2019. Visual exploration of migration patterns in gull data. *Information Visualization*, 18(1):138–152.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075.

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023b. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 13076–13084.

Can Liu, Yun Han, Ruike Jiang, and Xiaoru Yuan. 2021. Advisor: Automatic visualization answer for natural-language question on tabular data. In *2021 IEEE 14th Pacific Visualization Symposium (PacificVis)*, pages 11–20. IEEE.

Hu Liu, Yuliang Shi, Jianlin Zhang, Xinjun Wang, Hui Li, and Fanyu Kong. 2023. Multi-hop relational graph attention network for text-to-sql parsing. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Jinwei Lu, Yuanfeng Song, Chen Zhang, and Raymond Chi-Wing Wong. 2026. Multivis-agent: A multi-agent framework with logic rules for reliable and comprehensive cross-modal data visualization. *Proc. ACM Manag. Data*, 4(1).

Yuyu Luo, Nan Tang, Guoliang Li, Chengliang Chai, Wenbo Li, and Xuedi Qin. 2021a. Synthesizing natural language to visualization (nl2vis) benchmarks from nl2sql benchmarks. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1235–1247.

Yuyu Luo, Nan Tang, Guoliang Li, Jiawei Tang, Chengliang Chai, and Xuedi Qin. 2021b. Natural language to visualization by neural machine translation. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):217–226.

Paula Maddigan and Teo Susnjak. 2023. Chat2vis: Generating data visualizations via natural language using chatgpt, codex and gpt-3 large language models. *Ieee Access*, 11:45181–45193.

Takuya Maekawa, Kazuyoshi Ohara, Yizhe Zhang, Masaaki Fukutomi, Satoshi Matsumoto, Kenta Matsumura, Hajime Shidara, Shoji J Yamazaki, Ryoya Fujisawa, Keisuke Ide, Naoki Nagaya, Koji Yamazaki, Shinsuke Koike, Takashi Miyatake, Koutarou D Kimura, Haruko Ogawa, Shun Takahashi, and Ken Yoda. 2020. Deep learning-assisted comparative analysis of animal trajectories with DeepHL. *Nature Communications*, 11(1):5316.

Arpit Narechania, Arjun Srinivasan, and John Stasko. 2020. Nl4dv: A toolkit for generating analytic specifications for data visualization from natural language queries. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):369–379.

Geliang Ouyang, Jingyao Chen, Zhihe Nie, Yi Gui, Yao Wan, Hongyu Zhang, and Dongping Chen. 2025. nvagent: Automated data visualization from natural language via collaborative agent workflow. *arXiv preprint arXiv:2502.05036*.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36:36339–36348.

Zhiqian Qin, Yuanfeng Song, Jinwei Lu, Yuanwei Song, Shuaimin Li, and Chen Jason Zhang. 2025. MultiTEND: A multilingual benchmark for natural language to NoSQL query translation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24632–24657, Vienna, Austria. Association for Computational Linguistics.

Wonduk Seo, Seungyong Lee, Daye Kang, Zonghao Yuan, and Seunghyun Lee. 2025. Vispath: Automated visualization code synthesis via multi-path reasoning and feedback-driven optimization. *arXiv e-prints*, pages arXiv–2502.

Yuanfeng Song, Jinwei Lu, and Raymond Chi-Wing Wong. 2026. Covis: Neural and llm-driven multi-turn interactions for conversational text-to-visualization generation. *The VLDB Journal*, 35.

Yuanfeng Song, Xuefang Zhao, and Raymond Chi-Wing Wong. 2024. Marrying dialogue systems with

data visualization: Interactive data visualization generation from natural language conversations. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2733–2744.

Yuanfeng Song, Xuefang Zhao, Raymond Chi-Wing Wong, and Di Jiang. 2022. Rgvisnet: A hybrid retrieval-generation neural framework towards automatic data visualization generation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1646–1655.

KG Suma, Gurram Sunitha, J Avanija, Mohammad Gouse Galety, and Chinthapatla Pranay Varna. 2024. Geospatial data visualization with folium. In *Geospatial Application Development Using Python Programming*, pages 187–208. IGI global.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, and 1 others. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Yuan Tian, Weiwei Cui, Dazhen Deng, Xinjing Yi, Yurun Yang, Haidong Zhang, and Yingcai Wu. 2024. Chartgpt: Leveraging llms to generate charts from abstract natural language. *IEEE Transactions on Visualization and Computer Graphics*.

Eran Toch, Boaz Lerner, Eyal Ben-Zion, and Irad Ben-Gal. 2019. Analyzing large-scale human mobility data: a survey of machine learning methods and applications. *Knowledge and Information Systems*, 58:501–523.

Susana M Vieira, Uzay Kaymak, and João MC Sousa. 2010. Cohen's kappa coefficient as a performance measure for feature selection. In *International conference on fuzzy systems*, pages 1–8. IEEE.

Zhuoyue Wan, Yuanfeng Song, Shuaimin Li, Chen Jason Zhang, and Raymond Chi-Wing Wong. 2025. Datavist5: A pre-trained language model for jointly understanding text and data visualization. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, pages 1704–1717.

Yanzheng Xiang, Qian-Wen Zhang, Xu Zhang, Zejie Liu, Yunbo Cao, and Deyu Zhou. 2023. G3r: A graph-guided generate-and-rerank framework for complex and cross-domain text-to-sql generation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 338–352.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024a. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Zhiyu Yang, Zihan Zhou, Shuo Wang, Xin Cong, Xu Han, Yukun Yan, Zhenghao Liu, Zhixing Tan, Pengyuan Liu, Dong Yu, and 1 others. 2024b. Matplotagent: Method and evaluation for llm-based agentic scientific data visualization. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 11789–11804.

Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. 2010. T-drive: driving directions based on taxi trajectories. In *ACM SIGSPATIAL International Workshop on Advances in Geographic Information Systems*.

Yu Zheng, Hao Fu, Xing Xie, Wei Ying Ma, and Quannan Li. 2011. Geolife gps trajectory dataset - user guide.

Feng Zhu, Chen Chang, Zhiheng Li, Boqi Li, and Li Li. 2024a. A generic optimization-based enhancement method for trajectory data: Two plus one. *Accident; analysis and prevention*, 200:107532.

Yuanshao Zhu, James Jianqiao Yu, Xiangyu Zhao, Xuetao Wei, and Yuxuan Liang. 2024b. Unitraj: Learning a universal trajectory foundation model from billion-scale worldwide traces. *CoRR*.

A Implementation Details

A.1 Visualization Generation Examples

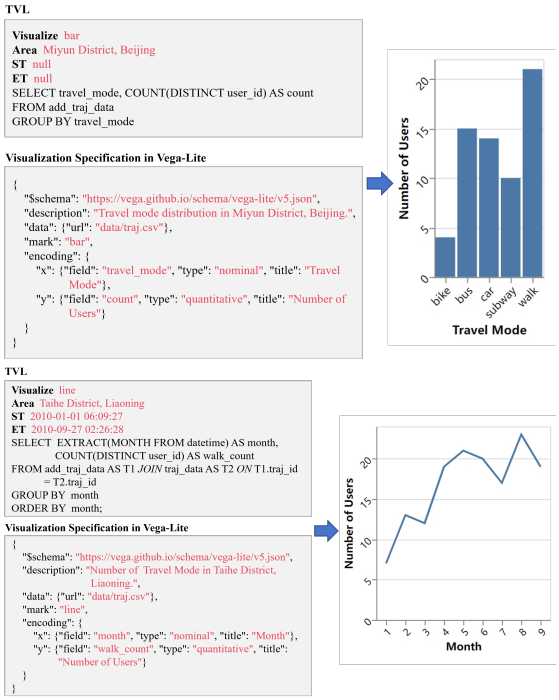


Figure 5: Examples of the visualization specification in Vega-Lite, and their corresponding visualizations.

The proposed Text-to-TrajVis system implements an automatic mapping mechanism from natural language questions to TVLs. As shown in Figure 5, the generated TVL contains three core components: (i) visualization type statement: specifying the chart presentation form; (ii) geographic and temporal constraint parameters: constructing geographic and temporal filtering conditions; and (iii) SQL query statement: realizing semantic fusion of trajectory attributes and geographic and temporal coordinates through multi-table association. For standardized chart types, including but not limited to bar charts, line charts, and pie charts, the system employs Vega-Lite, a state-of-the-art declarative visualization specification framework. By leveraging the widely adopted JSON syntax, Vega-Lite allows for the precise specification of visual coding rules, such as mapping data attributes to visual channels (e.g., position, color, size) and defining the overall chart structure. The proposed architecture ensures end-to-end interpretability from semantic parsing to visual presentation through formal language transformation mechanisms.

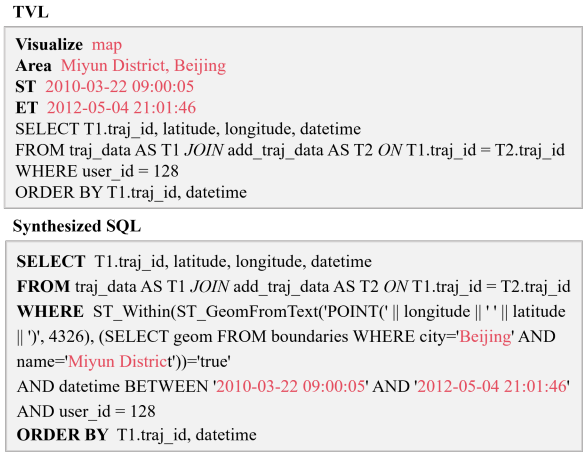


Figure 6: An example of synthesizing SQL based on TVL.

A.2 Synthesizing SQL from TVL

Given the TVL specification, the system achieves translation to SQL by parsing the logical structure inherent in TVL. During this conversion, geographic and temporal constraints declared within the TVL are dynamically mapped to the WHERE clause of the resulting SQL query, specifically encompassing the core parameters: Area, ST (Start Time), and ET (End Time). For instance, when TVL specifies that the value of Area is *"Miyun District, Beijing"*, ST is *"2010-03-22 09:00:00"*, ET is *"2012-05-04 21:01:00"*, and the basic SQL structure is "*WHERE ST_Within(ST_GeomFromText('POINT(' || longitude || ' ' || latitude || ')', 4326), (SELECT geom FROM boundaries WHERE city='Beijing' AND name='Miyun District'))='true' AND datetime BETWEEN '2010-03-22 09:00:05' AND '2012-05-04 21:01:46'* ". Using these parameters, the system is able to dynamically generate the SQL required to query the data, as shown in Figure 6.

A.3 Prompts for Generating NLQs

Prompt for generating multi-trajectory queries

```
Please write a single NLQ whose intent
covers all of the above TVLs.

## Special Rules
1. The resulting NLQ must be fluent, match
the user's tone and consistent with the
real world.
2. The NLQ is a fluent user query in one
sentence, don't appear to be listed in
separate entries.
3. The natural language question must
contain the intent of all TVLs.
4. The NLQ must accurately preserve all
distinct constraints from each TVL, the
```

```
same constraints can be merged.
5. The goal is for a downstream system to
reconstruct all original TVLs from a single
 structured NLQ, without merging or
confusing their individual logic.
6. The geographic area name and time in the
 natural language question need to be
consistent with the TVL and do not need to
be described differently.

Here are some examples:
TVL 1: Visualize map Area Baiyun Street,
Shandong ST 2008-08-21 12:42:33 ET
2009-09-12 01:50:33 SELECT T1.traj_id,
latitude, longitude, datetime FROM
traj_data AS T1 JOIN add_traj_data AS T2 ON
 T1.traj_id = T2.traj_id WHERE user_id =
140 ORDER BY T1.traj_id, datetime;
TVL 2: Visualize map Area Baiyun Street,
Shandong ST 2008-08-21 12:42:33 ET
2009-09-12 01:50:33 SELECT T1.traj_id,
latitude, longitude, datetime FROM
traj_data AS T1 JOIN add_traj_data AS T2 ON
 T1.traj_id = T2.traj_id WHERE user_id =
180 AND altitude BETWEEN -9725.41 AND
102482.8 ORDER BY T1.traj_id, datetime;
Generated NLQ:
Please show me the trajectory data for the
area Baiyun Street, Shandong from
2008-08-21 12:42:33 to 2009-09-12 01:50:33,
 including trajectories of user 140, and
those of user 180 with altitude between
-9725.41 and 102482.8?

Now please generate the NLQ based on the
given TVLs:

TVL: {tvl}
Generated NLQ:
```

Prompt for generating normal data

```
Please generate a NLQ for the given TVL.

## Special Rules
1. The resulting NLQ must be fluent, match
the user's tone and consistent with the
real world.
2. The natural language question must
contain the intent of all TVLs.
3. The geographic area name and time in the
 natural language question need to be
consistent with the TVL and do not need to
be described differently.

Now please generate the NLQ based on the
given TVL:
TVL: {tvl}
Generated NLQ:
```

Prompt for different descriptions of geographic area and time

```
You are a helpful assistant to complicate
the description of area name and time
strictly according to the user's
requirements.

## Requirements:
1. First, you must ensure that you do not
change the description of any information
in the base NLQ other than the area name
and time.
2. Complicate the area name descriptions in
 a way that is different from the base NLQ,
 but make sure they still represent the
same area name.
3. If there is time information in the base
 NLQ, complicate the time descriptions in a
 way that is different from the base NLQ,
but ensure that they still represent the
same time.
4. To ensure that you do not modify the
information in the base NLQ, it is
recommended that you first extract the area
 name in the base NLQ, complicate the area
name, and then replace the area name in the
 base NLQ.

Base NLQ:
{nlq}
Modified NLQ:
```

A.4 Prompts for Correcting NLQs

Prompt for correcting semantic redundancy

```
There is a problem with the description of
the NLQ in the example data, the generated
description contains information that is
not relevant to the TVL. Please examine the
 original NLQ carefully, simplify that NLQ
and save the information describing only
the individual fields in the TVL, correct
it, and then regenerate a new NLQ to ensure
 that the information described is not
redundant (Note: the answer just outputs
each NLQ without additional text).

Original NLQ:
{nlq}
Revised NLQ:
```

Prompt for correcting information missing

```
There is a problem with the description of
the NLQ in the example data, the
description information generated based on
the TVL is incomplete with missing
information. Please regenerate a new NLQ
based on the original NLQ and make sure
that the description information is
complete with all the information in the
TVL (Note: the answer just outputs each NLQ
 without additional text).

Original NLQ:
{nlq}
Revised NLQ:
```

Prompt for correcting information error

```
There is a problem with the description of
the NLQ in the example data, the generated
description information is inconsistent
with the TVL in fields such as time, area,
and SQL. Please examine the original NLQ
carefully, find the inconsistent
description, fix it, and regenerate a new
```

```
NLQ to ensure that the information in the
description is consistent with the
information in the TVL (Note: the answer
just outputs each NLQ without additional
text).

Original NLQ:
{nlq}
Revised NLQ:
```

A.5 Implementation Details of TVL-RAG Chain

```
## Question
Please provide the trajectory data for the
area Jingshan Street, Beijing from
2008-04-13 09:45:34 to 2010-01-04 05:44:45,
 including trajectories of user 17 on foot
and those of user 2 within an altitude
range between -12875.79 and 5657.95.

Decompose the task into sub tasks,
considering [Background] [Special Rules] [
Constraints], and generate the TVL after
thinking step by step:

**Sub task 1:** Generate visualize type and
 area name
By parsing the Question, we generated the
visualize type and standardized area name.
Visualize type: map, area name: Jingshan
Street, Beijing

**Sub task 2:** Construct sketches
By parsing the question, we found that 3
specific sketches with temporal structure
need to be constructed for the Question.
Sketch 1:
ST _ ET _ SELECT _ , _ , _ , _ FROM _ JOIN
_ ON _ WHERE _ ORDER BY _
Sketch 2:
ST _ ET _ SELECT _ , _ , _ , _ FROM _ JOIN
_ ON _ WHERE _ ORDER BY _

**Sub task 3:** Fill sketches
By parsing the question, we extracted
relevant information and filled them in the
 sketches to generate 3 complete SQL
queries with temporal information.
SQL 1:
ST 2008-04-13 09:45:34 ET 2010-01-04
05:44:45 SELECT T1.traj_id, latitude,
longitude, datetime FROM traj_data AS T1
JOIN add_traj_data AS T2 ON T1.traj_id = T2.
traj_id WHERE user_id = 17 AND travel_mode
= 'walk' ORDER BY T1.traj_id, datetime;
SQL 2:
ST 2008-04-13 09:45:34 ET 2010-01-04
05:44:45 SELECT T1.traj_id, latitude,
longitude, datetime FROM traj_data AS T1
JOIN add_traj_data AS T2 ON T1.traj_id = T2.
traj_id WHERE user_id = 2 AND altitude
BETWEEN -12875.79 AND 5657.95 ORDER BY T1.
traj_id, datetime;

**Sub task 4:** Combine into Final TVL
Add visualize type and area name to the SQL
 queries with time information to generate
the final TVL.
Final TVL:
TVL1:
Visualize map Area Jingshan Street, Beijing
 ST 2008-04-13 09:45:34 ET 2010-01-04
05:44:45 SELECT T1.traj_id, latitude,
longitude, datetime FROM traj_data AS T1
JOIN add_traj_data AS T2 ON T1.traj_id = T2.
traj_id WHERE user_id = 17 AND travel_mode
= 'walk' ORDER BY T1.traj_id, datetime;
TVL2:
Visualize map Area Jingshan Street, Beijing
 ST 2008-04-13 09:45:34 ET 2010-01-04
05:44:45 SELECT T1.traj_id, latitude,
longitude, datetime FROM traj_data AS T1
JOIN add_traj_data AS T2 ON T1.traj_id = T2.
traj_id WHERE user_id = 2 AND altitude
BETWEEN -12875.79 AND 5657.95 ORDER BY T1.
traj_id, datetime;
```

For a given NLQ, the TVL-RAG Chain first retrieves the top-k most similar NLQs from an embedding library and uses each retrieved NLQ as the Question in an in-context example. Since every NLQ is paired with a TVL, we decompose the corresponding TVL into structured components: (i) visualization type and geographic area name, (ii) SQL sketches with temporal structure, (iii) information-filled sketches, and (iv) the final TVLs. These structured elements guide the LLM to internalize the target TVL format and generate correct outputs step by step. All constructed examples are then embedded into the prompt and fed to the LLM.

B Experimental Details

B.1 Baselines

We evaluated TRCAT and baseline methods on LLMs using the TrajVL dataset, with the baseline methods and LLMs listed as follows:

- **Few-shot LLM**: The few-shot prompting is a core mechanism in In-Context Learning, using a limited set of examples to guide LLMs in performing tasks within specialized domains.
- **LLaMA3-8B**: LLaMA3-8B, an 8-billion-parameter open-weight language model developed by Meta, is optimized for high efficiency while maintaining robust performance across diverse NLP tasks. Pretrained on a large-scale multilingual corpus, it demonstrates strong reasoning and text-generation capabilities, positioning it as a versatile choice for applications requiring a balance between model size and inference speed.
- **Qwen2.5-7B**: Qwen2.5-7B, a 7-billion-parameter large language model developed

by Alibaba Group, is engineered to deliver competitive performance in both general and domain-specific tasks. Trained on diverse multilingual and multimodal corpora, it excels in natural language understanding, generation, and code-related tasks, while maintaining computational efficiency.

- **Gemma-7B**: Gemma-7B, an 7-billion-parameter large language model developed by Google's Gemma family, a series of lightweight, state-of-the-art open-source models built on the same research and technologies used to develop the Gemini models. As text-to-text decoder-based large language models, Gemma models are available in English, with both open-weight pre-trained and instruction-tuned variants. They are well-suited for a variety of text generation tasks, including question answering, summarization, and reasoning. Thanks to their relatively small size, Gemma models are also optimized for deployment in resource-constrained environments.

- **GPT-4o-mini**: GPT-4o-mini, a smaller version in OpenAI's GPT-4o series, leverages advanced architectures and training methodologies for strong performance on diverse NLP tasks. Designed for efficiency and responsiveness, it offers a balance between high-quality generation and reduced latency, making it suitable for applications requiring both sophisticated language understanding and rapid response times.

B.2 Metrics

We use the same metrics as NVBench, including Vis Accuracy, Axis Accuracy, and Data Accuracy, to assess LLM performance in TVL generation. Since geographic and temporal information is important for trajectory data, SQL statements for trajectory queries are synthesized based on geographic area, time and basic SQL query structure. Consequently, Data Accuracy is decomposed into Area Accuracy, Time Accuracy, SQL Accuracy and TVL Accuracy. For the generated multi-trajectory TVL queries, we adopt F1-score as the primary evaluation metric.

The evaluation metrics are formally defined as follows:

- **Vis Accuracy (Vis.Acc)**: This metric evaluates the accuracy of the Visualize component of the TVL generated by the model, measuring the model's ability to identify the type of visualization. It is calculated as:

$$Acc_{type} = \frac{N_{type}}{N}$$

Where N_{type} represents the count of exact match visualization types, N represents the total number of visualization types in the test set.

- **Axis Accuracy (Axis.Acc)**: This metric evaluates the performance of the model in recognizing vis components. For map visualizations, it evaluates the accuracy of the Area generated by the model, since proper visualization requires centering of the specified geographic area for map visualizations. For the x-axis and y-axis components of other types of visualizations such as bar, line, pie, etc., we measure the Select component of the vis query. It is calculated as:

$$Acc_{comp} = \frac{N_{comp}}{N}$$

Where N_{comp} represents the count of exact matches to the Area or Select component, N represents the total number of queries in the test set.

- **Area Accuracy (Area.Acc)**: This metric evaluates the accuracy of the Area component of the TVLs generated by the model to reveal the model's performance in recognizing geographical area names. It is calculated as:

$$Acc_{area} = \frac{N_{area}}{N}$$

Where N_{area} represents the count of exact matching Area components, N represents the total number of queries in the test set.

- **Time Accuracy (Time.Acc)**: This metric evaluates the accuracy of the Time component of the TVLs generated by the model to reveal the model's performance in recognizing temporal information. It is calculated as:

$$Acc_{time} = \frac{N_{time}}{N}$$

Where N_{time} represents the count of exact matching Time components, N represents the total number of queries in the test set.

- **SQL Accuracy (SQL.Acc)**: This metric evaluates the accuracy of the model in generating critical SQL by parsing and comparing SQL query structures in TVL. Specifically, it eliminates interference from syntactically equivalent but ordering differences through normalization (e.g., alphabetizing logical conditions in WHERE clauses), and recursively compares SQL structures through parse trees. It is calculated as:

$$Acc_{sql} = \frac{N_{sql}}{N}$$

Where N_{sql} represents the count of exact matching SQL queries, N represents the total number of SQL queries in the test set.

- **TVL Accuracy (TVL.Acc)**: This metric evaluates the accuracy of the model in generating a complete TVL. Based on the structure of the TVL, a model is considered to be able to generate the TVL correctly when it correctly generates Visualize, Area, Time and SQL. It is calculated as:

$$Acc_{tvl} = \frac{N_{tvl}}{N}$$

Where N_{tvl} represents the count of exact match TVLs, N represents the total number of TVLs in the test set.

- **F1 Score (TVL.F1)**: This metric addresses the challenge of evaluating multi-trajectory TVL queries, where changes in TVL order can cause misjudgments in traditional exact matching methods. We adopt F1-score as the primary evaluation metric to comprehensively reflect the model's performance in multi-trajectory semantic generation tasks. It is calculated as:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

  - True Positive (TP): The number of generated TVLs that correctly match the target TVLs.
  - False Positive (FP): The number of generated TVLs that do not match the target TVLs.
  - False Negative (FN): The number of target TVLs that are not matched by the generated TVLs.

B.3 Experimental Details

Under the baseline setting on TrajVL$_{\text{normal}}$, GPT-4o-mini, Qwen2.5, and LLaMA3 achieved their best accuracies with 6-shot or 7-shot configuration, whereas Gemma peaked at 3-shot configuration. On the TrajVL$_{\text{complex}}$, all four LLMs achieved their best accuracies with 6-shot or 7-shot configuration. Under the TRCAT setting on TrajVL$_{\text{normal}}$, GPT-4o-mini and LLaMA3 performed best with 5-shot or 6-shot configuration, Qwen2.5 and Gemma again with 2-shot configuration. On the TrajVL$_{\text{complex}}$, GPT-4o-mini and LLaMA3 performed best with 5-shot or 6-shot configuration, Qwen2.5 and Gemma again with 3-shot configuration. When the number of retrieval examples exceeds the optimal value for each LLM, excessively long context lengths may have a negative impact on the model's performance.

B.4 Experimental Prompt for Few-shot Configuration

```
Given a [Database schema] and a [Question],
 generate valid TVL sentences.

## Background
### TVL Structure:
Visualize [visualize] Area [area] ST [start
 time] ET [end time] SELECT [COLUMNS] FROM
[TABLES] [JOIN] [WHERE] [GROUP BY] [ORDER
BY]

### Key Components:
1. Visualization: The type of visual chart
that users expect.
2. Area: Extracting geographic area
information.
3. Time: Extracting time information. The
time format is: XXXX-XX-XX XX:XX:XX. (ST:
Start Time, ET: End Time. If no time is
specified, set to "null")
4. SQL Components: SELECT, FROM, JOIN,
WHERE, GROUP BY, ORDER BY.
You must consider which part in the SQL
components is necessary, which is
unnecessary.

When generating TVL, we should always
consider special rules and constraints:
```

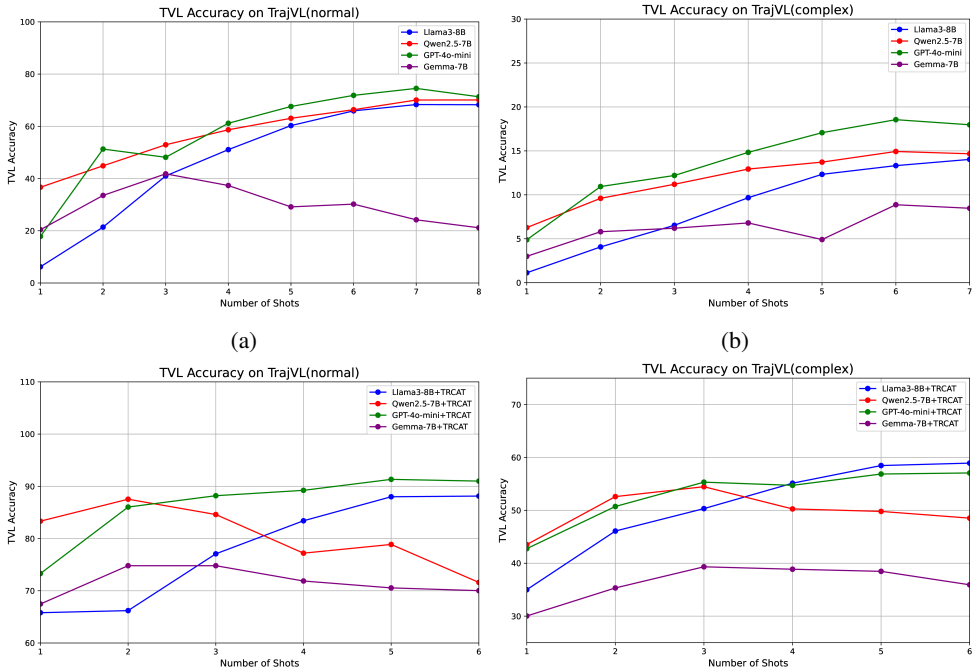


Figure 7: Parameter study of Baseline on TrajVL. The vertical axis denotes TVL.Acc, and the horizontal axis denotes the number of retrieved exemplars.

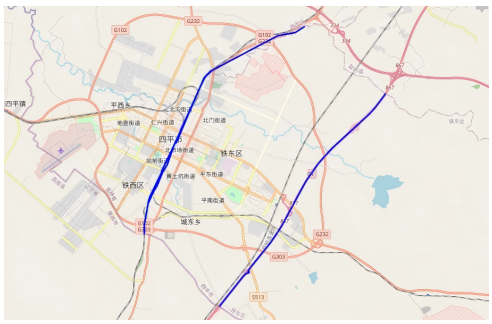
```
## Special Rules
a. For simple visualizations (bar, line,
pie):
    - SELECT exactly TWO columns, X-axis and
     Y-axis.
b. Aggregate Functions:
    - Use COUNT for counting occurrences
    - Use SUM only for numeric columns
    - When in doubt, prefer COUNT over SUM
    - When extract the month from datetime,
    use EXTRACT

## Constraints
- In SELECT <column>, make sure there are
at least two selected.
- Use only table names and column names
from the given database schema.
- Ensure GROUP BY precedes ORDER BY for
distinct values.

Here are some examples:
## Database schema
### Table: traj_data
[
  (id, Value examples: [20038172, 1, 2, 3].
  And this is an id type column),
  (traj_id, Value examples: [1522, 12917,
  14059, 8149]. And this is an id type
  column),
  (latitude, Value examples: [39.97539,
  39.97535, 39.97525, 39.97515].),
  (longitude, Value examples: [116.32769,
  116.327375, 116.327325, 116.45166].),
  (datetime, Value examples: ['2011-08-30
  00:51:53'].)
]
### Table: add_traj_data
[
  (id, Value examples: [20038172, 1, 2, 3].
  And this is an id type column),
  (traj_id, Value examples: [1522, 12917,
  14059, 8149]. And this is an id type
  column),
  (altitude, Value examples: [0.0, 164.0,
  128.0, 157.5].),
  (travel_mode, Value examples: ['walk', '
  bus', 'bike', 'car', 'train', 'subway'].),
  (user_id, Value examples: [128, 41, 17,
  68]. And this is an id type column)
]
## Question
{nl_query}
## Final TVL:
{TVL}

Now here is a new question:
## Database Schema
{database_schema}

## Question
{query}

Now, please generate the **Final TVL** for
the database schema and question.
(stop answering after you give the final
tvl)
```

B.5 Experimental Prompt for TRCAT Framework

```
Given a [Database schema] and a [Question],
 generate valid TVL sentences.
```

```
## Background
### TVL Structure:
Visualize [visualize] Area [area] ST [start
 time] ET [end time] SELECT [COLUMNS] FROM
[TABLES] [JOIN] [WHERE] [GROUP BY] [ORDER
BY]

### Key Components:
1. Visualization: The type of visual chart
that users expect.
2. Area: Extracting geographic area
information.
3. Time: Extracting time information. The
time format is: XXXX-XX-XX XX:XX:XX. (ST:
Start Time, ET: End Time. If no time is
specified, set to "null")
4. SQL Components: SELECT, FROM, JOIN,
WHERE, GROUP BY, ORDER BY.
You must consider which part in the SQL
components is necessary, which is
unnecessary.

When generating TVL, we should always
consider special rules and constraints:
## Special Rules
a. For simple visualizations (bar, line,
pie):
    - SELECT exactly TWO columns, X-axis and
     Y-axis.
b. Aggregate Functions:
    - Use COUNT for counting occurrences
    - Use SUM only for numeric columns
    - When in doubt, prefer COUNT over SUM
    - When extract the month from datetime,
    use EXTRACT

## Constraints
- In SELECT <column>, make sure there are
at least two selected.
- Use only table names and column names
from the given database schema.
- Ensure GROUP BY precedes ORDER BY for
distinct values.

Now we could think step by step:
1. First choose visualize type and area
name.
2. Second construct a sketch with temporal
structure for the natural language question.

3. Third extract the relevant information
and fill in the sketche to generate
complete SQL queries with temporal
information.
4. Fourth add visualize type and area name
to the SQL queries with time information to
 generate the final TVL.

Here are some examples:
## Database Schema
### Table: traj_data
[
  (id, Value examples: [20038172, 1, 2, 3].
  And this is an id type column),
  (traj_id, Value examples: [1522, 12917,
  14059, 8149]. And this is an id type
  column),
  (latitude, Value examples: [39.97539,
  39.97535, 39.97525, 39.97515].),
  (longitude, Value examples: [116.32769,
  116.327375, 116.327325, 116.45166].),
  (datetime, Value examples: ['2011-08-30
  00:51:53'].)
]
### Table: add_traj_data
[
  (id, Value examples: [20038172, 1, 2, 3].
  And this is an id type column),
  (traj_id, Value examples: [1522, 12917,
  14059, 8149]. And this is an id type
  column),
  (altitude, Value examples: [0.0, 164.0,
  128.0, 157.5].),
  (travel_mode, Value examples: ['walk', '
  bus', 'bike', 'car', 'train', 'subway'].),
  (user_id, Value examples: [128, 41, 17,
  68]. And this is an id type column)
]

## Question
{result['Matched NL Query'][i]}

Decompose the task into four sub-tasks,
considering [Background] [Special Rules] [
Constraints], and generate the TVL after
thinking step by step:

**Sub task 1:** Generate visualize type and
 area name
By parsing the Question, we generated the
visualize type and standardized area name.
{result['structure_fields'][i]}

**Sub task 2:** Construct sketches
By parsing the question, we found that {
result['tvl_num'][i]} specific sketches
with temporal structure need to be
constructed for the Question.
{result['TVL Sketch'][i]}

**Sub task 3:** Fill sketches
By parsing the question, we extracted
relevant information and filled them in the
 sketches to generate {result['tvl_num'][i]}
 complete SQL queries with temporal
information.
{result['SQL'][i]}

**Sub task 4:** Combine into Final TVL
Add visualize type and area name to the SQL
 queries with time information to generate
the final TVL.
Final TVL:
{result['TVL'][i]}

Now here is a new question:
## Database Schema
{database_schema}

## Question
{query}

Now, please generate the **Final TVL** for
the database schema and question after
thinking step by step.
(stop answering after you give the final
tvl)
```

NLQ	Show the trajectory data for the Tiedong District, Liaoning Province from 5:10:50 AM on February 28, 2009, to 10:37:04 AM on April 21, 2010, including all trajectories with an altitude between -7984.86 and 54056.95, and the trajectories of user 71 with an altitude between -16987.76 and 22297.35 from 4:42:47 AM on September 10, 2009, to 8:18:57 AM on January 2, 2012.
Target TVL	TVL1: Visualize map Area Tiedong District, Liaoning ST 2009-02-28 05:10:50 ET 2010-04-21 10:37:04 SELECT T1.traj_id, latitude, longitude, datetime FROM traj_data AS T1 JOIN add_traj_data AS T2 ON T1.traj_id = T2.traj_id WHERE altitude BETWEEN -7984.86 AND 54056.95 ORDER BY T1.traj_id, datetime TVL2: Visualize map Area Tiedong District, Liaoning ST 2009-09-10 04:42:47 ET 2012-01-02 08:18:57 SELECT T1.traj_id, latitude, longitude, datetime FROM traj_data AS T1 JOIN add_traj_data AS T2 ON T1.traj_id = T2.traj_id WHERE user_id = 71 AND altitude BETWEEN -16987.76 AND 22297.35 ORDER BY T1.traj_id, datetime
Few-shot	TVL1: Visualize map Area Tiedong District, Liaoning ST 2009-02-28 05:10:50 ET 2010-04-21 10:37:04 SELECT T1.traj_id, latitude, longitude, datetime FROM traj_data AS T1 JOIN add_traj_data AS T2 ON T1.traj_id = T2.traj_id WHERE (altitude BETWEEN -7984.86 AND 54056.95 OR (user_id = 71 AND altitude BETWEEN -16987.76 AND 22297.35 AND datetime BETWEEN '2009-09-10 04:42:47' AND '2012-01-02 08:18:57')) ORDER BY T1.traj_id, datetime →Figure 4a
TRCAT	TVL1: Visualize map Area Tiedong District, Liaoning ST 2009-02-28 05:10:50 ET 2010-04-21 10:37:04 SELECT T1.traj_id, latitude, longitude, datetime FROM traj_data AS T1 JOIN add_traj_data AS T2 ON T1.traj_id = T2.traj_id WHERE altitude BETWEEN -7984.86 AND 54056.95 ORDER BY T1.traj_id, datetime TVL2: Visualize map Area Tiedong District, Liaoning ST 2009-09-10 04:42:47 ET 2012-01-02 08:18:57 SELECT T1.traj_id, latitude, longitude, datetime FROM traj_data AS T1 JOIN add_traj_data AS T2 ON T1.traj_id = T2.traj_id WHERE user_id = 71 AND altitude BETWEEN -16987.76 AND 22297.35 ORDER BY T1.traj_id, datetime →Figure 4b



(a) ✗

(b) ✓

Table 4: Case study on multi-trajectory query. TVLs generated under few-shot and TRCAT configurations, along with their corresponding map visualizations (Errors are marked in red).

C Case Study

Table 4 presents a case study on multi-trajectory queries using GPT-4o-mini under different configurations. As shown, under the few-shot prompting configuration, the model fails to correctly identify the implicit multi-trajectory query structure within the NLQ, generating only a single TVL instance. Although this TVL instance includes all trajectory query conditions and visualizes two trajectories, it does not distinguish between them, making it difficult for users to distinguish individual trajectories (Figure 4a). This is because even if a user submits a single query, the corresponding results may contain multiple trajectory data points. For example,

NLQ	Provide a visualization of the trajectory taken on the thoroughfare known as Mingshui Avenue in the province of Shandong, filtering for those with an altitude between -7109.45 and 102496.41.
Target TVL	TVL1: Visualize map Area Mingshui Street, Shandong ST null ET null SELECT T1.traj_id, latitude, longitude, datetime FROM traj_data AS T1 JOIN add_traj_data AS T2 ON T1.traj_id = T2.traj_id WHERE altitude BETWEEN -7109.45 AND 102496.41 ORDER BY T1.traj_id, datetime
Few-shot	TVL1: Visualize map Area Mingshui Avenue, Shandong ST null ET null SELECT T1.traj_id, latitude, longitude, datetime FROM traj_data AS T1 JOIN add_traj_data AS T2 ON T1.traj_id = T2.traj_id WHERE altitude IS NOT NULL ORDER BY T1.traj_id, datetime → Figure 5a
TRCAT	TVL1: Visualize map Area Mingshui Street, Shandong ST null ET null SELECT T1.traj_id, latitude, longitude, datetime FROM traj_data AS T1 JOIN add_traj_data AS T2 ON T1.traj_id = T2.traj_id WHERE altitude BETWEEN -7109.45 AND 102496.41 ORDER BY T1.traj_id, datetime → 5b

(a) ✗ (b) ✓

Table 5: Case study. TVLs generated by few-shot and TRCAT configuration, together with their corresponding map visualizations (Errors are marked with red colors).

when a query covers a specific time period, the system may return multiple independent trajectories recorded for the user within that period. In contrast, TRCAT not only reproduces the target TVL structure but also accurately infers the multi-trajectory query structure from the NLQ, generating multiple TVL instances as needed. Benefiting from this structure, the map visualization can distinguish between trajectories using attributes such as color and query ID, as illustrated in Figure 4b.

Table 5 presents a case study of trajectory query using GPT-4o-mini under different configurations. As shown, the few-shot configuration produced incorrect Area keyword and erroneous data query constraints, resulting in the absence of the corresponding trajectory in Figure 5a. In contrast, GPT-4o-mini under TRCAT configuration not only generates TVL with the correct structure but also produces accurate area name and data constraints, leading to precise visualization as illustrated in Figure 5b.