TrajVisAgent: Automating Trajectory Visualization from Natural Language Queries via Collaborative Agent Workflow

Anonymous ACL submission

Abstract

The Natural Language to Trajectory Visualization (NL2TrajVis) task aims to automatically translate Natural Language Queries (NLQs) into trajectory data visualizations, thereby enabling natural language interaction within trajectory visualization systems. To overcome the limitations of existing benchmarks in cross-domain coverage and visual aesthetic modeling, we propose **TrajVL 2.0**, a novel benchmark dataset designed for NL2TrajVis. TrajVL 2.0 covers five representative application domains, comprises 4,552 high-quality samples and explicitly incorporates users' visual aesthetic preferences. Building on this dataset, we further propose **TrajVisAgent**, a multi-agent collaborative framework tailored for NL2TrajVis. TrajVisAgent comprises a **TVL Agent**, a **Code Agent**, and a **Visual Agent**, which collaboratively handle TVL generation and aesthetic attribute extraction, executable code synthesis with self-repair mechanisms, and iterative optimization guided by visual feedback. This framework enables end-to-end automation, spanning from natural language understanding to trajectory visualization generation and visualization quality refinement. We conduct a systematic evaluation against multiple existing methods on TrajVL 2.0. Experimental results demonstrate that TrajVisAgent consistently outperforms all baseline methods, achieving state-of-the-art performance. Ablation studies further validate the effectiveness of each individual agent as well as their collaborative design.

1 Introduction

With the rapid growth of large-scale trajectory data, numerous trajectory datasets have been released (Zhu et al., 2024; Amiri et al., 2024; Yuan et al., 2010). Trajectory visualization plays a critical role in uncovering spatiotemporal patterns and deriving actionable insights from trajectory data, thereby supporting analysis, decision-making, and planning. Although mature solutions based on programming frameworks (Zhu, 2013; Cheng et al., 2019; Wang, 2019) and Graphical User Interface (GUI)-based interactions (Scott and Janikas, 2009; Moyroud and Portet, 2018) are available, creating visualizations that are both accurate and aligned with users' aesthetic intentions still demands substantial programming expertise. This reliance creates a significant usability barrier for domain experts who lack a technical programming background.

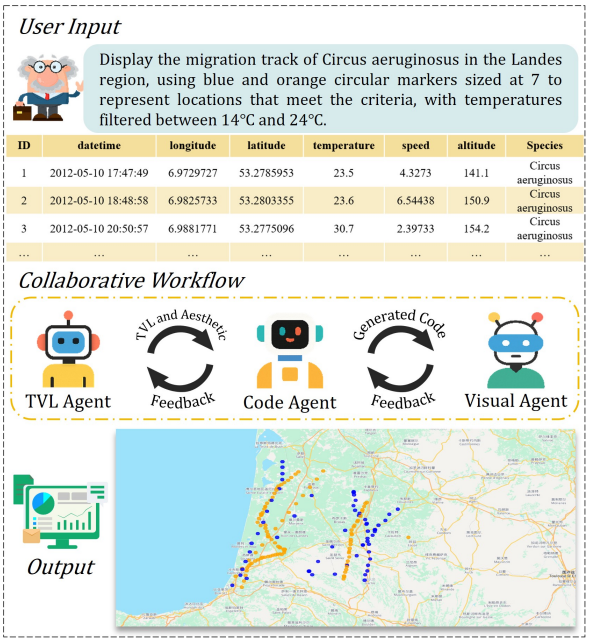


Figure 1: The pipeline of TrajVisAgent for generating trajectory visualizations from natural language queries.

To reduce the usability barrier of visualization systems, the Natural Language to Visualization (NL2VIS) task has attracted increasing research attention in recent years (Luo et al., 2021b,a; Song et al., 2023; Li et al., 2024; Zhang et al., 2024b; Song et al., 2024; Zhang et al., 2025; Luong and Nguyen, 2026). NL2VIS aims to automatically translate Natural Language Queries (NLQs) into structured visualization specifications or executable code. Owing to their strong semantic understand-

ing and code generation capabilities, Large Language Models (LLMs) have been widely adopted in NL2VIS, resulting in a series of LLM-based approaches (Maddigan and Susnjak, 2023; Chen et al., 2024; Dibia, 2023). However, most existing methods lack explicit task planning and visual feedback mechanisms, which limits their effectiveness when handling complex visualization requests. To address these limitations, recent studies have explored multi-agent frameworks (Ouyang et al., 2025; Yang et al., 2024; Goswami et al., 2025), in which specialized agents collaboratively support planning and visual feedback. Despite the progress achieved by multi-agent approaches, prior work has primarily focused on scientific or generic tabular data, leaving trajectory visualization underexplored. In particular, challenges arising from map-based rendering operations and spatiotemporal trajectory querying remain largely unaddressed.

Existing benchmark datasets (Luo et al., 2021a; Luo et al.; Lu et al., 2025; Rahman et al., 2025; Shuai et al., 2025) primarily focus on structured tabular data and conventional chart types and are not suitable for the Natural Language to Trajectory Visualization (NL2TrajVis) task. TrajVis (Bai et al., 2025) introduced the first benchmark dataset for natural language driven trajectory visualization by defining the Trajectory Visualization Language (TVL). However, this dataset relied on a single data source and did not fully account for users' aesthetic intentions during the visualization process. Moreover, for traditional chart types (e.g., bar and line charts), TrajVis primarily adopted template-based query generation, which limited the diversity of query logic and visual representations.

To overcome these limitations, we introduce **TrajVL 2.0**, a novel NL2TrajVis benchmark that spans multiple application domains and explicitly incorporates users' visual aesthetic preferences. TrajVL 2.0 covers five representative domains, including animal migration, disease transmission, logistics transportation, environmental monitoring, and human travel, and contains 4,552 carefully curated samples. To ensure data accuracy and expressive consistency, we adopt the TVL as an intermediate representation, which supports data querying, spatiotemporal constraint modeling, and visualization type specification. Rather than relying on fixed templates, we propose a constraint-aware dynamic SQL generation framework that substantially enhances the structural and semantic diversity of query logic. Furthermore, we design a scene-adaptive aesthetic attribute generator to systematically encode users' visual preferences during dataset construction process. As the first NL2TrajVis benchmark to jointly support multi-domain coverage and aesthetic-aware visualization, TrajVL 2.0 provides a solid foundation for evaluating and advancing future research in this area.

Building on this foundation, we propose **TrajVisAgent**, a multi-agent collaborative framework tailored for the NL2TrajVis task. TrajVisAgent comprises three specialized agents. The TVL Agent parses natural language queries into TVL and generates aesthetic attribute specifications in JSON format. The Code Agent translates the structured representations into executable visualization code and enhances code robustness through sandboxed execution and an error-attribution–based self-repair mechanism. The Visual Agent serves as the system's perception and feedback module, leveraging Multimodal Large Language Models (MLLM) to assess visualization quality and guide iterative refinement. The overall workflow of TrajVisAgent is illustrated in Figure 1.

We conducted a systematic evaluation of multiple methods on the TrajVL 2.0 dataset. The results demonstrate that TrajVisAgent significantly outperforms existing approaches, including LLM-based methods (e.g., Chat2VIS, CoML4Vis) and multi-agent frameworks (e.g., NVAgent). Specifically, under the GPT-5-mini configuration, the TrajVisAgent achieved a 21.78 improvement in visualization scores compared to Chat2VIS, the strongest baseline among existing methods. Ablation studies on individual agents further validate the effectiveness of the proposed multi-agent design. Collectively, these results demonstrate TrajVisAgent's ability to generate high-quality trajectory visualizations across multiple domains while effectively respecting user visualization preferences.

In summary, our contributions are as follows:

- We construct **TrajVL 2.0**, a multi-domain trajectory visualization benchmark that incorporates users' visual aesthetic preferences, providing a high-quality evaluation resource for the NL2TrajVis task.

- We propose **TrajVisAgent**, the first multi-agent collaborative framework designed for the NL2TrajVis tasks, achieving full-process automation from semantic parsing to visualization generation and optimization.

- We conduct experiments on the TrajVL 2.0 dataset, demonstrating that TrajVisAgent significantly outperforms existing methods on trajectory visualization tasks and achieves state-of-the-art performance.

2 Related Work

2.1 Text to Visualization Dataset

Existing NL2VIS benchmarks primarily focused on structured tabular data and conventional chart types. Notable examples include the nvBench series (Luo et al., 2021a; Luo et al.; Lu et al., 2025; Shuai et al., 2025; Song et al., 2026), which evaluated capabilities ranging from basic visual mapping to system robustness. Text2Vis (Rahman et al., 2025), DeepVIS (Shuai et al., 2025), and SIGN2Vis (Wan et al., 2025) further investigated reasoning, interaction, and multimodal understanding. Benchmarks such as NLV-Utterance (Maddigan and Susnjak, 2023), Chinese NL2VIS (Ge et al., 2024), RAViG Bench (Element et al.), and NALSpatial (Liu et al., 2023) examined linguistic diversity and spatial semantics understanding. However, these benchmarks are limited in supporting domain-specific visualization requirements for trajectory data. For trajectory scenarios, TrajVis introduced the first benchmark dataset in the NL2TrajVis domain. However, it relied on a single data source and did not model higher-level user intentions, such as visual aesthetic preferences. To address these gaps, we construct a large-scale NL2TrajVis benchmark spanning multiple application domains and explicitly incorporating visual aesthetic considerations.

2.2 Text to Visualization Method

Early NL2VIS studies generally treated the task as a sequence-to-sequence translation problem (Dibia and Demiralp, 2019; Luo et al., 2021a; Song et al., 2022; Wu et al., 2022; Song et al., 2024), relying heavily on large-scale annotated training datasets. With the emergence of LLMs, research attention has shifted toward LLM-based approaches (Xie et al., 2024; Wu et al., 2024; Ko et al., 2024; Shi et al., 2025). Chat2VIS (Maddigan and Susnjak, 2023), CoML4Vis (Chen et al., 2024), and LIDA (Dibia, 2023) leveraged LLMs' code generation capabilities to produce visualizations, while Prompt4Vis (Li et al., 2024) and ChartGPT (Tian et al., 2024) further improved semantic parsing via context optimization and stepwise reasoning. Despite these advances, LLM-based methods still lack explicit task planning and fail to effectively exploit visual feedback for iterative refinement.

To address these limitations, several recent approaches (Yang et al., 2024; Goswami et al., 2025; Chen et al., 2025; Zhang et al., 2024a) leveraged multi-agent systems with visual feedback or multi-path reasoning to enable iterative optimization during visualization generation. These methods mainly focused on scientific visualization scenarios that required fine-grained control and did not generalize to large-scale, cross-domain tasks. For the broader NL2VIS task, NVAgent (Ouyang et al., 2025) employed task planning and multi-agent collaboration to support complex multi-table queries and visualization generation. However, while effective for structured tabular data, NVAgent is less suited to the challenges posed by natural language–driven trajectory visualization. To bridge this gap, we propose TrajVisAgent, a multi-agent framework tailored for trajectory visualization, providing end-to-end automation from understanding user intentions to visualization generation.

3 TrajVL 2.0

3.1 Data Preparation

In constructing TrajVL 2.0, we retain the Trajectory Visualization Language (TVL) from TrajVL (Bai et al., 2025) as the intermediate representation. We preserve a subset of human travel data from TrajVL, including multi-trajectory queries, and extend the dataset with additional visualization types to increase diversity. To broaden domain coverage, we collect trajectory data from four new domains via Kaggle [1]: animal migration, logistics transportation, environmental monitoring, and disease transmission. For animal migration domain, trajectories for 27 species are manually curated from MoveBank (Kays et al., 2022). All data are stored in Postgres database (Douglas and Douglas, 2003), encompassing 69 tables and 2,420 geographic areas. Geographic boundaries are sourced from OpenStreetMap (Bennett, 2010), and base map rendering is provided by Baidu Maps [2].

3.2 TVL Generation

Given the structured design of TVL, we decompose the generation process into two stages: the data layer and the spatiotemporal constraint layer.

[1] https://www.kaggle.com/datasets
[2] https://lbsyun.baidu.com

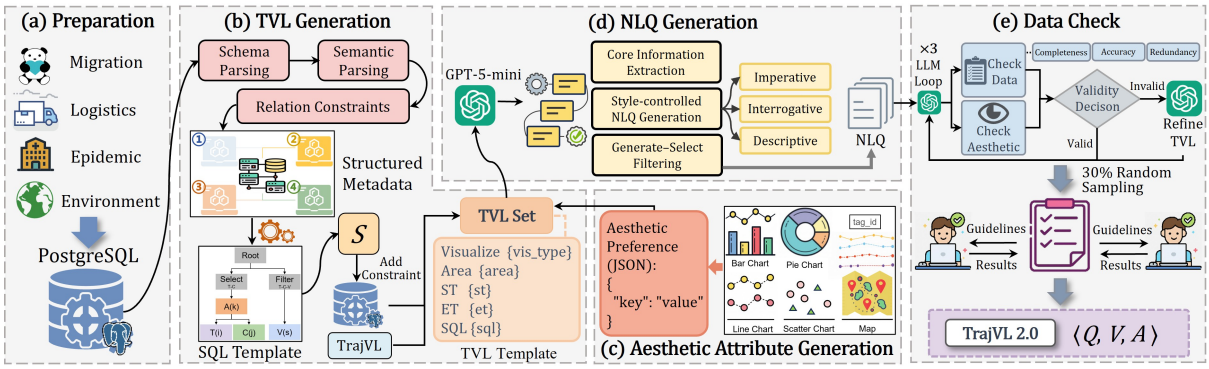


Figure 2: The construction process of the TrajVL 2.0 dataset. This process comprises five main steps: (a) Collecting and storing trajectory data from multiple domains into database; (b) Generating TVL instances using a metadata-driven, template-based SQL framework combined with spatiotemporal constraints; (c) Assigning adaptive aesthetic parameters based on visualization types; (d) Generating natural language queries based on TVLs and aesthetic parameters; (e) Validating generated queries through automated detection and manual review.

At the data layer, we propose a framework that integrates metadata modeling with template-driven SQL generation. We first construct a structured metadata schema to characterize table fields and inter-table relationships. Based on this schema, we design three categories of query templates, namely easy, medium, and hard, which correspond to single-table queries, conditional queries, and multi-table joins with complex aggregations, respectively. Templates are instantiated via metadata-aware random sampling and validated semantically. To ensure syntactic correctness, semantic validity, and sample diversity, we enforce data-type compatibility and relational constraints, and incorporate deduplication and failure-retry mechanisms.

Building on the generated queries, we further incorporate spatiotemporal constraints to construct complete TVLs. To ensure realistic visualization instructions, we adopt a data-driven sampling strategy. Specifically, we retrieve valid geographic areas (*Area*) and their corresponding temporal ranges (*ST, ET*) from the database, randomly sample matched area–time pairs, and integrate them with the SQL queries to produce the final TVLs.

3.3 Aesthetic Attribute Generation

To better reflect real-world visualization scenarios with user preferences, we design a scene-adaptive aesthetic attribute generator to enhance the expressiveness of each sample. In map-based visualizations, the generator first retrieves unique trajectory identifier from the PostgreSQL database and dynamically assigns color attributes to trajectory entities. It further adapts the base map style and rendering mode (point or line) according to data scale. For statistical chart visualizations, we construct chart-specific aesthetic attribute pools that comprising both shared and chart-dependent attributes. Guided by visual design best practices, the generator produces valid aesthetic configurations via conditional random sampling. These attributes are finally encapsulated with their corresponding TVL in structured JSON format, enriching the dataset with explicit aesthetic preference dimensions.

3.4 NLQ Generation

To construct high-quality samples, we synthesize NLQs from structured representations. Starting from TVL, we extract key semantic constraints, including temporal ranges, geographic entities, data filtering logic, and visualization aesthetic attributes. To promote linguistic diversity, we design three query styles: imperative, interrogative, and headline-style expressions. We adopt a generate–select strategy, producing multiple candidates per style and selecting the optimal query based on linguistic naturalness, syntactic fluency, and semantic completeness. Query generation is performed using GPT-5-mini with a temperature of 0.3 to balance reliability and diversity. The NLQ generation process is detailed in appendix A.2

3.5 Data Check

To ensure semantic completeness and accuracy of generated NLQs, we design a quality control pipeline combining automated verification with manual review. The automated validation using GPT-5-mini evaluates two aspects. First, data constraint validation checks whether essential information, such as visualization type, geographic area,

time range, and filtering conditions, is fully and correctly expressed. Second, aesthetic attribute verification assesses the completeness and correctness of aesthetic descriptions and the overall linguistic fluency. An NLQ is accepted only if it fully covers all non-empty core constraints without semantic errors or redundancy. Otherwise, a revised query is automatically generated. This automated verification is performed iteratively for three rounds.

To further assess reliability, we manually review a randomly sampled 30% of the data, independently evaluated by two graduate students. Manual inspection focuses on identifying missing critical information, semantic consistency with visualization metadata, and the presence of unnecessary technical details. See appendix A.3 for the detailed process of data verification. The overall dataset construction pipeline is illustrated in Figure 2.

3.6 TrajVL 2.0 Characteristics

The dataset contains 4,552 samples spanning 5 common visualization types. As summarized in Table 1, map-based visualizations constitute the largest portion, reflecting the characteristics of trajectory data, including 2,104 single-trajectory and 670 multi-trajectory samples. To support generalization to standard analytical tasks, the dataset also includes a balanced set of statistical charts: 826 line charts, 348 bar charts, 329 pie charts, and 275 scatter plots. To assess performance under varying query complexities, the dataset covers both single-table and multi-table scenarios, with 3,018 and 1,534 samples, respectively. Detailed statistics are reported in appendix B.1.

Vis	TVL	TVL+	Single	Multi	NLQ
Map	2,104	670	1,982	792	2,774
Bar	348	0	126	836	348
Line	826	0	395	431	826
Pie	329	0	228	101	329
Scatter	275	0	191	84	275
Total	3,882	670	3,018	1,534	4,552

Table 1: Statistics of the TrajVL 2.0 dataset. TVL+ denotes TVL supporting multi-trajectory queries, Single represents single-table queries, and Multi represents multi-table queries.

4 TrajVisAgent

To address the challenges in the NL2TrajVis task, we propose TrajVisAgent, as illustrated in Figure 3. TrajVisAgent is a multi-agent framework comprising three collaborative agents: the TVL Agent, the Code Agent, and the Visual Agent.

4.1 TVL Agent

The TVL Agent serves as the core component of TrajVisAgent, responsible for accurately translating NLQs into TVL while extracting visualization aesthetic attributes. To improve accuracy and robustness under complex queries, the agent integrates retrieval-augmented generation with hierarchical task decomposition. The TVL Agent first constructs a schema description containing field-level examples derived from the target database, providing well-defined data boundaries for subsequent model generation processes. This ensures the generated outputs are consistent with the target database. To prevent LLMs from generating structural errors when processing complex logic, the generation process is decoupled into two stages:

TVL Generation. The system employs a sentence-transformers-based retriever to retrieve examples from the knowledge base that are semantically most similar to the current natural language query, including the NLQ, corresponding TVL, and aesthetic attributes. Unlike approaches that directly concatenate retrieval results, we designed a heuristic prompt construction strategy in which retrieval examples are restructured according to a "task–subtask" logic. In this way, the prompt not only provides reference information but also guides the model to strictly adhere the specified generation path. The specific steps include: (a) extract visualization type and target area; (b) construct SQL templates with explicit start and end time constraints (*ST/ET*); (c) populate data and synthesize the complete TVL. This design ensures that the generation process conforms to with TVL's structural specifications, thereby preventing issues such as misalignment between temporal conditions and SQL filtering logic.

Aesthetic Attribute Generation. After the data logic is established and the TVL is generated, the system parses NLQs to extract visual presentation parameters, which are then encapsulated into independent JSON objects. The primary advantage of this layered task design is that users' visualization preferences remain decoupled from the underlying data logic, thereby enhancing the stability and reliability of the generation process.

4.2 Code Agent

The Code Agent serves as a bridge between structured data and visualizations. Its primary func-

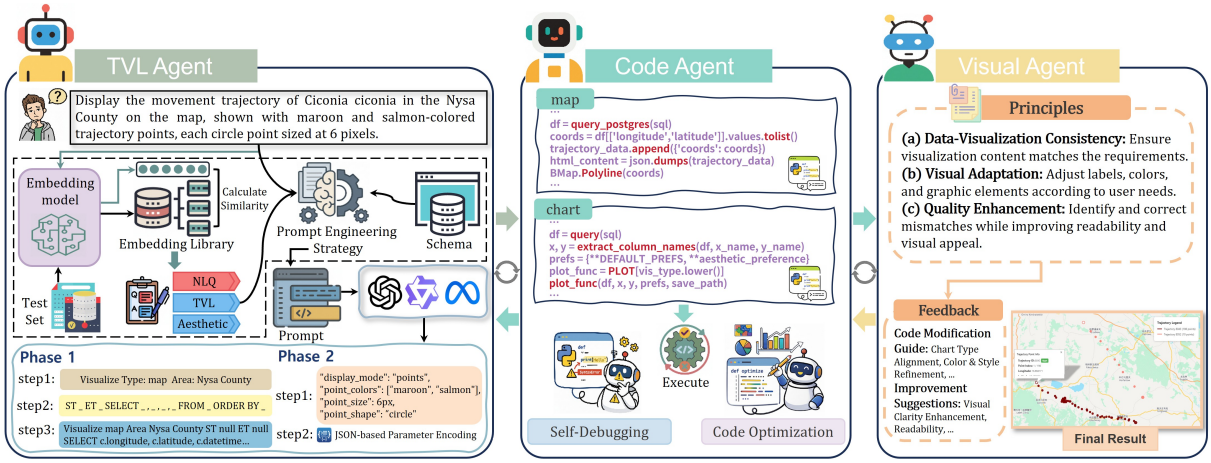


Figure 3: The workflow of the TrajVisAgent framework. The framework consists of three stages, each handled by a specialized agent: (a) TVL Agent, which maps natural language queries to TVL instances and generates aesthetic attributes via retrieval-augmented generation and task decomposition; (b) Code Agent, which converts TVL instances and aesthetic attributes into executable visualization code, performs script execution, and resolves errors through iterative feedback; (c) Visual Agent, which assesses visual quality and data–visual consistency of rendered outputs and provides structured feedback for refinement.

tion is to convert the TVL and JSON-formatted aesthetic attributes into executable Python scripts while ensuring stable and reliable execution. The Code Agent employs a rigorous error attribution and self-correction mechanism. Based on the visualization type specified in TVL, tasks are routed to different branches: (a) map branch, which generates HTML visualization pages based on Baidu Maps; (b) chart branch, which generates static charts using Matplotlib. Aesthetic attributes are mapped to the corresponding standard parameters in the selected visualization library. The generated code executes within a sandboxed environment to ensure security and controllability. If execution fails, for instance due to SQL syntax errors or empty data returns, the error messages are fed back to the TVL Agent for TVL correction. If the error stems from Python syntax or improper library usage, the Code Agent will automatically corrects the code internally. Upon receiving feedback from the Visual Agent, the Code Agent further refines the code to improve the quality of the final visualization. To prevent infinite loops, the number of debugging iterations is limited to a maximum of 3.

4.3 Visual Agent

While the Code Agent ensures syntactic correctness and stable execution, it cannot evaluate the visual quality or aesthetic effectiveness. To address this limitation, we introduce the Visual Agent, powered by MLLMs, serving as the perceptual feedback module of TrajVisAgent. After successful code execution and rendering, the resulting visualization is provided to the Visual Agent for qualitative assessment. A set of visual assessment criteria is defined to guide multidimensional evaluation, including data–visual consistency, label clarity, and overall aesthetic quality. Based on these criteria, the Visual Agent generates structured feedback, including modification guidelines that align adjustments with user intent, as well as fine-grained optimization suggestions for visual parameters. This feedback is subsequently returned to the Code Agent to initiate iterative refinement. By incorporating visual feedback into the multi-agent collaboration, TrajVisAgent enhances both the readability of visualizations and their alignment with user intent.

5 Experimental Setup

5.1 Dataset Splitting

To prevent data leakage, we adopt a signature-based hierarchical partitioning strategy for TrajVL 2.0. For each sample, a structured signature is derived from the TVL, capturing the core query logic, including table names, field attributes, aggregation operators, and spatial constraints. Samples sharing identical query logic are assigned to the same subset, effectively preventing set leakage. Partitioning is performed in a domain-aware manner: samples are first grouped by application domain to ensure balanced coverage, and unique signatures within each domain are randomly shuffled and split into

Model	Method	Vis.Acc	Data.Acc			TVL.Acc	Pass	Score	Aest.Acc
			Area	Time	SQL				
LLaMA3.1-8B	CoML4Vis	36.68%	-	-	-	-	61.86%	12.53	-
	Chat2VIS	20.41%	-	-	-	-	32.95%	17.98	-
	NVAent	84.20%	51.84%	72.32%	1.17%	1.00%	91.58%	9.42	-
	TrajVisAgent	98.10%	93.21%	**90.15%**	14.40%	13.99%	40.70%	22.86	23.15%
Qwen3-8B	CoML4Vis	35.63%	-	-	-	-	63.54%	22.85	-
	Chat2VIS	31.66%	-	-	-	-	56.56%	22.64	-
	NVAent	80.71%	28.04%	64.83%	1.92%	1.79%	87.87%	14.72	-
	TrajVisAgent	97.27%	96.14%	89.39%	30.52%	28.47%	79.85%	43.81	49.36%
GPT-5-mini	CoML4Vis	37.92%	-	-	-	-	70.02%	30.66	-
	Chat2VIS	44.05%	-	-	-	-	84.51%	37.58	-
	NVAent	71.65%	27.38%	65.09%	1.49%	1.27%	69.05%	10.21	-
	TrajVisAgent	**99.40%**	**98.13%**	88.64%	**32.56%**	**32.16%**	**93.03%**	**59.36**	**54.96%**

Table 2: Performance of various frameworks on the TrajVL 2.0 dataset when using different backbone models.

three subsets: **Training Set** (1,155 samples, used for constructing the knowledge base), **Validation Set** (1,129 samples), and **Test Set** (2,268 samples).

5.2 Baselines

As NL2TrajVis is a newly introduced task lacking dedicated benchmark frameworks, we select representative baselines from prior NL2VIS research that most closely align with the TrajVL 2.0 dataset and task configuration. To enable comprehensive comparison, we consider two categories of state-of-the-art methods: LLM-based frameworks and multi-agent collaboration frameworks. Specifically, we compare TrajVisAgent against: (a) **Chat2VIS** (Maddigan and Susnjak, 2023), which generates visualizations from natural language via prompt engineering; (b) **CoML4Vis** (Chen et al., 2024), a few-shot prompting approach that integrates multi-table information; and (c) **NVAgent** (Ouyang et al., 2025), a multi-agent framework supporting complex single- and multi-table queries through cooperative reasoning. To ensure fair comparison, we employ three identical large language model implementations across TrajVisAgent and all baselines: GPT-5-mini [3], Qwen3-8B (Yang et al., 2025), and LLaMA3.1-8B [4].

5.3 Evaluation Metrics

To comprehensively assess performance from data retrieval to visualization generation, we adopt a multidimensional evaluation framework encompassing intermediate representation accuracy, aesthetic alignment, code executability, and visual quality. For evaluating TVL quality, we follow the protocol used in TrajVL (Bai et al., 2025). Code executability is measured using Pass Rate (*Pass*), as introduced by NVAgent, which quantifies the proportion of syntactically valid and successfully executable code. Prior studies (e.g., MatplotAgent (Yang et al., 2024)) have shown that MLLM–based evaluations strongly correlate with human judgments. Accordingly, we employ GLM-4.1V-9B-Thinking (Hong et al., 2025) to assess visualization quality, reporting a Visualization Score (*Score*) ranging from 0 to 100. To evaluate alignment with users' aesthetic intentions, we introduce Aesthetic Accuracy (*Aest.Acc*), defined as the average proportion of predicted aesthetic attributes that exactly match the ground truth. Detailed definitions of all metrics are provided in the appendix B.3.

6 Results and Discussion

6.1 Performance Comparison

Table 2 presents a comprehensive comparison of different methods on the NL2TrajVis task. Appendix B.4 provides detailed information on the experimental parameter settings. Experimental results indicate that, across all selected large language model configurations (LLaMA3.1-8B, Qwen3-8B, and GPT-5-mini), our proposed multi-agent framework outperforms the baselines on all core evaluation metrics.

[3] https://platform.openai.com/docs/models/gpt-5-mini
[4] https://huggingface.co/meta-llama/Llama-3.1-8B

For Chat2VIS and CoML4Vis, which directly convert NLQ into visualization code and bypass the construction of the visualization intermediate representation, we evaluate only visualization type recognition accuracy (*Vis.Acc*), code execution pass rate (*Pass*), and visualization score (*Score*). Experimental results indicate that our proposed multi-agent framework consistently outperforms both methods across all evaluated metrics and model configurations. Similarly, NVAgent, which generates visual intermediate representations, exhibits significant maladaptation in trajectory visualization tasks. Its generated TVL accuracy peaked at only 1.79%. Notably, although NVAgent achieves a high code execution pass rate under the LLaMA3.1-8B configuration, this largely reflects its tendency to produce images containing incorrect information, rather than generating valid visualizations in the presence of logical errors or failed data queries. Consequently, NVAgent attains a visualization score of only 14.72, substantially lower than that of our proposed framework. In appendix B.5, we present case studies of the four frameworks.

In a comparative analysis across different LLM configurations, our proposed framework achieved the best performance when using GPT-5-mini as the backbone model. Specifically, the TVL accuracy reaches 32.16%, the code execution pass rate reaches 93.03%, and the visualization quality score is 59.36. The configuration using Qwen3-8B achieves the second-best performance, whereas LLaMA3.1-8B exhibits comparatively weaker overall results. Although the proposed multi-agent framework demonstrates substantial improvements over existing methods, the metrics *TVL.Acc*, *Aest.Acc*, and *Score* still reflect the intrinsic difficulty of the NL2TrajVis task. Existing models retain considerable room for improvement in accurately capturing user intent, as the task involves complex trajectory data queries with both temporal and spatial constraints, in addition to modeling users' visual aesthetic preferences.

6.2 Ablation Study

In this section, we perform ablation experiments to assess the effectiveness of individual agents within the TrajVisAgent framework and their respective contributions. Specifically, we first evaluated the full framework with all agents enabled. Subsequently, we conducted comparative analyses of the following configurations by systematically removing individual agents: (a) Without the TVL Agent (w/o TVL Agent); (b) Without the Code Agent (w/o Code Agent); (c) Without the Visual Agent (w/o Visual Agent). Since TrajVisAgent achieves optimal performance under the GPT-5-mini configuration, all ablation experiments are conducted using GPT-5-mini to ensure fair and consistent comparisons.

Setting	TVL.Acc	Aest.Acc	Pass	Score
TrajVLAgent	**32.16%**	**54.96%**	93.03%	**59.36**
- w/o TVL Agent	10.87%	27.28%	79.63%	43.99
- w/o Code Agent	31.48%	54.93%	89.42%	54.80
- w/o Visual Agent	30.84%	54.11%	**93.96%**	58.41

Table 3: Ablation results for each agent within TrajVisAgent based on GPT-5-mini.

The experimental results presented in Table 3 highlight the critical contributions of the three agents within the proposed framework. In particular, the TVL Agent, responsible for determining the data logic required for visualization, played a pivotal role. Removal of the TVL Agent leads to a substantial drop in performance across *TVL.Acc*, *Aest.Acc*, and *Score*, underscoring its central role in accurately parsing user query intent. Moreover, experimental results indicate that Code Agent and Visual Agent play essential roles in generating executable code and enhancing visualization quality, respectively. Ablation of either agent individually results in corresponding declines in *Pass* and *Score*, confirming their respective contributions to code reliability and visual effectiveness.

7 Conclusion

We introduce TrajVL 2.0, the most comprehensive benchmark to date for NL2TrajVis, and the first to explicitly incorporate user visualization preferences. The dataset covers multiple application domains and integrates a dynamic SQL generation framework with a scene-adaptive aesthetic attribute generator, jointly addressing complex query logic and users' aesthetic intent. Building on this benchmark, we propose TrajVisAgent, a multi-agent collaborative framework that forms a closed-loop pipeline from TVL generation and aesthetic extraction to visualization code synthesis, error correction, and visual feedback–driven refinement. Extensive experiments demonstrate that TrajVisAgent consistently outperforms existing methods in *TVL.Acc*, *Aest.Acc*, and *Score*. Future work will focus on improving trajectory query efficiency and further strengthening perceptual understanding and feedback in intelligent visualization agents.

Limitations

Although TrajVL 2.0 and the TrajVisAgent framework provide comprehensive benchmarks and methodological support for the NL2TrajVis task, certain limitations remain. In scenarios with large-scale trajectory data, multi-table join queries impose substantial data retrieval overhead. The current framework has not been systematically optimized for query efficiency on large-scale trajectory data, which partially constrains the overall speed of visualization generation. In future work, it will be essential to enhance query and retrieval algorithms to ensure efficient data access and visualization generation when handling large-scale trajectory datasets.

References

Hossein Amiri, Richard Yang, and Andreas Züfle. 2024. 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.

Tian Bai, Huiyan Ying, Kailong Suo, Junqiu Wei, Tao Fan, and Yuanfeng Song. 2025. Text-to-trajvis: Enabling trajectory data visualizations from natural language questions. *arXiv preprint arXiv:2504.16358*.

Jonathan Bennett. 2010. *OpenStreetMap*. Packt Publishing Ltd.

Nan Chen, Yuge Zhang, Jiahang Xu, Kan Ren, and Yuqing Yang. 2024. Viseval: A benchmark for data visualization in the era of large language models. *IEEE Transactions on Visualization and Computer Graphics*.

Zichen Chen, Jiefeng Chen, Sercan Ö Arik, Misha Sra, Tomas Pfister, and Jinsung Yoon. 2025. Coda: Agentic systems for collaborative data visualization. *arXiv preprint arXiv:2510.03194*.

Joe Cheng, Bhaskar Karambelkar, Yihui Xie, Hadley Wickham, Kenton Russell, Kent Johnson, Barret Schloerke, and Vladimir Agafonkin. 2019. Package 'leaflet'.

Victor Dibia. 2023. Lida: A tool for automatic generation of grammar-agnostic visualizations and infographics using large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 113–126.

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.

Korry Douglas and Susan Douglas. 2003. *PostgreSQL: a comprehensive guide to building, programming, and administering PostgresSQL databases*. SAMS publishing.

Missing Element, Low Color Contrast, and Hallucinated Data. Ravig-bench: Abenchmark for retrieval-augmented visually-rich generation with multi-modal automated evaluation.

Yan Ge, Victor Junqiu Wei, Yuanfeng Song, Jason Chen Zhang, and Raymond Chi-Wing Wong. 2024. Automatic data visualization generation from chinese natural language questions. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 1889–1898.

Kanika Goswami, Puneet Mathur, Ryan Rossi, and Franck Dernoncourt. 2025. Plotgen: Multi-agent llm-based scientific data visualization via multimodal feedback. *arXiv preprint arXiv:2502.00988*.

Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, and 1 others. 2025. Glm-4.1 v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning. *arXiv preprint arXiv:2507.01006*.

Roland Kays, Sarah C Davidson, Matthias Berger, Gil Bohrer, Wolfgang Fiedler, Andrea Flack, Julian Hirt, Clemens Hahn, Dominik Gauggel, Benedict Russell, and 1 others. 2022. The movebank system for studying global animal movement and demography. *Methods in Ecology and Evolution*, 13(2):419–431.

Hyung-Kwon Ko, Hyeon Jeon, Gwanmo Park, Dae Hyun Kim, Nam Wook Kim, Juho Kim, and Jinwook Seo. 2024. Natural language dataset generation framework for visualizations powered by large language models. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pages 1–22.

Shuaimin Li, Xuanang Chen, Yuanfeng Song, Yunze Song, and Chen Zhang. 2024. Prompt4vis: Prompting large language models with example mining and schema filtering for tabular data visualization. *arXiv preprint arXiv:2402.07909*.

Mengyi Liu, Xieyang Wang, Jianqiu Xu, and Hua Lu. 2023. Nalspatial: An effective natural language transformation framework for queries over spatial data. In *Proceedings of the 31st ACM International Conference on Advances in Geographic Information Systems*, pages 1–4.

Jinwei Lu, Yuanfeng Song, Haodi Zhang, Chen Jason Zhang, Kaishun Wu, and Raymond Chi-Wing Wong. 2025. Towards robustness of text-to-visualization translation against lexical and phrasal variability. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, pages 793–806. IEEE.

Tianqi Luo, Chuhan Huang, Leixian Shen, Boyan Li, Shuyu Shen, Wei Zeng, Nan Tang, and Yuyu Luo. nvbench 2.0: A benchmark for natural language to visualization under ambiguity.

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.

Hue Luong and Vinh T Nguyen. 2026. Nl2vis transformed: From linguistic abstraction to visual specification in the generative ai era. *SN Computer Science*, 7(1):19.

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.

Nicolas Moyroud and Frédéric Portet. 2018. Introduction to qgis. *QGIS and generic tools*, 1:1–17.

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*.

Mizanur Rahman, Md Tahmid Rahman Laskar, Shafiq Joty, and Enamul Hoque. 2025. Text2vis: A challenging and diverse benchmark for generating multimodal visualizations from text. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 31837–31862.

Lauren M Scott and Mark V Janikas. 2009. Spatial statistics in arcgis. In *Handbook of applied spatial analysis: Software tools, methods and applications*, pages 27–41. Springer.

Shengze Shi, Tao Ren, Guoliang Zhu, Guandong Feng, and Jun Hu. 2025. Closing the feedback loop in text2vis: Refining visualization with vision-language models. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 9053–9061.

Zhihao Shuai, Boyan Li, Siyu Yan, Yuyu Luo, and Weikai Yang. 2025. Deepvis: Bridging natural language and data visualization through step-wise reasoning. *arXiv preprint arXiv:2508.01700*.

Yuanfeng Song, Xiaoling Huang, Xuefang Zhao, and Raymond Chi-Wing Wong. 2023. Natural language generation meets data visualization: vis-to-text and its duality with text-to-vis. In *2023 IEEE International Conference on Data Mining (ICDM)*, pages 1337–1342. IEEE.

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(1):3.

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.

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*, 31(3):1731–1745.

Yao Wan, Yang Wu, Zhen Li, Guobiao Zhang, Hongyu Zhang, Zhou Zhao, Hai Jin, and April Wang. 2025. Sign2vis: Automated data visualization from sign language. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 17839–17857.

Yang Wang. 2019. Deck. gl: Large-scale web-based visual analytics made easy. *arXiv preprint arXiv:1910.08865*.

Yang Wu, Yao Wan, Hongyu Zhang, Yulei Sui, Wucai Wei, Wei Zhao, Guandong Xu, and Hai Jin. 2024. Automated data visualization from natural language via large language models: An exploratory study. *Proceedings of the ACM on Management of Data*, 2(3):1–28.

Zhengkai Wu, Vu Le, Ashish Tiwari, Sumit Gulwani, Arjun Radhakrishna, Ivan Radiček, Gustavo Soares, Xinyu Wang, Zhenwen Li, and Tao Xie. 2022. Nl2viz: natural language to visualization via constrained syntax-guided synthesis. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 972–983.

Liwenhan Xie, Chengbo Zheng, Haijun Xia, Huamin Qu, and Chen Zhu-Tian. 2024. Waitgpt: Monitoring and steering conversational llm agent in data analysis with on-the-fly code visualization. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, pages 1–14.

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*.

Zhiyu Yang, Zihan Zhou, Shuo Wang, Xin Cong, Xu Han, Yukun Yan, Zhenghao Liu, Zhixing Tan, Pengyuan Liu, Dong Yu, and 1 others. 2024. 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*.

Haodi Zhang, Xinhe Zhang, Jihua Zhou, Kaishun Wu, Yuanfeng Song, and Raymond Chi-Wing Wong. 2025. Speech-to-visualization: Toward end-to-end speech-driven data visualization generation from natural language questions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 437–453. Springer.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, and 1 others. 2024a. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*.

Weixu Zhang, Yifei Wang, Yuanfeng Song, Victor Junqiu Wei, Yuxing Tian, Yiyan Qi, Jonathan H Chan, Raymond Chi-Wing Wong, and Haiqin Yang. 2024b. Natural language interfaces for tabular data querying and visualization: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 36(11):6699–6718.

Nick Qi Zhu. 2013. *Data visualization with D3. js cookbook*. Packt Publishing Ltd.

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

A More Implementation Details

A.1 Generating SQL from TVL

The system performs SQL conversion by parsing the underlying logical structure encoded in TVL, thereby enabling trajectory data queries. During this process, spatiotemporal constraints specified in TVL are dynamically mapped to the *WHERE* clause of the generated SQL query. For example, when TVL specifies the *Area* as "Mamers", the Start Time (*ST*) as "2015-04-14 09:35:36", and the End Time (*ET*) as "2017-03-27 06:20:53", the resulting SQL follows the basic structure shown below: "*FROM animal.circus_aeruginosus c JOIN boundaries b ON b.name = 'Mamers' WHERE ST_Within(ST_SetSRID(ST_MakePoint(c.longitude, c.latitude), 4326), b.geom) AND datetime BETWEEN 2015-04-14 09:35:36 AND 2017-03-27 06:20:53*". Based on these parameters, the system dynamically generates the corresponding SQL query to retrieve trajectory data, as illustrated in Figure 4.

TVL

Visualize map
Area Mamers
ST 2015-04-14 09:35:36
ET 2017-03-27 06:20:53
SELECT c.datetime, c.longitude, c.latitude, c.tag_id
FROM animal.circus_aeruginosus c
WHERE c.speed > 0.01 AND c.speed < 42.7
ORDER BY c.traj_id, c.datetime

Synthesized SQL

SELECT c.datetime, c.longitude, c.latitude, c.tag_id
FROM animal.circus_aeruginosus c *JOIN* boundaries b ON b.name = ' Mamers '
WHERE ST_Within(ST_SetSRID(ST_MakePoint(c.longitude, c.latitude), 4326), b.geom)
AND datetime BETWEEN 2015-04-14 09:35:36' AND 2017-03-27 06:20:53 ' AND c.speed > 0.01 AND c.speed < 42.7
ORDER BY c.traj_id, c.datetime

Figure 4: The mechanism for generating a data query language from TVL.

A.2 Prompt for NLQ Generation

```
### Task Description:
You are an intelligent assistant. You will
create three natural language queries (command,
 question, and caption) based on a given TVL
and aesthetic preferences. Each query must
incorporate all information from the TVL and
aesthetic preferences without introducing extra
 elements.

### TVL:
{tvl}

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

### Aesthetic Preferences:
{aesthetic_str}

### Rules (Apply to ALL)
1. Content Rules:
- Must ensure that the three generated natural
language queries naturally include **the place
name: {area}**.
- The time range (ST to ET) must be accurate to
 the **second**.
- **Attention**: Must ensure that all WHERE
constraints in TVL must be included in the
three generated natural language queries
naturally ({where_clause}).
- If time range (ST to ET) in the TVL is "null",
 DO NOT describe time at all.
- DO NOT include database names (e.g., "animal.
bassaricyon_alleni" , just mention the species).


2. Aesthetic Preferences Rules:
- MUST incorporate **all the map visualization
aesthetic preferences** naturally into each
query.
- If there are multiple tag colors, ALL colors
must be listed in sequence.
- Make it sound like a real user request, not a
 technical specification.

3. Natural Language Rules:
- Sound fluent and natural, like real user
queries.
- Use natural, conversational language.
- Avoid overly technical or formal language.

### Step 1: Style-Specific Generation
For each of the three styles, generate **three
distinct variations** with clearly different
sentence openings and tone.

**1. Command-style (3 candidates):**
- Use strong, directive verbs (e.g., "Plot", "
Display", "Generate", "Create", "Render", "Show
", ...).
- Each variation must have a different phrasing
 and sentence flow.

**2. Question-style (3 candidates):**
- Use natural question forms (e.g., "How can we
...", "What's the best way to...", "Can you
show...", ...).
- Ensure each question has a unique tone and
phrasing.

**3. Caption-style (3 candidates):**
- Use declarative/descriptive tone (e.g., "
Visualization of...", "Map view of...", "A time-
based plot of...", ...).
- Each variation must read like a caption or
figure title.

### Step 2: Internal Selection
After generating the 3 variations for each
style, **select the single most natural, fluent,
```

```
 and complete one** from each group as the
final output.

### Step 3: Output Format
Return **exactly 3 final queries** (one per
style)  each already selected as the best from
its internal group, one per line, in this
format:
1. [Command-style]
2. [Question-style]
3. [Caption-style]

### Natural Language Queries (before generating,
 ensure that WHERE constraints are already
included in each natural language query):
```

A.3 Prompt for Data Check

```
### Task Description:
You are a validation assistant. Your task is to
 check whether this natural language query
accurately contains the core information
provided in a given TVL and its aesthetic
preferences.

### Input Information
**TVL:**
{tvl}

**Aesthetic Preferences:**
{aesthetic_str}

**Core Information:**
1. Visualize: map
2. Area: {area}
3. Time Range: [{time_range['ST']}, {time_range
['ET']}]
4. {where_clause}
5. Aesthetic preferences: {aesthetic_str}

**Natural Language Query:**
query_{query_index}: {nl_query}

### Evaluation Rules
#### A query is **Valid** if:
1. It contains all **Core Information**:
- **map** (always required)
- **Area** (always required)
- **Time Range** (if present and not null)
- **WHERE constraints** (if present and not
null)
- **Aesthetic preferences** (always required)
2. It does **not** contain any **redundant
database or field names**.
3. All **Core Information** is accurate and
consistent.
4. Minor stylistic or wording differences are
acceptable if the meaning remains correct.

#### A query is **Invalid** if:
- Any required element [map, Area, Time range (
ST to ET), WHERE constraints and Aesthetic
preferences (if applicable)] is missing.
- Any value among the required elements [map,
Area, Time range (ST to ET), WHERE constraints
and Aesthetic preferences (if applicable)] is
incorrect.
- It includes redundant or irrelevant database
information.
> Note: If a field (such as WHERE clause or
Time Range) is **absent or null** in the TVL, **
do not penalize** the query for not mentioning
it.

### Validation Task

For the query **query_{query_index}**:

1. Strictly follow the **Evaluation Rules** to
determine its **Validity**: "Valid" or "Invalid
".
2. List any issues found:
- **Missing Data:** Core Information not
included.
- **Incorrect Data:** Contains mismatched
values.
- **Redundant Data:** Contains prohibited
database or field names.
3. If the query is **Invalid**, provide a **
corrected version** that fixes only those
issues.
4. If the query is **Valid**, the `
corrected_query` field may be omitted.

### Output Format
Return your validation results **only** in the
following strict JSON structure:
"`json
{{
  "validity": "Valid" or "Invalid",
  "issues": {{
    "missing_data": ["list of missing items"],
    "redundant_data": ["list of redundant items
    "],
    "incorrect_data": ["list of incorrect items
    "]
  }},
  "corrected_query": "If invalid, provide the
  corrected version of the query here. If valid,
   this field can be null or omitted."
}}
"`

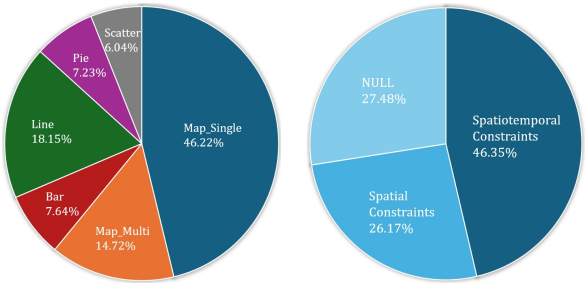
### Provide your validation:
(Please ensure that you have carefully reviewed
 and thoughtfully provided your verification
results to ensure no omissions.)
```

B More Experimental Details

B.1 Dataset Statistics

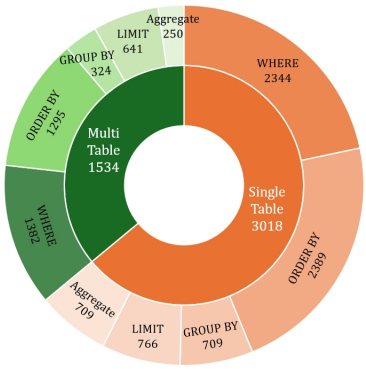
This section details the key characteristics of the constructed dataset, including visualization types, spatio-temporal constraint complexity, and underlying SQL structure.

Distribution of Visualization Types. The dataset comprises 4,552 samples, covering six mainstream visualization formats. As shown in Figure 5a, given the domain characteristics of trajectory data, map-based visualizations account for the largest proportion, comprising 2,104 single-



(a) Visualization Chart Type Distribution

(b) Granularity of Spatiotemporal Constraints



(c) SQL Structures

Figure 5: Key Statistics of TrajVL 2.0.

trajectory samples and 670 multi-trajectory samples. Additionally, to ensure the system's generalization capabilities for common analytical tasks, the dataset incorporates a balanced mix of line charts (826 samples) and bar charts (348 samples), pie charts (329 samples), and scatter plots (275 samples).

Spatio-temporal constraint granularity. A core contribution of this dataset is the systematic introduction of spatio-temporal constraints. Figure 5b illustrates the distribution of constraint dimensions: 2,110 samples feature complete spatio-temporal constraints (covering Area, ST, and ET), requiring models to accurately extract spatio-temporal information; 1,191 samples contain only spatial constraints (Area); and 1,251 samples lack spatio-temporal constraints, serving as supplementary data for basic semantic query scenarios.

Data Query Structure. To investigate the model's performance across varying levels of logical complexity, Figure 5c illustrates the distribution of SQL structures in single-table queries (3,018 instances) and multi-table queries (1,534 instances). The data encompasses diverse combinations of *WHERE* filtering, *GROUP BY* aggregation, and *ORDER BY* sorting.

B.2 Baselines

- **LLaMA3.1-8B:** A autoregressive large language model released by Meta with approximately 8 billion parameters. It employs an optimized Transformer architecture and is pre-trained on over 15 trillion tokens from publicly available sources, supporting multilingual text generation and reasoning. The model extends the context window to 128K tokens, enabling it to handle tasks such as long-form text comprehension, multi-turn conversations, and complex logical reasoning.

- **Qwen3-8B:** An open-source variant of the latest generation large language model in the Tongyi Qwen series, featuring approximately 8.2 billion parameters and pre-trained using an autoregressive Transformer architecture. The model was trained on large-scale, multilingual corpora and supports seamless switching between "thinking" and "non-thinking" modes, achieving a balance between complex logical reasoning, code generation, and conversational tasks.

- **GPT-5-mini:** A proprietary large language model provided by OpenAI, designed to strike a good balance between reasoning capabilities, generation quality, and computational efficiency. Compared to larger models, GPT-5-mini significantly reduces inference overhead while maintaining high generation quality, making it suitable as an experimental baseline model requiring efficient reasoning and consistent output.

- **CoML4Vis:** A prompt-optimized NL2V framework proposed in the VisEval benchmark, designed to evaluate and enhance large language models' ability to generate visualization code. This framework takes natural language queries and tabular data as input to directly generate executable Python visualization code, serving as one of the core implementation solutions in VisEval for systematically evaluating LLMs' visualization capabilities.

- **Chat2VIS:** An end-to-end NL2VIS system based on pre-trained large language models, designed to directly convert free-form natural language queries into executable Python visualization code. Through carefully engineered

prompts that combine user queries with structured metadata from datasets, this method guides the language model to autonomously complete visualization type selection, data field mapping, and chart generation.

- **NVAgent:** A multi-agent collaborative NL2VIS framework designed to address complex visualization demands in both single-table and multi-table scenarios. By introducing three distinct language model agents (Processor, Compiler, and Validator) with specialized functions, and integrating the intermediate representation Visualization Query Language (VQL), this method progressively transforms complex natural language queries into structured visualization specifications and generates executable code. Leveraging staged reasoning and iterative validation mechanisms, NVAgent demonstrates high accuracy and robustness in multi-table join queries and complex logic modeling.

B.3 Evaluation Metrics

The evaluation metrics are formally defined as follows:

- **Vis_Type Accuracy (Vis.Acc)**: This metric evaluates the accuracy of visual components generated by the model within TVL, measuring the model's ability to identify visualization types. Its calculation formula is as follows:

$$Acc_{vis} = \frac{Num_{vis}}{Num} \tag{1}$$

Where Num_{vis} represents the count of exact match visualization type, Num represents the total number of test set.

- **Area Accuracy (Area.Acc)**: This metric evaluates the accuracy of the *Area* component within TVLs generated by the model to reveal its performance in identifying geographic area names. Its calculation formula is:

$$Acc_{area} = \frac{Num_{area}}{Num} \tag{2}$$

Where Num_{area} represents the count of exact matching *Area* components, Num represents the total number of test set.

- **Time Accuracy (Time.Acc)**: This metric evaluates the accuracy of the temporal component in the TVL generated by the model, revealing its performance in identifying temporal information. Its calculation formula is as follows:

$$Acc_{time} = \frac{Num_{time}}{Num} \tag{3}$$

Where Num_{time} represents the count of exact matching Time components, Num represents the total number of 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 within TVL. Specifically, it eliminates interference through normalization and recursively compares SQL structures via parse trees. Its calculation formula is as follows:

$$Acc_{sql} = \frac{Num_{sql}}{Num} \tag{4}$$

Where Num_{sql} represents the count of exact matching SQL queries, Num represents the total number of test set.

- **TVL Accuracy (TVL.Acc)**: This metric evaluates the accuracy of a model in generating complete TVL. Based on the structure of TVL, a model is considered to correctly generate TVL when it can accurately produce the *Vis*, *Area*, *ST*, *ET*, and *SQL*. The calculation formula is as follows:

$$Acc_{tvl} = \frac{Num_{tvl}}{Num} \tag{5}$$

Where Num_{tvl} represents the count of exact match TVLs, Num represents the total number of test set.

- **Pass Rate (Pass)**: This metric evaluates the code generation pass rate of the model, measuring the proportion of executable code generated by the model relative to the total data. Its calculation formula is as follows:

$$Acc_{pass} = \frac{Num_{pass}}{Num} \tag{6}$$

Where Num_{pass} represents the total amount of executable code, Num represents the total number of test set.

- **Aesthetic Accuracy (Aest.Acc)**: This metric evaluates the model's ability to recognize aesthetic attributes by calculating the proportion of predicted aesthetic attributes that perfectly match the ground truth (i.e., the percentage of total matches relative to the original total attributes) and taking the average. Its calculation formula is as follows:

$$Acc_{aes} = \frac{1}{N}\sum_{i=1}^{N}\frac{Num^{(i)}match}{Num^{(i)}attr} \quad (7)$$

Where N denotes the total number of samples in the test set, $Num^{(i)}match$ represents the number of aesthetic attributes whose predictions exactly match the ground truth for the i-th sample, and $Num^{(i)}attr$ represents the total number of ground-truth aesthetic attributes for the i-th sample.

- **Visualization Score (Score)**: This metric evaluates the visual quality generated by the model by assigning a score (0–100) to each visualization and ultimately taking the average. When the model fails to successfully output a visualization, the score is 0. Its calculation formula is as follows:

$$Score_{vis} = \frac{1}{N}\sum_{i=1}^{N} s_i \quad (8)$$

Where N denotes the total number of samples in the test set, and $s_i \in [0, 100]$ represents the visualization quality score of the i-th sample. If the model fails to generate a valid visualization for the i-th sample, then $s_i = 0$.

B.4 Implementation Details

In our experiments, we evaluate three large language models, including two open-source models, LLaMA3.1-8B and Qwen3-8B, as well as one proprietary model, GPT-5-mini. To ensure a fair comparison across different methods, the number of retrieved examples is fixed to 1-shot for all models. During the generation phase, we set the temperature of all language models to 0 to ensure the stability of the reasoning process and the generated results, and we limit the maximum generation length to 2048 tokens. During the visualization evaluation phase, we employ the multimodal large language model GLM-4.1V-9B-Thinking to assess the quality of generated visualizations and provide modification suggestions. Given that this stage involves understanding visual results and generating improvement directions, we set the model temperature to 0.5 to avoid overly conservative evaluation outputs, thereby obtaining more comprehensive and informative feedback.

B.5 Case Study

Figure 6 illustrates an example of a natural language query and its corresponding TVL and aesthetic attribute representation. The python code for visualization and the final visualization are shown in Figure 10 and Figure 7, respectively. We also presented examples of representative basic chart visualizations (Figure 8, Figure 11 and Figure 9).

Additionally, we present the visual output results for natural language query from user, generated by our proposed TrajVisAgent framework and three baseline frameworks (Figure 12). The results indicate that since NVAgent was unable to successfully extract the visual intermediate representation, the visualization for map types ultimately only produced error messages. The trajectory shapes visualized by Chat2VIS and CoML4Vis frameworks are correct, but they can only be displayed on charts, not on maps. These cases highlight the applicability and stable performance of our proposed framework on the NL2TrajVis task.

Natural Language Query:
Can you show me a map of the Lviv area that visualizes the migration tracks of Hydroprogne caspia within the time range from July 31, 2017 at 17:30 to April 10, 2020 at 08:18:43, focusing on speeds between 15.75 and 87.96, with points sized at 6 and tagged in brown, violet, and gray?

Corresponding TVL:

```
Visualize map
Area Lviv
ST 2017-07-31 17:30:00
ET 2020-04-10 08:18:43
SELECT h.datetime, h.longitude, h.latitude, h.tag_id
FROM animal.hydroprogne_caspia h
WHERE h.speed BETWEEN 15.75 AND 87.96
ORDER BY h.tag_id, h.datetime;
```

Corresponding Aesthetic:

```
{
  "display_mode": "points",
  "tag_color_map": [
    " brown",
    " violet",
    " gray"
  ],
  "point_size": 6
}
```

Figure 6: An Example of natural language query for map-type visualization and its corresponding TVL and aesthetic attribute representations.

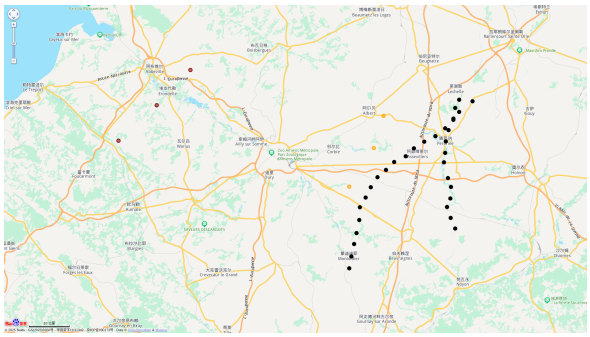


Figure 7: An example of generated map using TrajVisAgent.

Natural Language Query:
A line visualization of the maximum number of recovered COVID-19 cases, highlighting data points from countries within the range of 156.7 to 177.1. This plot features a line width of 2.5px, a cross point style, grid visibility, and the values clearly displayed.

Corresponding TVL:
Visualize line
Area null
ST null
ET null
SELECT d.date_value, MAX(d.recovered) AS max_recovered
FROM disease.daily_covid_stats d
WHERE d.num_countries BETWEEN 156.7 AND 177.1
GROUP BY d.date_value, d.recovered **ORDER BY** d.date_value

Corresponding Aesthetic:

```
{
 "grid_visibility": true,
 "point_style": "cross",
 "show_values": true,
 "line_width": "2.5px"
}
```


Figure 8: An Example of natural language query for chart-type visualization and its corresponding TVL and aesthetic attribute representations.

B.6 Prompt Template for TVL Agent

```
Given some [Examples], a [Database schema] and
a [Question], generate valid TVL (Trajectory
Visualization Language) sentences and the
Aesthetic Preference JSON.

====================
## TASK OVERVIEW
Given:
- Examples
- Database Schema
- Question

Your task:
1. Generate the final TVL sentences.
2. Generate the Aesthetic Preference JSON.

====================
## Background
### The Structure of TVL:
Visualize [Visualization Type] Area [Geographic
 Area] ST [Start Time] ET [End Time] SELECT [
COLUMNS] FROM [TABLES] [JOIN] [WHERE] [GROUP BY
] [ORDER BY] [LIMIT]

### Key Elements:
1. Visualization Type:
Choose exactly one from: map, bar, pie, line,
scatter.
2. Geographic Area:
Extract geographic or regional information if
mentioned.
3. Time Range:
Extract start time (ST) and end time (ET).
```

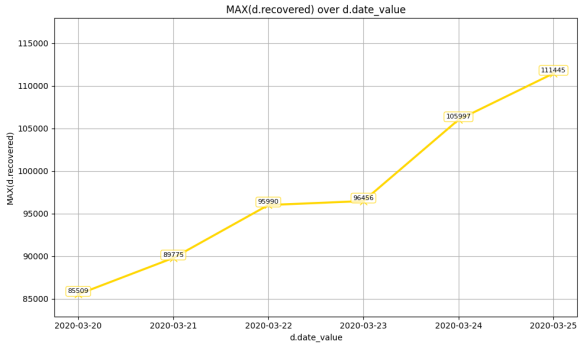


Figure 9: An example of generated line chart using TrajVisAgent.

```
Time format must be: YYYY-MM-DD HH:MM:SS.
4. SQL Clauses:
Use only necessary SQL clauses:
SELECT, FROM, JOIN, WHERE, GROUP BY, ORDER BY,
LIMIT.

====================
## Special Rules
A. For bar, line, pie, scatter:
- SELECT exactly two columns: X-axis and Y-axis.
 Y-axis should usually be an aggregated value.

B. Aggregation:
- Use COUNT for counting records.
- Use SUM only for numeric columns.
- Use EXTRACT to derive month or other units
from datetime.

C. Constraints:
- Enclose string literals in single quotes.
- SELECT must contain at least two columns.
- Use only table and column names from the
given schema.
- If a column may contain NULL/None, filter
using:
WHERE <column> IS NOT NULL, or use JOIN if
required.
- GROUP BY must appear before ORDER BY.

====================
## Examples
### PHASE 1 TVL Generation

{examples_prompt}

### PHASE 2 Aesthetic Preference (JSON)

{examples_aesthetic}

====================
## Database Schema:
{db_schema}

====================
## Question:
{query}

====================
## Output Format
Final TVL:
```

```
Aesthetic Preference (JSON):
{{
  "key": "value"
}}
```

B.7 Prompt Template for Code Agent

```
====================
## Task:
You are an expert specializing in data
visualization and Python. Your task is to fix a
 segment of Python code that is causing errors.
 Please systematically approach this task,
thinking through each step step by step.

====================
## Query:
{query}

====================
## Database Schema:
{db_info}

====================
## Current Python Code:
{code}

====================
## Error:
{error}

====================
Now, please analyze and refine the Python code.
 Provide:

### Explanation

### Corrected Python Code
```python

```

Remember:
- The code must be executable in a Python
environment.
- Ensure the visualization matches the
requirements of the original query.
```

B.8 Prompt Template for Visual Agent

```
====================
## Role
You are a seasoned data visualization and
Python development expert. Given a **User Query
**, **Current Code**, and the current **chart
image**, determine whether the chart faithfully
 reflects the user's query. Focus solely on the
 visual aesthetic requirements, there is no
need to focus on the data.

====================
## Task
Your task is to provide guidance ensuring the
chart strictly meets the query requirements.
Deliver detailed steps for modifying the Python
 code to enhance the chart.

====================
## Analytical Framework
1. **Intent Extraction:** Analyze the user
query to identify key constraints: chart type,
labels, titles, colors, and any other visual
elements.

2. **Visual Audit:** Compare these constraints
against the provided chart, documenting
discrepancies between the user's aesthetic
requirements and the current chart image.

3. **Solution:** Based on identified
discrepancies, provide detailed step-by-step
instructions guiding how to modify Python code
to meet user query requirements.

4. **Optimization Recommendations:** Propose
enhancements to improve visualization clarity,
readability, and aesthetics while ensuring the
primary goal remains fulfilling the user's
specified needs.

====================
## User Query:
{query}

====================
## Current Code:
{code}

====================
## Output Format
(Do not display the thought process, output
only the final answer.)

1. Code Revision Guidelines:

2. Optimization Suggestions:
```

```python
import json
import pandas as pd
from sqlalchemy import create_engine

DB = {"host": "localhost", "database": "traj", "user": "admin", "password": "admin", "port": 5433}
SQL = """SELECT h.datetime, h.longitude, h.latitude, h.tag_id
FROM animal.hydroprogne_caspia h JOIN boundaries b ON b.name = 'Lviv'
WHERE ST_Within(ST_SetSRID(ST_MakePoint(h.longitude, h.latitude), 4326), b.geom)
AND h.datetime BETWEEN '2017-07-31 17:30:00' AND '2020-04-10 08:18:43'
AND h.speed BETWEEN 15.75 AND 87.96 ORDER BY h.tag_id, h.datetime;"""

STYLE = {"mode": "points", "colors": ["brown", "violet", "gray"], "size": 6}

engine = create_engine(f"postgresql://{DB['user']}:{DB['password']}@{DB['host']}:{DB['port']}/{DB['database']}")
df = pd.read_sql(SQL, engine)

center = (df['longitude'].mean(), df['latitude'].mean())
trajs = []
for i, tag in enumerate(df['tag_id'].unique()):
    coords = df[df['tag_id'] == tag].sort_values('datetime')[['longitude', 'latitude']].values.tolist()
    trajs.append({'tag': tag, 'color': STYLE['colors'][i], 'coords': coords})

html = f'''<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <style>
        * {{ margin: 0; padding: 0; }}
        html, body {{ width: 100%; height: 100%; }}
        #map {{ width: 100%; height: 100%; }}
    </style>
    <script src="https://api.map.baidu.com/api?v=3.0&ak={BAIDU_AK}"></script>
</head>
<body>
    <div id="map"></div>
    <script>
        var map = new BMap.Map("map");
        map.centerAndZoom(new BMap.Point({center_lng}, {center_lat}), 13);
        map.enableScrollWheelZoom(true);
        var trajectories = {json.dumps(trajectory_data)};
        var displayMode = "{AESTHETIC['display_mode']}";
        var pointSize = {AESTHETIC['point_size']};
        var lineWidth = {AESTHETIC['line_width']};

        trajectories.forEach(function(traj) {{
            var coords = traj.coords;
            var color = traj.color;

            if (displayMode === "points") {{
                coords.forEach(function(coord) {{
                    var point = new BMap.Point(coord[0], coord[1]);
                    var marker = new BMap.Marker(point, {{
                        icon: new BMap.Symbol(BMap_Symbol_SHAPE_CIRCLE, {{
                            scale: pointSize, fillColor: color, f illOpacity: 0.8, strokeColor: color,  strokeWeight: 2
                        }})
                    }});
                    map.addOverlay(marker);
                }});
            }} else {{
                var points = coords.map(function(c) {{
                    return new BMap.Point(c[0], c[1]);
                }});
                var polyline = new BMap.Polyline(points, {{
                    strokeColor: color, strokeWeight: lineWidth, strokeOpacity: 0.9
                }});
                map.addOverlay(polyline);
            }}
        }});
    </script>
</body>
</html>'''

output_dir = Path("TrajVisAgent/generated_html_gpt")
output_dir.mkdir(exist_ok=True)
(output_dir / "map.html").write_text(html, encoding='utf-8')
```

Figure 10: An example of a python code module generated by TrajVisAgent.

```python

import pandas as pd
import matplotlib.pyplot as plt
from sqlalchemy import create_engine

DB_CONFIG = {
    "host": "localhost",
    "database": "traj",
    "user": "admin",
    "password": "admin",
    "port": 5433
}

SQL = """
SELECT d.date_value, MAX(d.recovered) AS max_recovered
FROM disease.daily_covid_stats d
WHERE d.num_countries BETWEEN 156.7 AND 177.1
GROUP BY d.date_value, d.recovered
ORDER BY d.date_value
"""

AESTHETIC = {
    "grid_visibility": True,
    "point_style": "cross",
    "show_values": True,
    "line_width": 2.5
}

engine = create_engine(
    f"postgresql+psycopg2://{DB_CONFIG['user']}:{DB_CONFIG['password']}"
    f"@{DB_CONFIG['host']}:{DB_CONFIG['port']}/{DB_CONFIG['database']}"
)

df = pd.read_sql(SQL, engine)

x_col = df.columns[0]
y_col = df.columns[1]
df[y_col] = pd.to_numeric(df[y_col], errors='coerce').dropna()

plt.figure(figsize=(10, 6))
plt.plot(df[x_col], df[y_col],
         color='blue',
         linewidth=AESTHETIC['line_width'],
         marker='x',
         markersize=8)

if AESTHETIC['show_values']:
    for x, y in zip(df[x_col], df[y_col]):
        plt.text(x, y, f'{y:.0f}', ha='center', va='bottom', fontsize=8)

plt.xlabel(x_col)
plt.ylabel(y_col)
plt.title(f"{y_col} over {x_col}")
plt.grid(AESTHETIC['grid_visibility'])
plt.tight_layout()
plt.savefig('output.png', dpi=100, bbox_inches='tight')
plt.close()
```

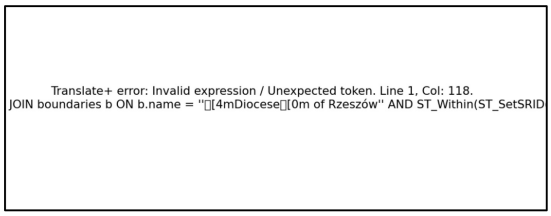
Figure 11: An example of a python code module generated by TrajVisAgent.

Natural Language Query:

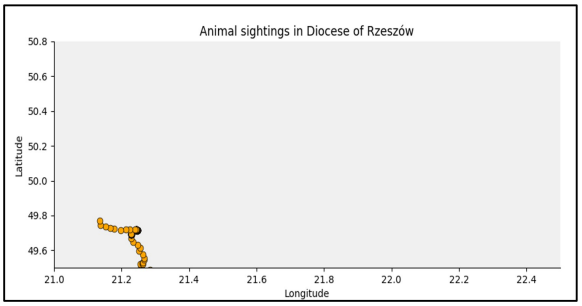
Display a map of the Diocese of Rzeszów showing the movement trajectory of Ciconia ciconia, with points represented as orange circles for animal sightings between March 27, 2025, at 11:50:08 and March 28, 2025, at 08:20:06, where the altitude is between 275.0 and 931.0.

TrajVisAgent Response:

NVAgent Response:



Chat2VIS Response:



CoML4Vis Response:

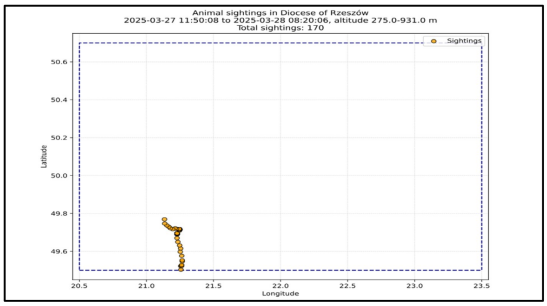


Figure 12: Examples of visualization generated using different frameworks for natural language query.