

# NL4ST: A Natural Language Query Tool for Spatio-Temporal Databases

Xieyang Wang, Mengyi Liu, Weijia Yi, Jianqiu Xu\*  
 Nanjing University of Aeronautics and Astronautics  
 Nanjing, China  
 {xieyang,liumengyi,wjyi\_x,jianqiu}@nuaa.edu.cn

Raymond Chi-Wing Wong  
 The Hong Kong University of Science and Technology  
 Hong Kong, China  
 raywong@cse.ust.hk

## Abstract

The advancement of mobile computing devices and positioning technologies has led to an explosive growth of spatio-temporal data managed in databases. Representative queries over such data include *range queries*, *nearest neighbor queries*, and *join queries*. However, formulating those queries usually requires domain-specific expertise and familiarity with executable query languages, which would be a challenging task for non-expert users. It leads to a great demand for well-supported natural language queries (NLQs) in spatio-temporal databases. To bridge the gap between non-experts and query plans in databases, we present *NL4ST*, an interactive tool that allows users to query spatio-temporal databases in natural language. NL4ST features a three-layer architecture: (i) *knowledge base and corpus for knowledge preparation*, (ii) *natural language understanding for entity linking*, and (iii) *generating physical plans*. Our demonstration will showcase how NL4ST provides effective spatio-temporal physical plans, verified by using four real and synthetic datasets. We make NL4ST online<sup>1</sup> and provide the demo video at <https://youtu.be/-J1R7R5WoqQ>.

### ACM Reference Format:

Xieyang Wang, Mengyi Liu, Weijia Yi, Jianqiu Xu, and Raymond Chi-Wing Wong. 2026. NL4ST: A Natural Language Query Tool for Spatio-Temporal Databases. In *Proceedings of . ACM*, New York, NY, USA, 5 pages. <https://doi.org/XXXXXX.XXXXXXX>

## 1 Introduction

In the past decades, although spatio-temporal databases have been extensively investigated in a wide range of applications [6], the assumption that users can efficiently formulate accurate queries poses challenges for non-expert users who are unfamiliar with spatio-temporal knowledge and structured database query languages. To this end, there has been an increasing attention paid to providing a natural language interface for databases (NLIDB) [3].

In the literature, significant advances have been made for Text-to-SQL [1, 4, 8]. However, these approaches primarily focus on relational databases and treat SQL as the final semantic representation.

<sup>1</sup><https://nl4st.cpolars.top/nl2secondo/>

\*Corresponding Author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM. <https://doi.org/XXXXXX.XXXXXXX>

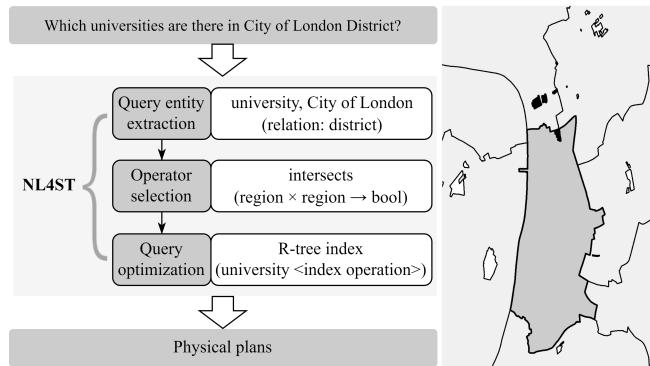


Figure 1: An example of transforming NLQs in spatio-temporal databases.

From a semantic perspective, SQL lacks native spatio-temporal operators and often leads to ambiguous mappings from natural language queries (NLQs). For example, the phrase “in City of London District” in Figure 1 may map to different spatial predicates in SQL, resulting in ambiguous constructions such as “*contains (tuple.location)*” or “*inside (tuple.location)*”. From an execution perspective, SQL is declarative and hides execution decisions, as operator selection and index usage are deferred to the optimizer. As a result, SQL fails to explicitly capture the execution intent of queries.

In this demo, we investigate **Text-to-query-plan** for spatio-temporal databases, which directly maps NLQs to query plans, including logical and physical plans. Logical plans describe domain-specific query semantics through algebraic operators, while physical plans specify domain-specific execution operators and index usage. We use the physical plan as the output, which allows the Text-to-query-plan translation to directly capture the semantics of NLQs and explicitly control the execution semantics and efficiency.

Although query plans offer a nuanced approach to data manipulation, Text-to-query-plan presents the following three challenges.

**Challenge 1. The Difficulty of Applying Text-to-SQL Methods to Text-to-query-plan.** Existing Text-to-SQL methods rely on large language models (LLMs), while spatio-temporal query plans require rich domain knowledge and complex executable structures, making training data construction costly. Moreover, current efficiency-control techniques generally assume a black-box context, raising challenges in (i) *domain knowledge alignment* and (ii) *correct and efficient plan generation*.

**Challenge 2. Lack of Domain-specific Knowledge.** The lack of domain-specific knowledge in LLM-based systems causes hallucinations and incorrect entity alignment. As shown in Figure 1, “*City of London District*” is often misaligned by GPT-4o at the column level, although the entity corresponds to a specific value-level representation, highlighting the necessity of domain-aware entity grounding to enhance natural language understanding.

**Challenge 3. The Complexity of Query Plan Generation.** The large combination space of operators and indexes makes physical plan generation difficult to scale while maintaining efficiency. For the NLQ in Figure 1, GPT-4o employs the *ininterior* operator instead of *intersect* and thus fails to utilize available indexes.

There are limited studies about NLIDB in the spatio-temporal domain [5, 7], as integrating spatial and temporal dimensions in a unified framework introduces complexity.

We develop NL4ST, a natural language query tool for spatio-temporal databases, that enables users to query spatial, temporal, and trajectory data using natural language. In comparison with Text-to-SQL, the approach of Text-to-query-plan (without involving the intermediate result SQL) is able to (i) *avoid ambiguity in text languages* and (ii) *empower the expression by involving operators and data types from specific domains*. The tool consists of three layers: (i) *knowledge base and corpus*, (ii) *natural language understanding*, and (iii) *generating physical plans*. NL4ST generates knowledge bases through database analysis and constructs a corpus by collecting NLQs from published papers on top conferences and journals, which are further augmented by LLMs. The knowledge bases and corpus are leveraged to enable knowledge retrieval and example reference, guiding users in submitting robust queries, which can also be independently utilized for subsequent natural language understanding. NL4ST utilizes natural language processing (NLP) tools for coarse-grained entity extraction, followed by fine-grained entity extraction using an advanced algorithm and the knowledge base. NL4ST leverages a query type classification model and structured language models corresponding to query types, which prunes the candidate set for physical plan generation. In addition, NL4ST employs a model to predict the optimal physical plan. The system is published online and can be accessed at <https://nl4st.cpolar.top/nl2seconde/>.

## 2 System Architecture

We provide the framework of NL4ST in Figure 2, which adopts a three-layer architecture: (i) *knowledge preparation*, (ii) *natural language understanding*, and (iii) *physical plan generation*.

### 2.1 Knowledge Preparation

NL4ST addresses the lack of spatio-temporal domain knowledge through (i) *the knowledge base generator* and (ii) *the corpus generator*.

**Knowledge Base Generator.** We construct a spatio-temporal knowledge base to support knowledge retrieval and entity candidate pruning. The knowledge base consists of a relation knowledge base and a location knowledge base. The relation knowledge base stores relational tables with identifiers, names, and spatio-temporal attributes, while the location knowledge base contains objects with *point*, *line*, and *region* representations. The two knowledge bases

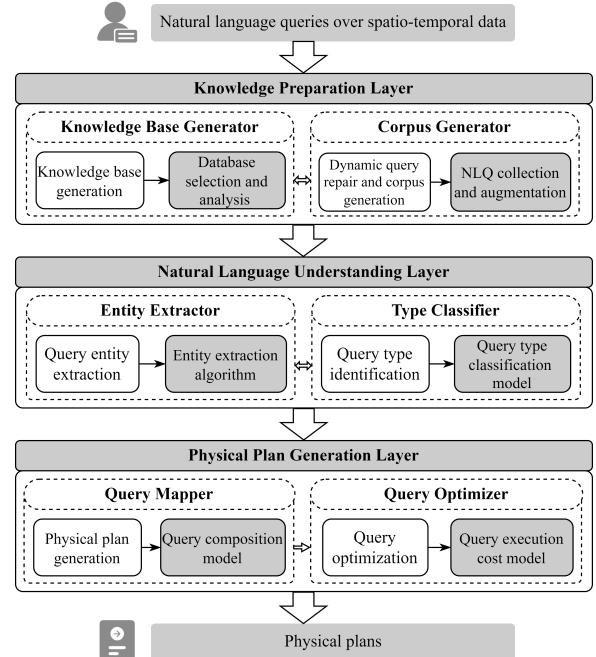


Figure 2: The architecture overview of NL4ST.

are tightly coupled, where location entries reference identifiers defined in the relation knowledge base.

**Corpus Generator.** We construct an NLQ corpus with about 5,000 templates covering *spatial queries* and *moving objects queries* to support user guidance and subsequent query type identification. NLQs are gathered from published papers and augmented by GPT-4o with designed prompts. To ensure alignment with databases, an automatic detection and repair method is employed to refine the NLQs. We also construct templates to enable the generation of a large volume of high-quality NLQs tailored to different databases. The corpus consists of query types and NLQs. The relations and entities in the NLQs are derived from selected databases.

### 2.2 Natural Language Understanding

NL4ST primarily performs NLP, including two modules: (i) *the entity extractor* and (ii) *the type classifier*.

**Entity Extractor.** The extractor aligns the entities in NLQs with database values. *spaCy* is used for initial entity recognition, which is an NLP tool, followed by candidate entity generation into two lists: (i) *the number list*, containing entities labeled with time, number, cardinal, or quantity, and (ii) *the information list*, potentially containing relations, locations, or objects. Then, we prune candidates by a fine-grained extraction algorithm to extract the exact key entities. The object identifier and the nearest neighbor identifier encompass the contextual information pertaining to the object and the keyword like “nearest”. Leveraging the number and the associated distance unit in the phrase, the precise value of the distance threshold is determined. Besides, we utilize the constructed spatio-temporal knowledge base to perform direct retrieval of relations, objects, and location information in the information list.

**Type Classifier.** Given the diversity of spatio-temporal queries, providing a general translation model for all query types is impractical. Identifying and categorizing query types significantly narrows the search space for constructing physical plans. LSTM offers good performance and reduced training time, and thus, we employ LSTM for model training by the given corpus. The trained model can be utilized for type identification.

### 2.3 Physical Plan Generation

NL4ST advances to the physical plan generation, including two key modules: (i) *the query mapper* and (ii) *the query optimizer*.

**Query Mapper.** We construct the candidate physical plans by combining entities and operators required for different query types through mapping rules, based on a pre-defined query model. We take a moving object nearest neighbor query as an example, limited by space. The operator *knearest* generates a tuple stream that mirrors the structure of the input stream. Limited by the time when objects arrive among the  $k$  nearest moving objects, we create a version of *moving trains*, where units are sorted by the start time. We can obtain the physical plan by using the relation *UTOrdered*. The template of the query expression is as follows, and the content in “ $\diamond$ ” will be extracted from NLQs.

```
query UTOdered feed filter [(deftime(UTrip) intersects < period
))] knearest[UTrip, < object >, < k >]consume;
```

**Query Optimizer.** To further enhance the user-friendliness, we optimize query efficiency by focusing on index utilization. We count and collect the number of records involved in each spatio-temporal relation. Then, we estimate the filter rate based on the operator constraints and the relations involved in the query, where the filter rate refers to the percentage of records that meet specific query criteria relative to the total number of records in the dataset. For queries with a high filter rate, we enumerate candidate physical plans with indexes for queries generated by the query mapper. Subsequently, we employ a cost model to execute the physical plans with sampled small-scale data to predict the temporal overhead of each candidate physical plan, ultimately providing users with the optimal physical plan.

## 3 Demonstration

We deploy NL4ST in a desktop PC (Intel(R) Core(TM) i9-11950H CPU, 2.60 GHz, 32 GB memory, 512 GB disk) running Ubuntu 22.04 (64 bits, kernel version 6.8.0-49-generic). We provide an interactive web UI for users, and the backend is integrated into an extensible database system SECONDO [2] as an algebra module. Users perform queries based on the loaded datasets and obtain physical plans as well as visualization results. We leverage four datasets, three of which pertain to urban data<sup>2</sup> while the fourth is focused on hydrological data<sup>3</sup>. (i) The dataset *nanjingtest* contains taxis, districts, part of roads and POIs in Nanjing. (ii) The dataset *londontest* includes a subset of POIs and districts in London. (iii) The dataset *berlintest* encapsulates urban data of Berlin, encompassing trains, POIs, and rivers. (iv) The dataset *chinawater* stores partial water systems in China. Detailed statistics are reported in Table 1. We demonstrate the capacity of NL4ST to transform spatio-temporal

<sup>2</sup><https://github.com/zongmove/NL4ST>

<sup>3</sup><https://www.hydrosheds.org/products>

NLQs into physical plans, aiming to provide an interactive tool for users to learn the working mechanisms of the Text-to-query-plan techniques introduced above.

**Table 1: Data statistics.**

Datasets	#tables	#points	#lines	#regions	#moving objects
nanjingtest	6	9,000	887	13	147
londontest	6	9,032	9,728	12,669	0
berlintest	50	3,040	4,078	330	562
chinawater	2	0	8,399	2,907	0

### 3.1 Demonstration Scenarios

Figure 3 presents the screenshot of NL4ST, which enables users to perform (i) *basic spatial query*, (ii) *time interval query*, (iii) *range query*, (iv) *nearest neighbor query*, (v) *join query*, (vi) *similarity query*, and (vii) *aggregation query*. We perform the demonstration in the following steps and provide two query scenarios.

**Corpus Generation and Example Reference.** Initially, users can select varying quantities of corpus data for generation and utilize the generated spatio-temporal query corpus to train their own models. These NLQ examples in the corpus can also guide users in formulating high-quality queries.

**Knowledge Retrieval.** Subsequently, users can retrieve entities from the pre-constructed knowledge base, which can be utilized for formulating subsequent NLQs with referencing examples.

**Database Selection and NLQ Input.** Given that the schema affects the execution of the intended physical plan, users must first select the target database for the query. To simplify this step, users can check sample queries that can be selected and used as input directly. After inputting the NLQ, users can click the “Submit” button.

**Execution Result Display.** After submission, the tool presents the parsing process. This process first performs coarse-grained entity recognition, followed by fine-grained entity disambiguation, and then conducts detailed entity extraction for query template instantiation. Based on the extracted information, the system generates a physical plan and returns the corresponding query results.

**Map Visualization and Interaction.** Users can click the “Show in SECONDO” button to view visualization results inside the database. The tool also visualizes the results on an interactive map. For each query type, users can get a tailored visualization design, which also guides users to detect the real-world use cases of the results.

**Error Handling.** When the input contains an unsupported query type, a syntax error, or an entity related error, users can click the “Help” button. The tool explains the cause of the error and provides possible corrections. Users can revise the input based on the suggestions and resubmit the query.

**Query Efficiency Display.** Finally, the tool reports the execution time of the physical plan. Users can click the “Switch” button to compare the execution time of the original query with that of the optimized query. Users can also inspect the physical plan through a visual operator tree via the “Operator Tree” button.

**Table 2: Example queries.**

Example NLQ	Type
Q1: What is the fastfood at each university in London?	Spatial join query
Q2: Show me fifty nearest neighbors to the train 5 between 6am and 11am.	Moving objects nearest neighbor query

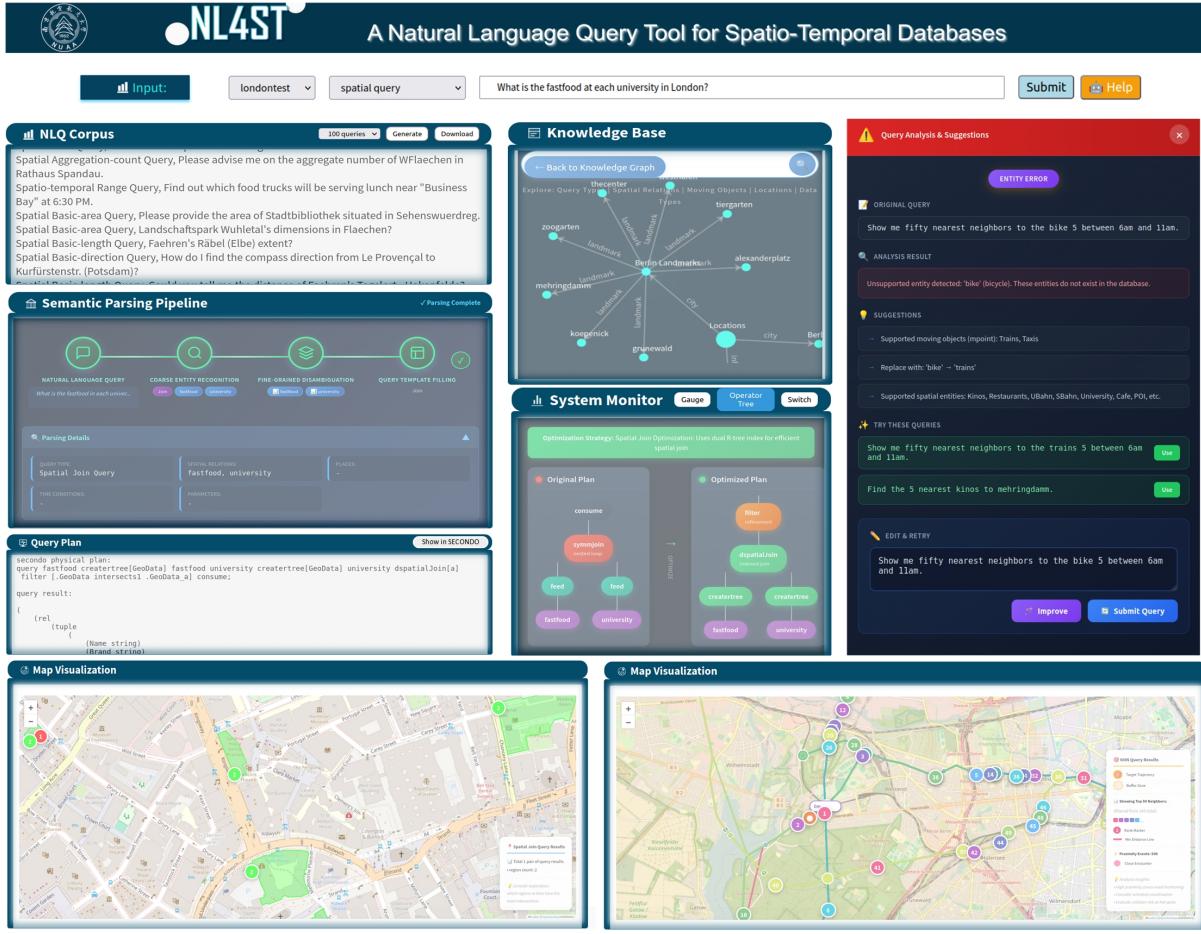


Figure 3: The screenshot of NL4ST.

**Scenario 1: Querying Spatial Data.** We consider the spatial join query  $Q_1$  provided in Table 2 and demonstrate the system workflow following the steps described above. Users can first inspect the corpus and the knowledge base, which provide a set of standard queries and entities for reference. Users then select a target database and submit an NLQ, optionally choosing from the recommended queries. The detailed semantic parsing process is presented in Figure 3, based on which the system generates a physical plan, and the query results are visualized on the map. NL4ST applies an R-tree index for optimization, and users can examine the system monitor to obtain the query execution time. The query execution process is also exposed through a visual operator tree, which can be accessed by clicking the “Operator Tree” button.

**Scenario 2: Querying Moving Objects.** The nearest neighbor query  $Q_2$  showcased in Table 2 also follows the above steps to generate the results and provide a visualization. In the *Map Visualization* module, NL4ST uses connecting lines to indicate the nearest neighbor order and distance. This visualization guides users to detect real-world use cases, such as identifying locations or time periods where moving objects are in close proximity. If existing issues, such as entity related errors as shown in Figure 3, users can click the “Help” button to obtain error explanations and guidance. In

the analysis result area, users can inspect detailed error messages, while the suggestions area provides potential corrections. Users can also directly try the recommended queries listed.

### 3.2 Evaluation Metrics

We evaluate NL4ST using three metrics: (i) *response time*, (ii) *translatability*, and (iii) *translation precision*. Translatability measures the fraction of NLQs that can be translated into physical plans, while translation precision measures the fraction of translated physical plans that produce expected results. The average response time, translatability, and translation precision of NL4ST are 1.9 seconds, 93%, and 90%, respectively.

### References

- [1] Jiaqi Guo, Zecheng Zhan, Yan Gao, and et al. 2019. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. In *ACL*. 4524–4535.
- [2] Ralf Hartmut Güting, Thomas Behr, and Christian Düntgen. 2010. SECONDO: A Platform for Moving Objects Database Research and for Publishing and Integrating Research Implementations. *IEEE Data Eng. Bull.* 33, 2 (2010), 56–63.
- [3] George Katsogiannis-Meimarakis, Mike Xydas, and Georgia Koutrika. 2023. Natural Language Interfaces for Databases with Deep Learning. *Proc. VLDB Endow.* 16, 12 (2023), 3878–3881.
- [4] Fei Li and H. V. Jagadish. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. *Proc. VLDB Endow.* 8, 1 (2014), 73–84.

- [5] Mengyi Liu, Xieyang Wang, Jianqiu Xu, and et al. 2025. NALSpatial: A Natural Language Interface for Spatial Databases. *IEEE Trans. Knowl. Data Eng.* 37, 4 (2025), 2056–2070.
- [6] Yongxin Tong, Jieying She, Bolin Ding, and et al. 2016. Online mobile Micro-Task Allocation in spatial crowdsourcing. In *ICDE*. 49–60.
- [7] Xieyang Wang, Mengyi Liu, Jianqiu Xu, and et al. 2023. NALMO: Transforming Queries in Natural Language for Moving Objects Databases. *Geoinformatica* 27, 3 (2023), 427–460.
- [8] Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and et al. 2025. OpenSearch-SQL: Enhancing Text-to-SQL with Dynamic Few-shot and Consistency Alignment. *Proc. ACM Manag. Data* 3, 3 (2025), 194:1–194:24.