NALMOBench: Towards Benchmarking Natural Language Interfaces for Moving Objects Databases

Xieyang Wang
Nanjing University of Aeronautics and Astronautics, China
xieyang@nuaa.edu.cn

Weijia Yi
Nanjing University of Aeronautics and Astronautics, China
wjyi_x@nuaa.edu.cn

Mengyi Liu*
Nanjing University of Aeronautics and Astronautics, China
liumengyi@nuaa.edu.cn

Chenchen Zong
Nanjing University of Aeronautics and Astronautics, China
chencz@nuaa.edu.cn

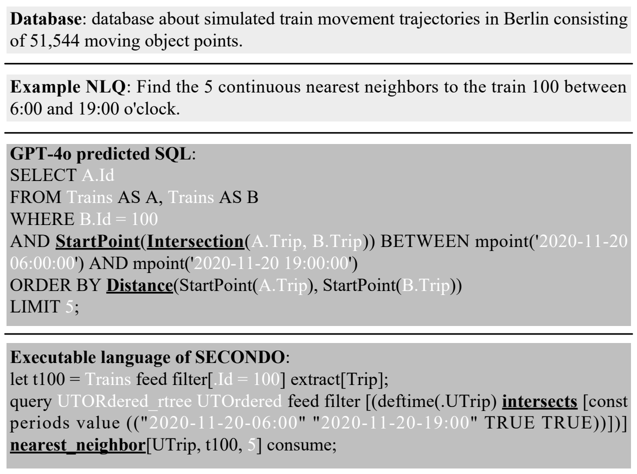
ABSTRACT

Existing natural language interfaces for databases (NLIDBs) benchmarks target relational databases and lack unified quality metrics, support for complex spatio-temporal multi-dimensional queries, and real-world validation of translation accuracy and efficiency. To fill this gap, we propose a benchmark for natural language query (NLQ) translation in moving objects databases (MODs). The benchmark covers five commonly encountered moving objects queries and two real-world scenario queries. Such NLQs are generated to construct the corpus with the usage of large language models (LLMs). Robust NLQs are extracted from the corpus and annotated with the assistance of automatic processes and domain experts. The annotation process incorporates formal verification to ensure query quality. We propose a two-phase query transformation baseline method, including natural language understanding and structured language generation for comprehending the intent of NLQs and generating optimized executable languages, respectively. Experimental results indicate that existing methods struggle to be directly applied in designing NLIDBs for MODs, failing to effectively handle complex MOD queries and lack precise control over query efficiency. This highlights significant research potential in NLIDB for MODs. The method proposed in this paper achieves average translatability, translation precision, and translation efficiency score of 68.96%, 60.07%, and 68.37, respectively.

VLDB Workshop Reference Format:
Xieyang Wang, Weijia Yi, Mengyi Liu, and Chenchen Zong. NALMOBench: Towards Benchmarking Natural Language Interfaces for Moving Objects Databases. VLDB 2025 Workshop: LLM+Spatial.

1 INTRODUCTION

In recent years, the development of spatio-temporal databases has been driven by an ever-increasing demand for sophisticated management of location-aware data. Natural language, being the most familiar means of communication, has become the preferred choice for non-experts to use databases. Indeed, advances in LLM-based text-to-SQL techniques have catalyzed the development of natural language interfaces for domain-specific databases. However, standard SQL falls short of capturing the rich spatial and temporal operators required for deep spatio-temporal semantic reasoning, whereas native spatio-temporal engines expose an extensive set of executable languages that can express complex queries [12]. To bridge this gap, researchers have begun exploring direct translation from natural language into executable languages (text-to-EXE). Yet this emerging field lacks a unified benchmark and public datasets. Referring to the maturity of text-to-SQL benchmarks, we need to discuss the following three questions.



Database: database about simulated train movement trajectories in Berlin consisting of 51,544 moving object points.

Example NLQ: Find the 5 continuous nearest neighbors to the train 100 between 6:00 and 19:00 o'clock.

GPT-4o predicted SQL:
```
SELECT A.Id
FROM Trains AS A, Trains AS B
WHERE B.Id = 100
AND StartPoint(Intersection(A.Trip, B.Trip)) BETWEEN mpoint('2020-11-20 06:00:00') AND mpoint('2020-11-20 19:00:00')
ORDER BY Distance(StartPoint(A.Trip), StartPoint(B.Trip))
LIMIT 5;
```

Executable language of SECONDO:
```
let t100 = Trains feed filter[.Id = 100] extract[Trip];
query UTORdered_rtree UTOrdered feed filter [(deftime(.UTrip) intersects [const periods value (("2020-11-20-06:00" "2020-11-20-19:00" TRUE TRUE))])]
nearest_neighbor[UTrip, t100, 5] consume;
```

Figure 1: An example of GPT-4o predicted SQL and the corresponding executable language of SECONDO. White parts are entities. Bold and underlined parts are operators used for explaining the structured semantics expression of nearest.

Q1: Why do we need text-to-EXE compared to text-to-SQL for moving objects databases? While significant advances have been made, most achievements focus on relational databases, such as the rule-based method NaLIR [6], neural network-based method IRNet [3], and LLM-based methods like FinSQL [16]. Relational databases using SQL for querying may encounter challenges when

*Corresponding author

Proceedings of the VLDB Endowment. ISSN 2150-8097.

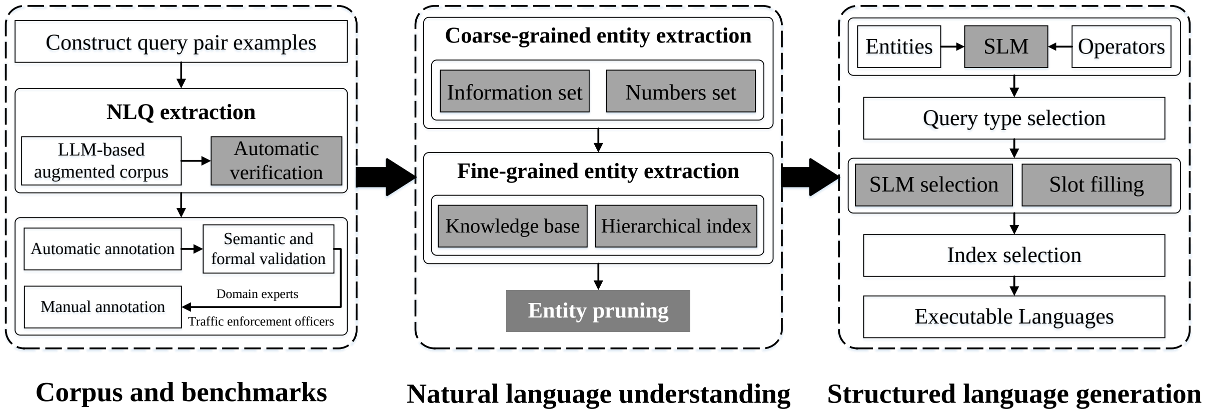


Figure 2: Overview of the benchmark.

handling complex moving objects queries involving multidimensional data. Existing NLIDBs often depend on optimizers for query optimization, which can result in reduced control over data and performance. In contrast, executable languages offer the possibility of solving semantic parsing and query translation due to the support for a wide range of spatio-temporal queries and operators, and can directly optimize the query with index usage and operator selection. As illustrated in Figure 1, the SQL generated by GPT-4o necessitates a comprehensive combination of operators and predicates to accurately represent the semantics of a nearest neighbor query. Executable languages can convey the same semantics just using the *nearest_neighbor* operator. Additionally, the executable database query can optionally utilize the 3D R-tree index, represented by *UTOrdered_rtree*. Note that subsequent executable languages are exemplified using the executable languages of SECONDO [4], which is a real MOD system.

Q2: Can LLMs dominate the text-to-EXE landscape? As mentioned previously, LLM-based methods now dominate the text-to-SQL landscape, leading us to consider whether LLMs can be effectively used for text-to-EXE [9]. Yet LLMs face two principal obstacles when generating executable languages. On one hand, existing approaches rely on task-specific corpus for executable languages, which often necessitates prompt engineering or costly fine-tuning with newly created datasets. Relying on prompts introduces a trade-off in accuracy of entity linking, operator selection, and index usage. Additionally, the design and creation of specialized corpus for fine-tuning are often cost-prohibitive. On the other hand, even with the usage of prompts or fine-tuning, the accuracy of entity linking in the text-to-SQL domain remains below 75%, let alone for complex spatio-temporal queries. To comprehensively investigate whether existing translation methods and LLMs can offer valuable insights for text-to-EXE mechanisms in MODs, a benchmark is demanded.

Q3: How to evaluate text-to-EXE systems? Mainstream benchmarks primarily focus on text-to-SQL evaluation, failing to address the complex spatio-temporal query requirements of MODs. Early datasets like Geoquery [15] only provide NLQs without corresponding SQL. While WikiSQL [18] offers NLQ and SQL pairs but only for simple queries. Spider [14] and Bird [7] support cross-domain queries but target fixed datasets, lacking specialized evaluation for complex MOD queries. MOD queries typically represent real-world needs. Current benchmarks exhibit two fundamental limitations: (i) the text-to-SQL focus neglects domain-specific text-to-EXE evaluation and (ii) existing assessments lack real-world moving objects scenario validation, overemphasizing translation accuracy while underrepresenting executable query efficiency. These gaps necessitate specialized benchmarks that capture domain-specific characteristics and semantics, coupled with a comprehensive evaluation framework balancing both accuracy and efficiency.

Challenges. To construct an in-depth benchmark for text-to-EXE functionality within the context of MODs, we should tackle three primary challenges: (i) ***The lack of domain knowledge-based benchmark construction***. The complex domain knowledge of MODs and the wide range of extensible operators pose difficulties for the construction and annotation of NLQs. (ii) ***The hardness of natural language understanding***. Both spatial and temporal information can be ambiguous. The synonymous, abbreviated and ambiguous entities need to be considered when designing queries. (iii) ***The complexity of structured language generation***. Due to the diversity of operators and richness of query types, structured language generation must simultaneously address the composition of entities and operators, and query optimization challenges.

To address these limitations, we present NALMOBench, an LLM-driven benchmark for evaluating text-to-EXE translation in MODs. The complex real-world benchmark assesses the performance of text-to-EXE systems and LLMs. Our proposed LLM-based automated corpus generation method efficiently produces large-scale annotated text-to-EXE datasets with domain generalization capabilities, while simultaneously addressing natural language understanding and structured language generation challenges. We also provide baseline methods for comparison. To the best of our knowledge, NALMOBench represents the first benchmark collaboratively developed by spatio-temporal database experts and traffic law enforcement officers. The benchmark comprehensively supports standard moving objects queries, including time interval queries, range queries, nearest neighbor queries, join queries, and trajectory similarity queries, and real-world traffic enforcement scenarios. Unique among existing benchmarks, NALMOBench jointly optimizes for both translation accuracy and execution efficiency. Figure 2 illustrates the benchmark architecture. The contributions are summarized as follows:

- We propose NALMOBench, a new benchmark platform designed to evaluate text-to-EXE systems and LLMs for MODs, featuring over 600 paired natural language and executable database queries that cover both standard operations and real-world scenarios, categorized into 7 query types and stratified across 3 difficulty levels.
- We construct a corpus of approximately 6,000 moving object queries through a hybrid approach that combines manually collected NLQs with LLM-based automated data augmentation, incorporating semantic and formal verification.
- We further optimize the natural language understanding algorithm by constructing NLQ candidates and database candidates, respectively. We propose a divide-and-conquer paradigm to optimize structured language generation.
- We evaluate different NLIDB systems and LLMs on NALMOBench. Experimental results demonstrate that the baseline method achieves 68.96%, 60.07%, and 68.37 in translatability, translation accuracy, and translation efficiency score, respectively, outperforming existing approaches.

2 RELATED WORK

Benchmarks play a pivotal role in the development and validation of NLIDB systems. High-quality benchmarks can promote the vigorous development and continuous progress of NLIDB research. Early research on NLIDB benchmarks often utilize proprietary datasets, which are tailored to specific domains. For instance, parsing based systems like NaLIR, PRECISE [10], and ATHENA++ [11] rely on datasets such as Geoquery, MAS, and YELP, which focus on individual domains and lack generalizability, despite performing well in the respective domains.

With the development of neural network technology, research on NLIDB places higher requirements on system generalization. Although neural network-based technologies are emerging, a test dataset able to be used in a cross-domain context is missing. Thus, WiKiSQL is proposed, which is the first large-scale multi-domain dataset for relational databases. WiKiSQL contains 80,654 NLQs and 77,840 SQL statements. However, the SQL statements in WiKiSQL are relatively simple, lacking complex operations such as sorting, grouping, and subqueries. Therefore, Spider further addresses the limitations of WiKiSQL by constructing a large and complex NLIDB dataset for cross-domain semantic analyses. Spider contains 10,181 NLQs, 5,693 SQL statements, and involves more than 200 databases in 138 different domains. There are common simple SQL queries as well as complex ones such as nested queries. Although WikiSQL and Spider propose cross-domain datasets, the two benchmarks still emphasize on database schema and are insufficient to explore further on database values. KaggleDBQA [5] addresses the above problem by using databases extracted from real-world data sources, which are more relevant to real-world scenario applications.

With the arrival of the era of LLMs, the research of NLIDB has ushered in a new paradigm. Bird is proposed to evaluate LLMs. Bird contains 12,751 text-to-SQL pairs and 95 databases with a total size of 33.4 GB, spanning 37 professional domains. Bird proposes a validation metric to evaluate the efficiency of NLIDB, though primarily focusing on time cost.

While Spider and Bird claim high generalization capabilities, the majority of these databases are specifically created by students, which may not adequately represent real-world database complexities. ScienceBenchmark [17] is the first text-to-SQL benchmark designed with complex real-world scientific databases. Real-world applications are likely to require high performance in a single domain, which may be more important than the ability to generalize across domains.

3 BENCHMARK CONSTRUCTION

Existing benchmarks are predominantly oriented towards text-to-SQL tasks, with no benchmarks specifically constructed for text-to-EXE scenarios. While claimming cross-domain applicability, these benchmarks fail to adequately address domain-specific requirements, such as moving object queries. Moreover, most benchmarks lack validation in real-world scenarios. To address these limitations, we develop NALMOBench, specifically designed for text-to-EXE validation in MODs. Algorithm 1 shows the workflow of the benchmark construction.

Algorithme 1 : Corpus Construction and Annotation

```
Input : the list of $n$ NLQs, $Q$;
        the schema information of the database, $S$;
        moving objects NLQ corpus, $MOC$;
        the list of $n$ executable database queries, $EL$;
Output : validated query pairs, $P$
initialize $Q$ with collected NLQs
for $q_i \in Q$ do
    $el_i \leftarrow$ manual annotation of $q_i$
$Models$ construction according to $Type(q_i)$
for $Model \in Models$ do
    $q \leftarrow Model(S)$
    $Q.append(q)$
for $q_i \in Q$ do
    if $q_i$ fails to meet the semantic, syntax, and entity
      constraints then
        $Q.remove(q_i)$
for $q_i \in Q, el_i \in EL$ do
    $el_i \leftarrow NALMO(q_i)$
    if $z3(el_i) \wedge semantic(el_i) = semantic(q_i)$ then
        $EL.append(el_i)$
$P.append(Q, EL)$
return $P$
```

3.1 Corpus Construction Methodology

We develop a template-based methodology leveraging collected NLQs, augmented by LLMs, to construct a preprocessed corpus of 6,000 NLQs about moving objects, covering time interval query, range query, nearest neighbor query, join query, trajectory similarity query, cross-region query, and detour query.

3.1.1 Natural Language Query Conllection. We collect and design 100 robust moving objects NLQs. For each type of query, except real-world queries, we collect and manually refine 80 NLQs from journals

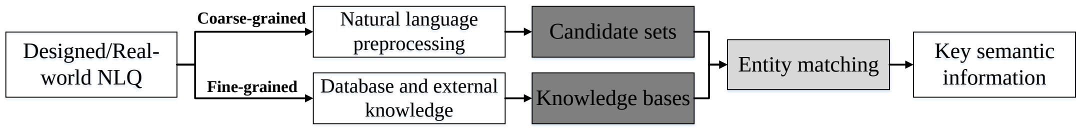
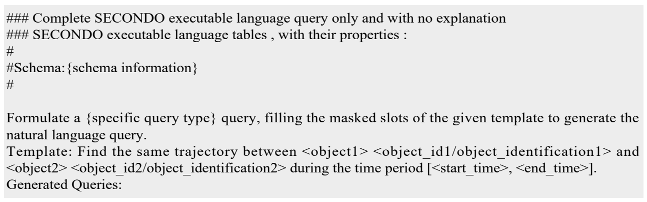


Figure 3: Natural language understanding.

and conferences. We suppose that queries sourced from top-tier database journals and conferences are of high quality. Real-world scenario queries are proposed by traffic enforcement officers and subsequently modified by experts, which reflect actual scenarios and are further fine-tuned by experts to align with the research context. Additionally, we manually annotate the 100 NLQs, and the detailed annotation process will be described later.

3.1.2 Automatic Augmentation. Building upon the collected queries, we develop advanced NLQ augmentation templates for each query type. Limited by space, we show one template of the range query: "*Which <object> pass through <location_type> <specific_location> between <start_time> to <end_time>?*". Such templates serve as the foundation for constructing corresponding prompts using the one-shot methodology. We then leverage GPT-4o to expand the corpus, thereby enriching the expressive diversity of the NLQs. Corresponding prompts are given in Figure 4.


```
### Complete SECONDO executable language query only and with no explanation
### SECONDO executable language tables , with their properties :
#
#Schema:{schema information}
#

Formulate a {specific query type} query, filling the masked slots of the given template to generate the
natural language query.
Template: Find the same trajectory between <object1> <object_id1/object_identification1> and
<object2> <object_id2/object_identification2> during the time period [<start_time>, <end_time>].
Generated Queries:
```

Figure 4: Prompt of natural language generation with natural language templates.

3.1.3 Question Review. To further ensure the quality of the NLQs, we conduct a secondary validation process, using chain-of-thought (CoT) methodology to design prompts for GPT-4o to verify the following aspects of each query: (i) *Whether the syntax is correct.* (ii) *Whether the entity information corresponds to the query type.* (iii) *Whether there is any ambiguity.* Non-compliant queries are reviewed by experts. Since further NLQ extraction will be conducted during the benchmark construction process, the primary focus of the initial validation is to ensure the syntactic correctness and robustness of the query content, without interacting with the database information.

3.2 Benchmark Construction Methodology

We employ a hybrid approach combining manual and automated methods to construct NLQ and executable language pairs. Initially, we manually create 100 robust pairs of moving objects queries. Subsequently, we extract approximately 500 NLQs from an existing corpus using automated validation methods. In addition, we utilize the methods given in NALSpatial [8] for initial annotation, followed by validation of the annotated queries using both syntactic and formal verification techniques. Finally, we rely on experts to label queries that cannot be automatically annotated.

3.2.1 Manual Generation. We engage three graduate students specializing in spatio-temporal databases and two traffic enforcement officers in the manual construction of query pairs. The verification for non-real-world scenarios is conducted by the three graduate students. Queries related to cross-region and detour operations are proposed by the enforcement officers. The graduate students then fine-tune these queries based on the database and construct the corresponding pairs, which are validated by the enforcement officers through executing the executable database queries. In total, 100 query pairs are constructed. Meanwhile, we utilize the constructed corpus to extract NLQs and use GPT-4o to automate the validation process through CoT with organized examples to extract the NLQs.

3.2.2 Annotation. We utilize the method provided by NALSpatial for initial annotation. However, the method can generate only part types of queries. So, we use manually construct pairs as examples and leverage the GPT-4o model to assess the semantic equivalence between generated executable languages and original NLQs. Subsequently, we conduct Z3 formal verification, which involves parsing the executable languages to extract the query types, structure, and components. Symbolic variables are created for entities, and conditions in different clause categories are converted into Z3 constraints to check constraint consistency, specifically to identify and resolve any contradictions within the query constraints. For queries that either can not be generated or fail the verification processes, we enlist the expertise of domain specialists to adjust or reconstruct the executable database queries. In particular, all real-world scenario queries are manually constructed.

4 NATURAL LANGUAGE UNDERSTANDING

The extensive semantics and diverse syntactic variations of natural language impede direct machine comprehension, leading to substantial difficulties in entity recognition and extraction. To address these challenges, we develop a sophisticated two-stage process, comprising coarse-grained and fine-grained natural language processing (NLP) modules, as depicted in Figure 3. In the initial coarse-grained stage, the system generates a candidate set of entities, including time, locations, objects, relations, moving object identifiers, and nearest neighbor numbers. Subsequently, in the fine-grained stage, a pre-constructed knowledge base is leveraged to prune the candidate entity set and extract precise entities.

4.1 Natural Language Candidate Entity

In this phase, our primary objective is the construction of the natural language candidate set. Efficiency is a key consideration, necessitating the rapid identification of entity information. The inherent complexity of semantic structures necessitates the use of advanced NLP tools to approximate and locate potential entities.

Considering the imperative for swift processing, we select spaCy as the primary NLP tool, with NLTK as a potential alternative. Our framework initial implementation predominantly relies on the robust tokenization and named entity recognition capabilities of spaCy. Employing spaCy, we construct two principal sets: (i) *the information set* and (ii) *the number set*. The information set encompasses candidate locations, objects, and relations. The number set comprises the number of nearest neighbors, moving object identifiers, and time information.

Given that the number set encapsulates numerical and temporal information, we can precisely locate time and some of the numbers. For instance, when encountering a time expression containing a colon, such as "*8:00 o'clock*", we utilize a regular expression to achieve accurate matching and extraction. For hour expressions lacking a colon, such as "*8am*", we also employ a regular expression for matching and extraction. The number of nearest neighbors can then be determined according to the semantic distance of the keywords like "*nearest*". Due to the potential ambiguity of the query object, object identification necessitates further validation.

4.2 Database Candidate Entity

Aligning natural language entity information with deterministic database information presents significant challenges due to inherent ambiguities and uncertainties in natural language. To address this, we propose the construction of database candidates for precise entity extraction. To achieve accurate extraction of location, objects, relations, and object identifiers, we propose a sophisticated moving objects knowledge base (MOKB). MOKB is composed of two sub-knowledge bases: (i) *the relation knowledge base (RKB)* and (ii) *the location knowledge base (LKB)*.

We conduct a thorough analysis of the objects within the database to identify relations containing the $\underline{\textit{mpoint}}$ attribute, which is utilized to represent moving objects. This relation encapsulates the target moving objects. The LKB primarily extracts objects with attributes such as *point*, *line*, and *region* from the database. Recognizing the complexities posed by synonyms, abbreviations, and ambiguous terms for locations, we leverage the capabilities of GPT-4o to augment the existing LKB with potential synonyms, abbreviations, and ambiguous expressions. The extended terms are mapped to the original locations solely for entity matching and will not be manifested as final location entities. The RKB is employed to determine moving objects relations and objects. Once an object is identified, the nearest numerical value in semantic distance from the object, existing within the number set and not previously identified as another entity, is ascertained as the object identifier. The location information can then be retrieved from the LKB.

When managing large-scale databases, the knowledge base can expand significantly, leading to prohibitive time consumption during retrieval processes. To mitigate this, we devise a hierarchical index for MOKB to expedite retrieval operations.

5 STRUCTURED LANGUAGE GENERATION

The inherent complexity of moving object queries renders the development of a universal structured language model (SLM) impractical and costly, especially in real-world scenarios. Therefore, we propose a divide-and-conquer strategy to reduce model design costs and develop a query optimization model that refines generated structured languages through index selection, with the detailed construction process illustrated in Figure 5.

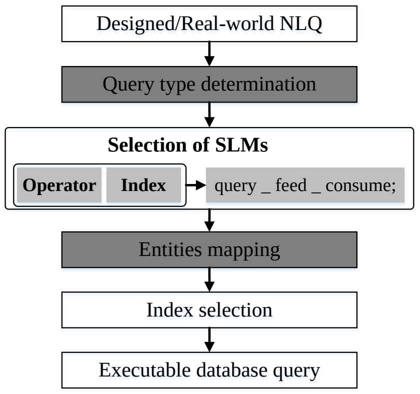


Figure 5: Structured language construction.

5.1 Structured Language Model Sketch and Query Type Classification

Figure 5 delineates the sketch SLM, where $\langle k \rangle$, $\langle t \rangle$, $\langle o_i \rangle$, $\langle place \rangle$, and $\langle relation \rangle$ denote the undetermined number of nearest neighbors, time, object and identifier, location, and moving objects relation, respectively. In scenarios where a location is encapsulated within a moving objects relation, the $\langle place \rangle$ placeholder in the model is substituted with the following expression. Here, $\langle tmp\ relation \rangle$ and $\langle name \rangle$ represent the moving objects relation and the name of the object: "$\langle tmp\ relation \rangle$ *feed filter* [*.Name* = $\langle name \rangle$] *extract* [*Trip*]". Table 1 enumerates part of the key operators, encompassing names, signatures, and functionalities.

Table 1: Operators in SECONDO.

Operator	Signature	Meaning
filter	stream × filter condition → stream	Filter the elements of the stream by a predicate.
trajectory	$\underline{\text{mpoint}} \rightarrow$ line	Map continuous moving points into trajectories.
deftime	$\underline{\text{mpoint}} \rightarrow$ periods	Project motion onto the time dimension.
at	$\underline{\text{mpoint}} \times \{\text{point, region}\} \rightarrow \underline{\text{mpoint}}$	Return moving points to a certain point or region.
knearest	stream × $\underline{\text{mpoint}}$ × int k → stream	Compute the k nearest neighbors.
dist_euclidean	pointseq × pointseq → real	Compute the trajectory similarity.

For the type classification model training, the corpus is partitioned into a training set and a test set with a ratio of 8:2. LSTM is employed for training and the trained model can be utilized to determine the query type of the input NLQ.

5.2 Structured Language Model Selection and Slot Filling

For different query types, we map the corresponding entities to respective slots using predefined mapping rules and combine them with the required operators to generate the structured languages. SLMs for time interval queries, range queries, and nearest neighbor queries can be constructed based on methodologies described in NALMO [13]. In this paper, we focus on the similarity threshold, usage of indexes, and construction of structured languages in real-world scenarios. Limited by space, we just take the cross-region query as an example. Detailed description of SLMs are given in NALSpatial and NALMO.

To determine whether a taxi trajectory intersects with the operational boundary and within the operational area, we use the *intersects* operator. If there is an intersection or the trajectory is outside the operational area, we deduce the existence of a cross-region operation. However, if the taxi exits the operational boundary and then re-enters within a short period, this behavior will not be considered as a cross-region operation.

6 EXPERIMENTAL RESULTS AND DISCUSSION

We report extensive experimental results in this section. The implementation is developed in a computer (Intel(R) Core(TM) i9-10850K CPU, 3.60 GHz, 32 GB memory, 512 GB disk) running Ubuntu 20.04 (64 bits, kernel version 5.14.0-1051-oem). We aim to investigate the performance of existing text-to-SQL methods and LLMs in addressing the text-to-EXE problem, with a focus on their applicability and robustness in the domain of MODs. To ensure a comprehensive evaluation, we not only assess the translation accuracy but also the quality of the generated executable languages, including translation efficiency and execution efficiency. We provide a thorough understanding of the strengths and weaknesses of different approaches, offering valuable insights for real-world applications.

6.1 Experimental Setup

Setting. As described in Section 3, a combined approach of manual and LLM-based annotation is employed. Although leveraging LLMs for generation can reduce workload, a certain degree of manual review remains necessary. Consequently, 600 pairs of NLQs and structured query pairs are ultimately annotated. Example pairs [1] are available. Existing queries primarily fall into two categories: (i) *those specifically designed for databases* and (ii) *those tailored to real-world requirements*, which both are considered. The specialized queries include time interval queries (TI), range queries (RA), nearest neighbor queries (NN), join queries (J), and trajectory similarity queries (TS), while real-world scenario queries encompass cross-region queries (CR) and detour queries (DT). According to the query length, SQL query difficulty is categorized into four levels, including simple, medium, hard, and extra hard in Spider. Building upon this, we further classify moving object queries into three complexity levels, including simple, medium, and hard, by incorporating their intricate query types, shown in Table 3.

Datasets. The experiment utilize datasets comprising both real-world taxis, cars, and buses movement trajectories and system-simulated train movement trajectories, as shown in Table 2, ensuring the comprehensiveness and accuracy of our experimental results. The real-world trajectory dataset *nanjingtest* spans the period from 00:00 to 24:00 on June 15, 2024, capturing movement fragments from 302 moving objects, containing a total of 917,006 moving object points. Meanwhile, the system-simulated train trajectory dataset *berlintest* covers the period from 06:00 to 09:00 on November 20, 2020, involving trajectory fragments from 562 trains and consisting of 51,544 moving object points.

Table 2: Details of the datasets used in the benchmark.

Dataset	#Points	#Lines	#Regions	#Moving objects
nanjingtest	9000	887	13	302
berlintest	3040	4078	330	562

6.2 Evaluation Metrics

In the domain of text-to-SQL, the evaluation of system performance mainly hinges upon three critical metrics: (i) *Exact Match (EM)*, (ii) *Execution Accuracy (EX)*, and (iii) *Valid Efficiency Score (VES)* [7]. Our evaluation metrics are inspired by the three metrics above.

To evaluate the natural language understanding capabilities of the text-to-EXE systems, we propose the metric of Translatability (TA). Given the set of executable database queries generated by the system and the set of input NLQs, denoted by *EQ* and *N*, respectively, TA is defined as:

$$TA = \frac{|EQ|}{|N|} \tag{1}$$

This metric is used to assess the semantic completeness of executable languages, which are both logically correct and entity-complete. TA broadens the scope by acknowledging all structurally viable database queries, provided executable queries incorporate the requisite entities and maintain logical integrity.

Furthermore, to evaluate the proficiency in formulating structured languages, we propose the Translation Precision (TP) metric. Defined over the set of executable database queries that align with expected results and the input NLQs *N*, TP assesses whether executable languages deliver the expected outputs post-execution. We compare the results between the generated executable database query and the golden one, assuming the same execution results indicate equivalence. Let S_n be the executable database query set for the n-th example, G_n be the golden database query set for the n-th example, $R(S_n)$ be the result set after executing S_n, $R(G_n)$ be the result set after executing G_n, TP can be defined as:

$$TP = \frac{1}{N}\sum_{n=1}^{N}\left(\mathbb{1}(S_n, G_n) \cdot \mathbb{1}(R(S_n), R(G_n))\right) \tag{2}$$

where $\mathbb{1}(A, B)$ is an indicator function defined as:

$$\mathbb{1}(A, B) = \begin{cases} 1, & \text{if } A = B \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

To comprehensively assess the quality of the generated executable languages, our analysis extends to encompass efficiency metrics, called Translation Efficiency Score (TES), including execution time and a composite measure incorporating CPU time during execution. TES goes beyond VES by considering execution time and CPU time, thus providing a more comprehensive assessment dimension. We only consider statements that are capable of producing the same results as the given executable languages. Let $E(\cdot)$ represent the measurement of execution time and $C(\cdot)$ represent the

[1]https://pan.nuaa.edu.cn/share/c9bd35cbe21c175137d5de4db9.

Table 3: Results of TA and TP.

Method	TA/%								TP/%							
	Easy	Medium				Hard		All	Easy	Medium				Hard		All
	TI	RA	NN	J	TS	CR	DT		TI	RA	NN	J	TS	CR	DT	
NaLIR	5.84	3.06	1.79	1.05	2.00	0.00	0.00	2.52	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ATHENA++	37.27	31.61	21.43	27.37	28.00	10.17	23.53	29.87	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ValueNet	35.04	20.41	19.64	18.00	16.95	11.86	15.69	24.33	16.79	14.29	10.71	5.26	2.00	3.39	5.88	9.23
LLaMA3-70B (few-shot)	51.10	33.16	30.17	22.11	16.50	10.17	31.37	30.20	23.36	19.47	20.54	10.00	5.50	5.93	6.86	14.18
GPT-4o (few-shot)	68.98	44.75	45.54	30.53	35.50	11.87	37.26	42.87	47.45	34.19	35.72	20.53	9.50	7.63	18.65	27.01
NALMO+	89.05	79.78	78.57	73.68	77.00	45.76	58.82	**68.96**	72.99	61.22	57.14	61.05	65.00	37.29	41.18	**60.07**

TI, Time Interval Query; RA, Range Query; NN, Nearest Neighbor Query; J, Join Query; TS, Trajectory Similarity Query; CR, Cross-Region; DT, Detour.

measurement of CPU time. The two functions are used to measure the absolute time in a given environment. We can design a formula that combines these two functions:

$$TES = \frac{\sum_{n=1}^{N} \left(\mathbb{1}(R(S_n), R(G_n)) \cdot \mathrm{Cost}(Y_n, \hat{Y}_n) \right)}{N} \quad (4)$$

$$Cost(Y_n, \hat{Y}_n) = \frac{\sqrt{E(Y_n)}}{\sqrt{E(\hat{Y}_n)}} \cdot \frac{\sqrt{C(Y_n)}}{\sqrt{C(\hat{Y}_n)}} \quad (5)$$

where $Cost(\cdot)$ denotes the relative efficiency of the generated executable languages compared to the standard executable languages. We comprehensively consider the execution time and CPU time ratio of the standard executable languages versus the generated executable languages to ensure a thorough evaluation.

6.3 Baseline Methods

We conduct a comparative analysis of six systems. Given the rapid advancements in text-to-SQL, we select representative methods from three popular categories: (i) *Rule-based methods.* We employ NaLIR and ATHENA++. NaLIR is a classical rule-based approach that utilizes parse trees for entity extraction and alignment, and constructs SQL with user feedback. ATHENA++ utilizes OQL queries as an intermediate representation through a two-phase generation method. (ii) *Neural network-based methods.* We choose ValueNet [1], which builds upon IRNet by extending the SemQL grammar to include values in the generated queries. (iii) *LLM-based methods.* We leverage GPT-4o and the open-source model LLaMA3 as comparative methods. In the BIRD leaderboard, over half of the top ten methods are based on GPT-4o. LLaMA3 is considered, which is one of the most widely used open-source models. We design experiments using the methodology of DAIL-SQL [2], which integrates schema information into prompt design and example organization, utilizing few-shot methods. Additionally, we introduce the optimized method called NALMO+ for comparison. NALMO+ is a hybrid architecture integrated into the SECONDO system as an algebra module, and the corresponding operator is also developed.

6.4 Experimental Results

We consider the query translation capabilities. Table 3 presents TA and TP of all methods, providing detailed data for different levels of difficulty and query types. Since the executable languages in the MOD differ in both syntax structure and semantic representation from standard SQL, rule-based methods focus solely on the natural language understanding process.

NaLIR constructs a semantic parse tree and then maps the nodes onto SQL fragments. This database preset approach yields high error rates in entity recognition and operator selection. Consequently, NaLIR acquires low TA, and TP cannot be directly measured. ATHENA++ combines a domain ontology with Stanford CoreNLP for advanced NLP in simple queries. simple queries involving few entities. However, ATHENA++ fails to extract or map fuzzy entities, such as synonyms, abbreviations, or ambiguous terms. In real-world scenario queries, where natural language often provides minimal entity information and multi-step decomposition is required, ATHENA++ breaks down entirely.

ValueNet emits an intermediate representation (IR), making the post-processing step that converts IR into structured SQL critical. By ingesting actual database contents, ValueNet matches the TA of CoreNLP-based methods. Yet, because ValueNet depends on large-scale supervised training, the ability to generate executable languages remains limited. TP stays below 20% across all categories, and declines further as query complexity increases.

To analyze the performance of LLMs, we design few-shot prompting experiments. Due to the syntactic and semantic gap between moving objects executable languages and standard SQL, LLMs without example prompts fail all query tasks. GPT-4o outperforms LLaMA3 in both TA and TP, showing over 50% and 100% relative improvements, respectively. However, the rich semantic patterns and lack of fixed operator entity templates make single-shot prompts insufficient for unseen queries, while the generated logic may be correct, the omission of an initial temporal filter step leads to slower execution than the golden executable database query. Increasing to five examples yields over 20% gains in both TA and TP for LLaMA3, and over 40% gains for GPT-4o. However, further increasing sample size may risk hallucinations, including operator misuse and the invention of non-existent operators.

For TES, both the small model of ValueNet and LLMs exhibit suboptimal performance. ValueNet, LLaMa3, and GPT-4o achieve 12.43, 26.37, and 60.71, respectively. Due to the inherent white-box nature of executable languages, query efficiency can be directly controlled. Small models are tightly bound to the training datasets, and LLMs have not been exposed to enough training data directly related to database indexing, especially in terms of optimizing queries with indexes. This requires substantial re-training or fine-tuning with large datasets. In contrast, the hybrid approach NALMO+ demonstrates a notable advantage, maintaining relatively efficient queries, achieving 68.37 of TES even in the absence of an extensive dataset.

In conclusion, these findings underscore the challenges posed by the NALMOBench benchmark, further reinforcing that text-to-EXE research is still in its early stages, with considerable potential for further exploration and advancement in the field.

6.5 Discussion of the Experiments

Query complexity is fundamentally governed by the intrinsic difficulty of the query. For simple TI queries, which only incorporate temporal constraints, target objects, and relation information. The structured languages are relatively rigid and utilize exclusively temporal operators. Consequently, both the natural language understanding phase and the structured language generation phase remain straightforward, enabling all evaluated methods to achieve highest scores in TA and TP. By contrast, moderate complexity queries, including RA, NN, J, and TS queries. entail a far richer set of entity attributes (e.g., time, locations, relations, object identifiers, and nearest neighbor numbers). This enriched semantic scope significantly heightens the difficulty of natural language parsing. Concurrently, the structured language grammar must cover an expanded operator vocabulary, causing a combinatorial explosion in operator and entity pairings. LLMs exhibit three principal shortcomings in this situation: (i) *Entity mapping errors.* LLMs struggle to perform precise date transformations and to resolve fuzzy spatial references, resulting in misidentified temporal or locational entities. (ii) *Operator selection failures.* When confronted with unseen operators, LLMs either omit necessary operators or fabricate non-existent ones, thereby rendering the generated queries syntactically invalid. (iii) *Module omissions.* LLMs often neglect critical sub-modules required for complete structured language generation, culminating in queries that cannot be executed.

Consider the TS query as a concrete example. The similarity threshold must be empirically calibrated on each dataset, demanding an operation that depends on dataset-specific experiments and manual adjustment. However, LLMs cannot infer optimal threshold values and frequently generate spurious operators or default to seldom used but valid ones, which degrades query performance. At the hard level, CR and DT queries typically involve multiple sequential processing steps or nested query constructs. Because real-world execution requires iterative query handling, existing approaches falter, with both TA and TP metrics remain below 50%.

7 CONCLUSION

In this paper, we introduce NALMOBench, a novel benchmark designed to evaluate both the capability and quality of translating NLQs into executable languages for MODs. NALMOBench covers not only synthetic moving objects datasets but also real-world trajectory data. We further present an LLM-based approach for generating a domain-specific database corpus. This method encompasses both routine moving objects queries and those arising from intricate real-world scenarios. To tackle the text-to-EXE problem, we decompose the problem into two sub-problems: (i) *natural language understanding* and (ii) *structured language generation*, and provide baseline algorithms for each stage.

Our experiments demonstrate that NALMOBench poses significant challenges to existing text-to-EXE methods and LLMs. We contend that NALMOBench establishes a rigorous new standard for evaluating text-to-EXE systems in the MOD field. NALMOBench lays the groundwork for addressing the inherent complexities of the usage of real-world trajectory data and for advancing research into optimizing the efficiency of post-translated executable languages.

8 ACKNOWLEDGMENT

This work was supported by Postgraduate Research & Practice Innovation Program of Jiangsu Province (KYCX24_0608) and Interdisciplinary Innovation Fund for Doctoral Students of Nanjing University of Aeronautics and Astronautics (KXKCXJJ202505).

REFERENCES

[1] Ursin Brunner and Kurt Stockinger. 2021. ValueNet: A Natural Language-to-SQL System that Learns from Database Information. In *ICDE.* 2177–2182.
[2] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proc. VLDB Endow.* 17, 5 (2024), 1132–1145.
[3] 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.
[4] 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.
[5] Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. KaggleDBQA: Realistic Evaluation of Text-to-SQL Parsers. In *ACL/IJCNLP.* 2261–2273.
[6] Fei Li and H. V. Jagadish. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. *Proc. VLDB Endow.* 8, 1 (2014), 73–84.
[7] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. In *NeurIPS.*
[8] Mengyi Liu, Xieyang Wang, Jianqiu Xu, Hua Lu, and Yongxin Tong. 2025. NALSpatial: A Natural Language Interface for Spatial Databases. *IEEE Trans. Knowl. Data Eng.* 37, 4 (2025), 2056–2070.
[9] Yi Liu, Xiangyu Liu, Xiangrong Zhu, and Wei Hu. 2024. Multi-Aspect Controllable Text Generation with Disentangled Counterfactual Augmentation. In *ACL.* 9231–9253.
[10] Ana-Maria Popescu, Oren Etzioni, and Henry A. Kautz. 2003. Towards a theory of natural language interfaces to databases. In *IUI.* 149–157.
[11] Jaydeep Sen, Chuan Lei, Abdul Quamar, and et al. 2020. ATHENA++: Natural Language Querying for Complex Nested SQL Queries. *Proc. VLDB Endow.* 13, 11 (2020), 2747–2759.
[12] Yongxin Tong, Jieying She, Bolin Ding, Libin Wang, and Lei Chen. 2016. Online mobile Micro-Task Allocation in spatial crowdsourcing. In *ICDE.* 49–60.
[13] Xieyang Wang, Mengyi Liu, Jianqiu Xu, and Hua Lu. 2023. NALMO: Transforming Queries in Natural Language for Moving Objects Databases. *GeoInformatica* 27, 3 (2023), 427–460.
[14] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *EMNLP.* 3911–3921.
[15] John M. Zelle and Raymond J. Mooney. 1996. Learning to Parse Database Queries Using Inductive Logic Programming. In *AAAI.* 1050–1055.
[16] Chao Zhang, Yuren Mao, Yijiang Fan, Yu Mi, Yunjun Gao, Lu Chen, Dongfang Lou, and Jinshu Lin. 2024. FinSQL: Model-Agnostic LLMs-based Text-to-SQL Framework for Financial Analysis. In *SIGMOD/PODS.* 93–105.
[17] Yi Zhang, Jan Deriu, George Katsogiannis-Meimarakis, Catherine Kosten, Georgia Koutrika, and Kurt Stockinger. 2023. ScienceBenchmark: A Complex Real-World Benchmark for Evaluating Natural Language to SQL Systems. *Proc. VLDB Endow.* 17, 4 (2023), 685–698.
[18] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *CoRR* abs/1709.00103 (2017).