NALSpatial: A Natural Language Interface for Spatial Databases

Mengyi Liu, Xieyang Wang, Jianqiu Xu, Hua Lu, *Senior Member, IEEE*, and Yongxin Tong, *Member, IEEE*

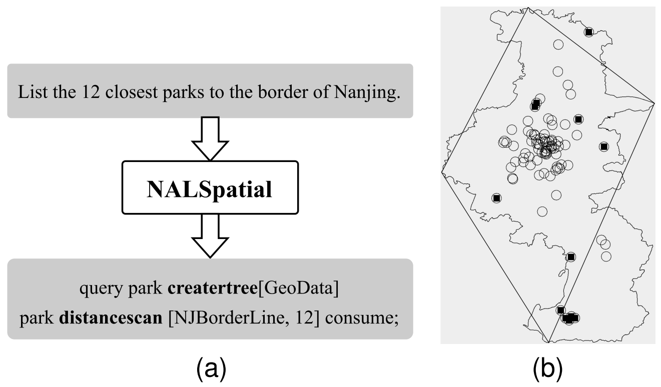
***Abstract*—Spatial databases play a vital role in a number of applications ranging from geographic information systems to location-based services. Application tasks typically access underlying spatial data to answer queries. However, non-experts lack the expertise necessary for formulating spatial queries. To fill in this gap, we propose an effective framework that translates natural language queries over spatial data into executable database queries, called NALSpatial. The framework consists of two core phases: (i) *natural language understanding* and (ii) *natural language translation*. Phase (i) extracts key entity information, comprehends the query intent and determines the query type by employing natural language processing techniques and deep learning algorithms. The key entities and query type are passed to phase (ii), which makes use of entity mapping rules and structured language models to construct executable database queries. NALSpatial supports dealing with five types of queries including (i) *basic queries (e.g. distance and area)*, (ii) *range queries*, (iii) *nearest neighbor queries*, (iv) *spatial join queries* and (v) *aggregation queries*. We develop NALSpatial in an open-source extensible database system SECONDO. Extensive experiments show that NALSpatial on average achieves response time of about 2.5 seconds, translatability of 95% and translation precision of 92%, outperforming three state-of-the-art methods.**

***Index Terms*—Spatial Database, Natural Language Interface, Semantic Parsing, Query Processing.**

I. Introduction

SPATIAL databases manage massive spatial data to support a wide range of applications such as geographic information systems, location-based services and urban planning [1]. Emerging application tasks require an increasing number of users to derive insights from the data as quickly as possible. Traditionally, users send their queries to the database system to retrieve the underlying data, but formulating complex structure query languages is not a trivial task for non-technical users. To solve the issue, the natural language interface for database (NLIDB) enables such non-technical users to explore the data in a convenient way without relying on experts' help [2]. Although a great deal of research has been conducted on spatial databases such as *data partitioning* [3], [4], *indexing structures* [5]–[8], *keyword queries* [9]–[11], and *spatial crowdsourcing* [12]–[14], little effort has been devoted to natural language interfaces to spatial databases. To bridge this severe gap, we build a natural language interface for non-experts to friendly interact with spatial databases.

In the literature, NLIDBs have been mainly studied in relational databases (e.g. PRECISE [15]), XML databases (e.g. DaNaLIX [16]), RDF Q/A (e.g. TEQUILA [17]) and crowd mining (e.g. NL2CM [18]). Such interfaces cannot be directly applied to the generation of executable languages in the spatial domain because of the complexity of spatial queries (e.g., advanced tasks involving nearest neighbor search and spatial aggregation). There are two major challenges in building natural language interfaces for spatial databases: (i) Semantic understanding is an intractable problem due to ambiguity in natural language. If the user intent is not accurately understood, incorrect results will be produced. (ii) Structured language construction. The executable sentence follows specific system-level rules. One application task could be formatted in several ways and how to produce the optimal query plan including appropriate operators, data structures and the execution order is an open issue.


List the 12 closest parks to the border of Nanjing.

```
query park createrree[GeoData]
park distancescan [NJBorderLine, 12] consume;
```

Fig. 1. Transforming NLQs over spatial data into executable database queries. (a) Processing NLQ_1. (b) The results of NLQ_1.

Regarding the issue of semantic understanding, former NLIDBs leverage natural language processing (NLP) tools, including *NLTK* [19], *spaCy*[1], *Stanford CoreNLP* [20] and *Stanza* [21]. Nevertheless, such tools have difficulty in accurately parsing spatial queries due to the intricate geometric relationships and diverse query types inherent to spatial data. Recently, GPT-4, a prominently discussed large language model (LLM), excels in parsing the semantics of natural language queries (NLQs) over spatial data, spawning a new paradigm of NLIDB. However, one of the primary constraints is the insufficiency of datasets tailored for executable languages, which necessitates prompt engineering or fine-tuning using newly developed corpus. Reliance on prompts to consistently incorporate operator is not advisable, as this approach introduces a trade-off in accuracy [22]. Additionally, the design and creation of specialized corpus for fine-tuning are often cost-prohibitive. **R2.**C1 **R3.**C3

M. Liu, X. Wang, and J. Xu are with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China. E-mail: {liumengyi, xieyang, jianqiu}@nuaa.edu.cn.

H. Lu is with the Department of People and Technology, Roskilde University, 4000 Roskilde, Denmark. E-mail: luhua@ruc.dk.

Y. Tong is with the State Key Laboratory of Software Development Environment, School of Computer Science and Engineering, Beihang University, Beijing 100190, China. E-mail: yxtong@buaa.edu.cn.

(Corresponding author: Jianqiu Xu.)

[1]https://spacy.io/

Taking NLQ_1 as an example, several steps have to be performed to process the natural language query including understanding the meaning "*the border of Nanjing*", retrieving the corresponding data, utilizing appropriate operators (e.g., *boundary*, *nn*), and formatting the executable language. We report the actual border of Nanjing, all parks and also mark the result (solid square) in Figure 1. However, ChatGPT produces a diamond shape to represent the border of Nanjing but this does not accurately transform the query.

EXAMPLE 1. NLQ_1*: "List the 12 closest parks to the border of Nanjing."*

Regarding the issue of structured language construction, SQL may encounter difficulties when handling complicated spatial queries with multidimensional data. Existing NLIDBs rely on optimizers to generate final query results after translating natural language to SQL [23]. The reliance can reduce control over data and performance, which are crucial aspects in spatial databases. We explore the possibility of transforming natural languages into executable languages that can handle different operations and optimize queries, such as operator selection and combination. Explicit textual plans help understanding what the optimizer does. Although the executable level is complex to utilize, the user has full control over the steps of manipulating data. The full power of the kernel system is available, including the latest incorporated types or index structures. Taking NLQ_1 as an example, SQL utilizes the keywords *ORDER BY* and *LIMIT* and the operator *ST_Distance* to compute the 12 nearest parks, while the executable language provides affluent spatial semantics directly using the operator *distancescan*. **R2**.C1 **R3**.C3

We propose a natural language interface for spatial database, named NALSpatial, which works in two phases. Phase (i) focuses on natural language understanding. We coarsely extract entities to generate a candidate entity set by NLP tools. The candidate entity set is then pruned using pre-constructed knowledge bases and the entity information extraction algorithm to determine the precise entities, including the number of nearest neighbors, distance threshold, spatial relations, and locations. Moreover, we train the pre-built corpus to identify query types. The corpus is constructed utilizing ChatGPT and exclusively includes spatial NLQs, which simplifies construction and ensures quality compared to a corpus containing pairs of NLQs and executable database queries for spatial data. Our semantic understanding approach effectively addresses the issues of extensive training data and external knowledge supplementation associated with schema and value linking. Phase (ii) is dedicated to natural language translation. The structured language model is selected by the query type, and then key entity information is filled into the model following the mapping rules, which ultimately generate the executable language. To reduce the complexity of constructing structured language models, mapping rules and templates for structured languages are built directly from query types, thus avoiding the need for enumeration. **R2**.C1 **R2**.C3 **R3**.C3 NALSpatial supports five kinds of spatial queries including (i) *basic queries*, (ii) *range queries*, (iii) *nearest neighbor queries*, (iv) *spatial join queries* and (v) *aggregation queries*. Our proposal is developed in an open-source extensible database system SECONDO [24] but not confined to a particular system. The framework can be adapted for other spatial databases such as PostGIS[2], as long as syntax rules for databases are followed and operators in the structured language models are replaced with operators of counterpart functionality.

Our main contributions are summarized as follows:

- We propose a framework to address the issue of transforming natural language queries over spatial data into executable database queries.
- To parse natural language, we present an algorithm that extracts key entities, and construct a spatial natural language query corpus for identifying query types.
- We design structured language models for five kinds of queries including basic queries (distance, direction, length and area), range queries, nearest neighbor queries, spatial join queries (join conditions involve location and distance) and aggregation queries (aggregation functions include count, sum and max).
- We develop NALSpatial in a prototype database system and conduct a comprehensive experimental study using real datasets. The results demonstrate the advantage of our framework under various settings.

This paper is an extension to the work presented in [25]. There are four major differences from the original version: (i) the studied problem is comprehensively defined and the related work is extensively discussed; (ii) the natural language understanding module involves fundamental spatial queries and the procedure of query type identification is introduced; (iii) the supported queries and structured language models are formalized in natural language translation; (iv) the experimental evaluation is extended by conducting comparative evaluation and performance verification, and the generality is demonstrated by comparing NALSpatial with GPT-4o.

The rest of the paper is organized as follows. We review the related work in Section II and provide an overview of the framework in Section III. Natural language understanding and natural language translation are introduced in Section IV and Section V, respectively. The experimental evaluation is reported in Section VI. Section VII concludes the paper.

II. Related Work

We briefly review (i) *spatial databases* and (ii) *natural language interfaces for databases*.

A. Spatial databases

The growth of mobile computing devices and the continuous evolution of positioning technologies have significantly augmented the production of spatial data [26]. Therefore, there is a growing demand for efficient processing of spatial data. In response to this, several prototype systems have been developed to manage spatial data, such as GeoSpark [27], Ganos [28] and SECONDO [24]. GeoSpark is a cluster computing system designed for visualizing large-scale geospatial data. Ganos offers comprehensive support for cloud-native databases, specifically tailored for processing moving objects

[2]https://postgis.net/

and 3D scene data. SECONDO is an extensible database system designed for conducting research on spatial data and moving objects, which offers users a wide range of basic and specialized operators, with the flexibility of integrating custom implementations. In the context of spatial data owned by multiple parties, data federation is an effective way to securely query and analyze distributed data [29]. Existing works on spatial databases primarily including the following four parts but little effort has been devoted to NLQ.

(i) *Spatial data partitioning.* A recent study by *Shehab et al.* [3] involves an enhanced partitioning algorithm for SpatialHadoop. To select an appropriate partitioning method for spatial data, a significant analysis in the context of reinforcement learning is presented [4].

(ii) *Spatial indexing.* *Cong et al.* [30] propose a location-aware indexing framework for top-k text retrieval leveraging the inverted file and R-tree. In 2018, *Kraska et al.* [31] first introduced the concept of learned indexing and developed recursive model index. LISA [32] is a learned index for spatial data that employs machine learning models to generate the data layout and thus adapt to diverse datasets. R1.C3

(iii) *Spatial keyword queries.* A qualitative study by *Ahmed et al.* describes how to find the spatial regions where a given keyword is in the top k most frequent keywords [10]. *Luo et al.* apply instant spatial keyword queries to the road networks by addressing typographical errors in keywords [11].

(iv) *Spatial crowdsourcing.* The core issue of spatial crowdsourcing is how to efficiently assign tasks to workers [33], [34]. A visual analysis system is built to present real-time task assignment and help users analyze the process of task assignment [12].

B. Natural language interfaces for databases

The Transformer architecture has led to notable success of LLMs in NLP tasks [35]. The models effectively capture the deep structure and semantic information of language by pre-training and fine-tuning [36]. Decoder-only, encoder-only and encoder-decoder are the principal structures of LLMs.

(i) *The decoder-only model*, represented by GPT [37], [38], exclusively comprises a decoder and generates output sequences progressively through an autoregressive approach. The model is suitable for generative tasks such as text generation and dialogue systems [39]. However, due to the autoregressive nature, the model exhibits limited effectiveness when processing long texts.

(ii) *The encoder-only model*, represented by BERT [40], contains only an encoder and extracts context through bidirectional training. This architecture is applicable to tasks involving context comprehension and supervised learning. Lacking a direct output generation mechanism, the model is unsuitable for generative tasks. In addition, the model cannot handle variable-length outputs in sequence-to-sequence tasks.

(iii) *The encoder-decoder model*, represented by T5 [41], consists of an encoder and a decoder. The encoder maps the input sequence to a high-dimensional contextual representation, and the decoder uses this representation to generate the output sequence. The architecture excels in tasks requiring global information transfer, such as machine translation and summary generation [42]. However, the computational resource demands of the model are high.

TABLE I
TRANSFORMATION METHODS FOR NATURAL LANGUAGE QUERIES

Approach	Typical methods	Domain	Target language
Rule-based	PRECISE [15]	Relational	SQL
	NaLIR [43], [44]	Relational	SQL
	ATHENA [45]	Relational	SQL
	NL2CM [18]	Crowd mining	OASSIS-QL
	NALMO [46]	Moving Objects	Executable language of SECONDO
Machine learning-based	SpatialNLI [47]	Spatial	Lambda expression
	IRNet [48]	Relational	SQL
	ValueNet [49]	Relational	SQL
	Unnamed method [50]	Relational	SQL
	Unnamed method [51]	RDF Q/A	SPARQL

The improvement of NLP contributes to the development of NLIDB. Notably, the growing popularity of ChatGPT opens new possibilities for NLP in NLIDB. ChatGPT supports NLQ over spatial data and provides a reasonable SQL framework.

EXAMPLE 2. NLQ_2*: "Can you tell me what POIs are available in Jiangning District?"*

ChatGPT transforms NLQ_2 into the following structured language:

SELECT *POI.name*
FROM *POI*
JOIN *district*
ON *ST_Within(POI.geom, district.geom)*
WHERE *district.name = 'Jiangning District';*

The SQL employs the *ST_Within* function to check whether each *POI* is within *Jiangning District*. ChatGPT extracts the entity information, including *POI* and *district*, along with identifying the query type as a spatial join query.

However, ChatGPT is mainly oriented to traditional relational databases and has limited ability to represent spatial data. While ChatGPT can adeptly process straightforward objects like points, its representation capability falters when dealing with more intricate objects such as lines and regions (e.g. "*the border of Nanjing*" in NLQ_1).

Several NLIDBs have been developed to tackle the significant challenges of semantic parsing and structured language generation [2]. To address these issues, researchers have proposed two primary approaches: (i) *rule-based* and (ii) *machine learning-based.* The typical methods for each approach are shown in Table I.

The rule-based approach for NLIDB requires explicit rules to facilitate the translation of natural language queries into structured language queries. For example, PRECISE [15] defines the concept of semantic ease of processing, which is utilized to identify a subset of natural language queries that can be precisely translated into SQL. To handle more complex natural language queries, PRECISE leverages user interaction to facilitate semantic understanding. Moreover, NL2CM [18] decomposes natural language queries into generic and inde-

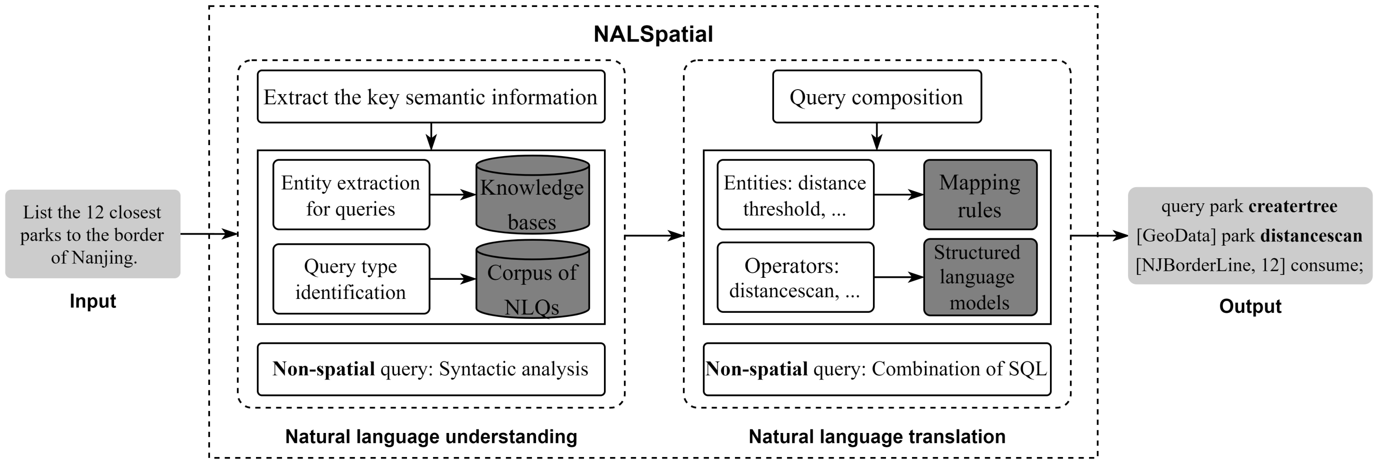

NALSpatial
List the 12 closest parks to the border of Nanjing.
Input
Extract the key semantic information
Entity extraction for queries
Knowledge bases
Query type identification
Corpus of NLQs
Non-spatial query: Syntactic analysis
Natural language understanding
Query composition
Entities: distance threshold, ...
Mapping rules
Operators: distancescan, ...
Structured language models
Non-spatial query: Combination of SQL
Natural language translation
query park creatertree [GeoData] park distancescan [NJBorderLine, 12] consume;
Output

Fig. 2. The architecture of NALSpatial.

pendent parts using an individual expression detector. The two parts are then processed separately and recombined to generate the final structured languages. While the rule-based approach can yield superior results in certain domains, the semantic representation of natural language queries raised by users may be limited [46].

The machine learning approach can be used to directly learn the mapping from natural language to the semantic representation, eliminating the need for an intermediate representation like a parse tree [49]. SpatialNLI [47] leverages an external spatial understanding model to extract spatial entities, followed by a natural language transformation model seq2seq that learns the semantic structure. IRNet [48] identifies the columns and tables mentioned in the NLQ and assigns different types to the columns based on how the columns are mentioned. A grammar-based neural model is then employed to synthesize an intermediate representation that connects natural language and SQL. Finally, IRNet infers SQL from the intermediate representation. Nevertheless, the performance of machine learning-based approach is significantly contingent on the quality of the training data. Consequently, some researchers strive to amalgamate deterministic algorithms with machine learning techniques to enhance overall performance [52].

The majority of NLIDB systems are primarily developed for relational databases. However, owing to the distinctiveness and intricacy of spatial data, one needs NLIDB systems in the spatial database domain in which the availability of suitable training sets is rather limited.

III. The Framework

A. An overview

NALSpatial takes queries over spatial data expressed by natural languages in English as input. The output is the transformed executable language, contingent upon the specific database management system in use. The framework consists of two phases: (i) *natural language understanding* and (ii) *natural language translation*, as illustrated in Figure 2.

The task of phase (i) is to analyze a natural language query to obtain accurate semantic meaning and context. In order to effectively extract key entity information, NLP techniques and deep learning algorithms are utilized and the corpus and knowledge bases are applied. Then, the procedure comprehends the query intent and determines the query type. Users are able to customize queries during this phase. NALSpatial adapts to a variety of spatial data types and query operations, providing a flexible and sustainable solution for evolving data requirements. The task of phase (ii) is to transform the comprehended query into an executable language to manipulate the underlying spatial data. Key semantic information and query operators are combined to construct the executable language. NALSpatial achieves high adaptability to the underlying database by dynamically adjusting the structured language models. This design ensures the generality of NALSpatial, enabling it to seamlessly integrate and run on any spatial database without excessive modifications.

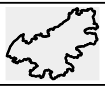
Relation	Schema	Instance
district	((Name string) (GeoData region))	Jiangning District 
road	((Name string) (GeoData line))	Zhixing Road
poi	((Name string)(Type string) (GeoData point))	Meiling Palace ● (118.85, 32.05)
park	((Name string) (GeoData point))	Yanziji Park ● (118.82, 32.15)

Fig. 3. An example dataset.

B. Query transformation flow

EXAMPLE 3. NLQ_3: *"Which district has the largest number of parks in Nanjing?"*

Considering NLQ_3 and the dataset in Figure 3, the workflow is as follows.

Extract the key semantic information, including (i) *the entities* and (ii) *the query type*. The named entity recognition function of *spaCy* is employed to obtain cardinal and quantity lists from the NLQ. By making use of the position of the word *nearest*, *closest* or *neighbor*, the number of nearest neighbors is determined from the cardinal list. According to the presence of the word *meter* or *kilometer*, the distance threshold is determined from the quantity list. Furthermore, by utilizing the tokenization function of *spaCy*, a list of nouns is obtained from the NLQ. Subsequently, the spatial relations and locations are determined based on whether the nouns in the list exist in the knowledge bases. Finally, the query type is determined using the model trained on the corpus. The spatial relations of NLQ_3 are determined as *district* and *park* and the query type is an aggregation query for the maximum value.

Query composition. We combine operators to pre-build structured language models for each type of query. Aggregation queries for the maximum value in the SECONDO system use operators (i) *interior*, (ii) *sortby* and (iii) *head*. The operator *interior* determines whether a point is inside a region. The operator *sortby* sorts a relation by attributes in ascending or descending order. The operator *head* extracts the first *i* objects of a relation according to the parameter *i*. In Section V, we illustrate structured language models for the SECONDO system and define rules for mapping entities in natural language queries to undetermined elements in the models. According to the extracted key semantic information, structured language models and mapping rules, executable database queries can be combined.

Furthermore, NALSpatial is able to transform non-spatial NLQ into SQL, as illustrated in Figure 2.

Syntactic analysis. Following the identification as a type of non-spatial query, the syntactic analysis is conducted on the natural language query to specify its goal and condition. The goals of queries can be divided into two categories: (i) *attributes of relations, e.g. "population"*, (ii) *functions of attributes, e.g. "the largest population"*. The conditions of queries consist of attribute names, relationship words, and attribute values. Relationship words are classified into two types: (i) *comparative relationship, e.g. "population greater than 4000"*, (ii) *fixed collocation, e.g. "population between 4000 and 5000"*. We pre-construct two mapping tables, one storing attributes and their corresponding description in natural language, and the other storing relationship words and their corresponding description in natural language. During the process of syntactic analysis, we utilize the mapping tables and knowledge bases, employing string matching algorithms to determine the goals and conditions of queries. If the goal and condition of the query do not belong to the same relation, we need to join different relations.

Combination of SQL. The identified goal and condition of the query, and relation are transformed into the SELECT clause, WHERE clause and FROM clause, respectively. The final generated SQL is as follows:

SELECT *⟨goal of the query⟩*
FROM *⟨table name⟩*
WHERE *⟨condition of the query⟩;*

EXAMPLE 4. *NLQ_4: "Return the largest postcode of cities with a population greater than 500000."*

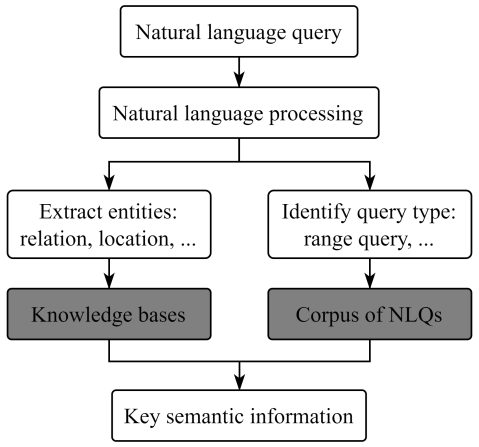

Natural language query
Natural language processing
Extract entities: relation, location, ...
Identify query type: range query, ...
Knowledge bases
Corpus of NLQs
Key semantic information

Fig. 4. Natural language understanding.

Performing syntactic analysis on NLQ_4, we identify the goal as "*the largest postcode*" and the condition as "*population greater than 500000*". In accordance with the mapping tables, "*largest*", "*postcode*", "*population*" and "*greater than*" are mapped to the function *MAX*, the attribute *PLZ*, the attribute *Bev* and the relationship $>$, respectively. The goal and condition of NLQ_4 belong to the relation *city*. The goal is transformed into the clause "*SELECT MAX(PLZ)*" and the condition is transformed into the clause "*WHERE Bev* $>$ *500000*". The corresponding SQL for NLQ_4 is:

SELECT *MAX(PLZ)*
FROM *city*
WHERE *Bev* $>$ *500000;*

IV. Natural Language Understanding

This process is conducted in tandem with the location knowledge base and spatial relation knowledge base, facilitating the extraction of key entity information, as illustrated in Figure 4. The entity information includes the number of nearest neighbors, distance threshold, spatial relations and locations. Furthermore, the pre-constructed corpus is used to train an LSTM network as a model that recognizes query types. The relevant definitions are as follows.

DEFINITION 1 (The number of nearest neighbors). Let k be the number of nearest neighbors, and $word\_NN$ be a word from $\{$"$nearest$", "$closest$", "$neighbor$"$\}$. If a natural language query does not contain $word\_NN$, we set $k = 0$. If a natural language query contains $word\_NN$, let K be the set of cardinal numbers in the natural language query.

(i) If $|K| = 0$, $k = 1$.
(ii) If $|K| = 1$, $k = k_1 \in K$.
(iii) If $|K| > 1$, let $D\left(word_1, word_2\right)$ be the number of words between $word_1$ and $word_2$ in a natural language query. $k = n \in K$, where $\forall k_i \in K$, we have

$$D\left(n, word\_NN\right) = \min_{1 \leq i \leq |K|} D\left(k_i, word\_NN\right) \quad (1)$$

EXAMPLE 5. *Consider the query "List the 12 closest parks to the border of Nanjing.". We have "closest" and $K = \{12\}$. According to Definition 1, we have $k = 12$. Consider the query*

"List the 12 closest parks to the road 3.". We have "closest" and $K = \{12, 3\}$. The distance between 12 and the word "closest" is the shortest for all cardinal numbers in K and thus we have $k = 12$.

DEFINITION 2 (**Entity).** An entity is a tuple $E = (k, d, relation, location)$, in which

(i) k stores the number of nearest neighbors.
(ii) d stores the distance threshold.
(iii) $relation$ is an array of dictionaries storing spatial relations.
(iv) $location$ is an array of strings storing locations.

A. Entity extraction for queries

In light of (i) *the need for expeditious processing and comprehensive functionality*, and (ii) *the preference for Python as the development language*, *spaCy* is chosen as the NLP tool, with *NLTK* as a potential alternative. The framework implementation predominantly relies on the tokenization and named entity recognition functions offered by *spaCy*.

Before processing natural language queries, we construct two knowledge bases with the foundation of spatial database: (i) *location knowledge base* and (ii) *spatial relation knowledge base*. The location knowledge base comprehensively archives details concerning all locations within the spatial database. Simultaneously, the spatial relation knowledge base encapsulates information pertaining to all spatial relations within the spatial database, encompassing unique identifiers, names, and spatial properties. The establishment of the knowledge bases involves the digitalization of location and spatial relation information, which is then segregated and stored in distinct CSV files, thereby constituting the foundation for the location knowledge base and the spatial relation knowledge base.

We propose an algorithm for extracting key entity information from NLQs over spatial data, including (i) *the number of nearest neighbors*, (ii) *distance threshold*, (iii) *spatial relations* and (iv) *locations*. The specific details are outlined in Algorithm 1. To extract the number of nearest neighbors, the named entity recognition function of *spaCy* is used to obtain a list of cardinal numbers from the NLQ. Subsequently, in accordance with Definition 1, the number of nearest neighbors is determined. In the process of extracting the distance threshold, the named entity recognition function of *spaCy* is utilized to collect a list of quantities from the NLQ. The distance threshold is expressed as a quantity phrase indicative of a distance value. Leveraging the number and the associated distance unit in the phrase, the precise value of the distance threshold in meters is determined. For the extraction of spatial relations and locations, the tokenization function of *spaCy* is used to obtain a list of nouns or proper nouns from the NLQ. Then according to whether the words in the list correspond to entries in the knowledge bases, the spatial relations and locations are determined. The knowledge bases are searched using hash tables. Conflicting elements are stored by linked lists and a dynamic resizing strategy is employed to enhance performance and memory utilization. Given the hash table capacity and number of elements, denoted as *capacity* and *size*, respectively, the load factor is defined by

Algorithm 1 EIE (Entity information extraction)

```
Input: natural language expression, $Q$;
       location knowledge base, $LKB$;
       spatial relation knowledge base, $SRKB$;
Output: an entity, $E$
 1: $doc \leftarrow nlp\ (Q)$
 2: for $token \in doc$ do
 3:     if $token$ means the number of neighbors then        ▷ Definition 1
 4:         $E.k \leftarrow token$
 5:     if $token$ means the distance threshold then
 6:         $E.d \leftarrow get\_threshold(token)$
 7:     if $token.pos\_$ = $NOUN$ $\vee token.pos\_$ = $PROPN$ then
 8:         $noun\_list.append(token.text)$
 9: for $word \in noun\_list$ do
10:     if $search(word, SRKB.name)$ then
11:         $E.relation.append(word)$
12:     if $search(word, LKB.name)$ then
13:         $E.location.append(word)$
14: return $E$
```

$load\_factor = \frac{size}{capacity}$. The strategy expands or reduces the hash table and re-hashes all elements when the load factor exceeds or falls below the specified threshold. R1.C1

The analysis of time complexity. Assuming the number of words in a natural language query is *len*, *m* represents the number of locations in the location knowledge base and *n* is the number of relations in the spatial relation knowledge base, we can deduce that *m* is greater than *n* (since a spatial relation involves at least one location). The time complexity of dynamically adjusting the hash table is *O(m)*. Hash table expansion is typically conducted in a multiplicative manner, resulting in a low frequency of rehashing operations. Assuming the initial capacity of the hash table is 64, expansions are triggered when the number of elements reaches $\lfloor load\_factor \times 64 \rfloor + 1$, $\lfloor load\_factor \times 128 \rfloor + 1$, or $\lfloor load\_factor \times 256 \rfloor + 1$. The time complexity of searching the knowledge bases is *O(1)*, resulting in *O(len)* for Algorithm 1. R1.C1

B. Query type identification

We construct a query corpus over spatial data for model training[3]. The corpus contains 3000 queries classified into five categories: (i) *basic query*, (ii) *range query*, (iii) *nearest neighbor query*, (iv) *spatial join query* and (v) *aggregation query*. Each query is marked with accurate type. The queries are extracted from 60 relevant research papers and further expanded manually. The construction process of the corpus is detailed as follows:

(i) *Query extraction.* We extract 60 robust NLQs over spatial data from papers in the relevant domain, as presented in Table II. Under the guidance of experts in the field of spatial databases, the queries are reviewed to ensure the relevance to the research on natural language transformation.

[3]https://github.com/nuaaer16/NALSpatial/tree/main/LSTM

TABLE II
EXAMPLE QUERIES IN THE CORPUS

Example NLQ		Type
Original RLS	Calculate the distance from Mount Huang to Mount Tai. What's the distance from Mount Huang to Mount Tai?	BQ
Original RLS	What restaurants are located in Jiangning District? Could you provide a list of restaurants located in Jiangning District?	RQ
Original RDE	Find the two nearest lakes to West Lake. Find the five nearest lakes to West Lake.	NNQ
Original RDE	What restaurants are within 2 kilometers of each subway station? What parks are within 2 kilometers of each subway station?	SJQ
Original RLS	How many amusement parks are there in Los Angeles? Please provide the number of amusement parks in Los Angeles.	AQ

RDE, Replace data entities; RLS, Replace language styles;
BQ, Basic query; RQ, Range query; NNQ, Nearest neighbor query;
SJQ, Spatial join query; AQ, Aggregation query.

TABLE III
PERFORMANCE OF DIFFERENT TRAINING MODELS ON CORPUS

Model	Training time	Prediction accuracy
TextCNN	5s	98.79%
LSTM	1s	99.60%
BERT	92s	99.58%

(ii) *Manual expansion.* We utilize ChatGPT to perform data augmentation by replacing data entities and language styles to expand the corpus to 3000 queries. Three experts are then selected to review the queries produced by ChatGPT based on the following criteria, including compliance of query types, clarity of semantics, unambiguity of entity expression, and absence of grammatical errors. Through manual review and proofreading, we ensure the high quality of the corpus, making it applicable to a wide range of research on spatial data queries and practical application scenarios.

R1.C2
R3.C2

Considering the examples in Table II, for the query *"Find the two nearest lakes to West Lake."*, we identify the word *"two"* as a number of nearest neighbors. New queries are generated by replacing *"two"* with other numbers. For the query *"Calculate the distance from Mount Huang to Mount Tai."*, we recognize the words *"Mount Huang"* and *"Mount Tai"* are locations. Keeping the locations constant, new queries are generated by expressing the distance calculation using alternative language styles.

When selecting an appropriate model, the corpus is partitioned into a training set and a test set with a ratio of 8:2. TextCNN, LSTM, and BERT models are employed for training, and the results are presented in Table III. Both LSTM and BERT exhibit comparable prediction accuracy on the test set, surpassing the performance of TextCNN. However, the training time for BERT is approximately 1.5 minutes due to its large number of parameters, whereas LSTM completes training in approximately 1 second. BERT does not have a significant advantage in solving classification challenges involving small samples, multiple categories and short texts. On the contrary, TextCNN and LSTM with simple structures and less training time, are viable solutions. LSTM network is chosen based on the following considerations:

(i) For classification tasks, although feedforward networks like CNN exhibit superior performance, LSTM can faithfully represent or simulate human behaviors, logical developments and cognitive processes of neural organizations, and it is suitable for handling complex tasks [53].

(ii) The number of queries in the corpus is not particularly large and training overly complex deep learning models probably results in overfitting. Compared to recently proposed models such as Transformer and BERT, LSTM is simple and easy to implement.

V. NATURAL LANGUAGE TRANSLATION

The selection of the structured language model is contingent upon the query type, and subsequently, the key entity information is filled into the model following the mapping rules, culminating in the derivation of the executable language. The relevant definitions are as follows.

Let $Type(p) \in \{point, line, region\}$ denote the data type of a location p. We define a generic function $compLoc$ to determine if locations p and q share any common parts, namely:

$$compLoc(p,q) = \begin{cases} intersects(p,q) & \text{if } Type(p) \in \{line, region\} \\ & \wedge Type(q) \in \{line, region\} \\ within(p,q) & \text{if } Type(p) = point \\ & \wedge Type(q) = region \end{cases} \tag{2}$$

DEFINITION 3 (**Spatial range query**). Given a set of spatial objects R and a query location p, the spatial range query returns all objects fulfilling the condition involving a certain spatial predicate and the query location, denoted by set $R^{'} = \{r \mid r \in R \wedge compLoc(r,p)\}$.

DEFINITION 4 (**Nearest neighbor query**). Given a set of spatial objects R, a query location p and a positive integer k, the nearest neighbor query returns a set $R^{'} = \left\{r_1^{'}, ..., r_k^{'}\right\} \subseteq R$, where $\forall r \in R - R^{'}$, we have

$$dist(r,p) \geq \max_{1 \leq i \leq k} dist\left(r_i^{'}, p\right) \tag{3}$$

DEFINITION 5 (**Spatial join query**). Spatial join queries are categorized into two types according to the join conditions.

- Given two sets of spatial objects R and S, the spatial join query returns a set $T = \{(p,q) \mid p \in R \wedge q \in S \wedge compLoc(p,q)\}$.
- Given two sets of spatial objects R and S with a distance threshold d, the distance join query returns a set $T = \{(p,q) \mid p \in R \wedge q \in S \wedge dist(p,q) \leq d\}$.

DEFINITION 6 (**Spatial aggregation query**). Aggregation queries are categorized into three types based on the aggregation functions.

- Given a set of spatial objects R and a location p, the spatial aggregation query for quantity returns an integer $n = \left|R^{'}\right|$, where $R^{'} = \{r \mid r \in R \wedge compLoc(r,p)\}$. The result of the aggregation query for quantity on two sets of spatial objects R and S is a set $R' = \{r' \mid r' \in R \wedge r'.Cnt = \left|S'\right|, S' = \{s \mid s \in S \wedge compLoc(s, r')\}\}$.

- Given a set of spatial objects $R = \{r_1, ..., r_n\}$ and a location p, the spatial aggregation query for sum returns a floating-point number

$$f = \sum_{i=1}^{n} area\left(intersection\left(r_i, p\right)\right) \tag{4}$$

- Given two sets of spatial objects R and S, the spatial aggregation query for maximum value returns an object $p \in R'$, where $R' = \{r' \mid r' \in R \wedge r'.Cnt = \left|S'\right|, S' = \{s \mid s \in S \wedge compLoc(s, r')\}\}$, we have

$$p.Cnt = \max_{1 \le i \le |R|} r_i'.Cnt \tag{5}$$

DEFINITION 7 (Structured language models, SLMs). SLMs are built upon syntax rules and represent executable database queries by combining operators and entities. Formally, $SLM = (L, O, E)$, in which

(i) L is the set of syntax rules for the database language.
(ii) O is the set of operators in the database.
(iii) $E = E_k \cup E_d \cup E_{place} \cup E_{relation}$, is the set of entities. Here, E_k, E_d, E_{place} and $E_{relation}$ respectively denote the set of numbers of nearest neighbors, distance thresholds, locations and spatial relations.

EXAMPLE 6. *Consider the nearest neighbor query in the SECONDO system. We have L = {query ⟨value expression⟩;}, O = {createtree, distancescan}, $E = E_k \cup E_{place} \cup E_{relation}$, and SLM in Table IV.*

A. Structured language models

We outline the structured language models in Table IV. In the models, ⟨*k*⟩, ⟨*d*⟩, ⟨*place*⟩ and ⟨*relation*⟩ represent the undetermined number of nearest neighbors, distance threshold, location and spatial relation, respectively. If a location is stored within a spatial relation, the ⟨*place*⟩ in the model is replaced with the following statement, where ⟨*tmp_relation*⟩ and ⟨*name*⟩ respectively represent the spatial relation and the name of the location.

(⟨ *tmp_relation* ⟩ *feed filter* [*.Name* = " ⟨ *name* ⟩ "] *extract* [*GeoData*])

We present key operators in Table V including name, signature and meaning. For example, the operator *feed* converts a relation into a stream of tuples. The operator *consume* collects a stream of tuples and converts them into a relation in the database, and the operator *count* outputs the number of tuples. The operator *filter*, followed by a filter condition, collects tuples from the stream that satisfy the condition.

B. Structured language construction

In the natural language understanding phase, we obtain key entities and the type of NLQ. Entities and the structured language model are selected according to the query type, and then executable database queries are constructed based on mapping rules. The process of structured language construction is illustrated in Figure 5.

TABLE IV
STRUCTURED LANGUAGE MODELS

	Structured language model
(i) Basic query	
distance	query **distance** (⟨place1⟩, ⟨place2⟩);
direction	query **direction** (⟨place1⟩, ⟨place2⟩);
length	query **size** (⟨place⟩);
area	query **area** (⟨place⟩);
(ii) Range query	
range query	query ⟨relation⟩ feed filter [.GeoData **intersects** ⟨place⟩] consume;
(iii) Nearest neighbor query	
nearest neighbor	query ⟨relation⟩ **creatertree**[GeoData] ⟨relation⟩ **distancescan** [⟨place⟩, ⟨k⟩] consume;
(iv) Spatial join query	
location join	query ⟨relation1⟩ feed a ⟨relation2⟩ feed b **symmjoin** [.GeoData_a **intersects** ..GeoData_b] consume;
distance join	query ⟨relation1⟩ feed a ⟨relation2⟩ feed b **symmjoin** [**distance**(.GeoData_a, ..GeoData_b) ≤ ⟨d⟩] consume;
(v) Aggregation query	
quantity	query ⟨relation1⟩ feed extend [Cnt: fun(t: TUPLE) ⟨relation2⟩ feed filter [.GeoData **intersects** attr(t,GeoData)] **count**] consume;
sum	query ⟨relation⟩ feed extend [IntersectionArea: **area** (**intersection** (.GeoData, ⟨ place ⟩))] **sum** [IntersectionArea];
maximum	query ⟨relation1⟩ feed extend [Cnt: fun(t: TUPLE) ⟨relation2⟩ feed filter [.GeoData **intersects** attr (t,GeoData)] count] **sortby** [Cnt desc] **head** [1] consume;

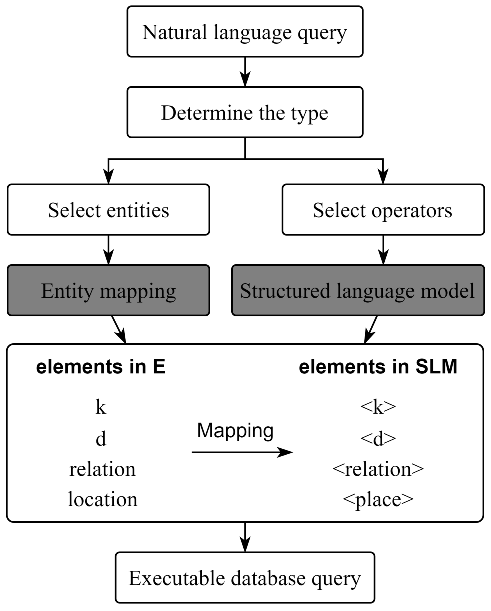


Fig. 5. Structured language construction.

Mapping rules. When constructing an executable language, according to the rules shown in Figure 5, each element in the entity *E* can be mapped to the corresponding element in the set of undetermined elements {⟨*k*⟩, ⟨*d*⟩, ⟨*place*⟩, ⟨*relation*⟩} in the structured language models *SLM*. A distinct relational link

TABLE V
OPERATORS IN SECONDO

Operator	Signature	Meaning
filter	stream × filter condition → stream	Filter the elements of the stream by a predicate.
distance	{point, line, region} × {point, line, region} → real	Compute the distance between two spatial objects.
direction	point × point → real	Compute the direction between two points.
size	line → real	Return the length of a line.
area	region → real	Return the area of a region.
intersects	{line, region} × {line, region} → bool	TRUE, if both arguments intersect.
intersection	{point, line, region} × {point, line, region} → T, where T is point if point is one of the arguments, otherwise T is the argument having the smaller dimension	Intersection of two spatial objects.
distancescan	rtree × relation × object × int k → stream	Compute the k nearest neighbors for a query object.
sortby	stream × attribute × asc / desc → stream	Sort a stream of tuples by a given list of attributes.
head	stream × int n → stream	Fetch the first n elements from a stream.

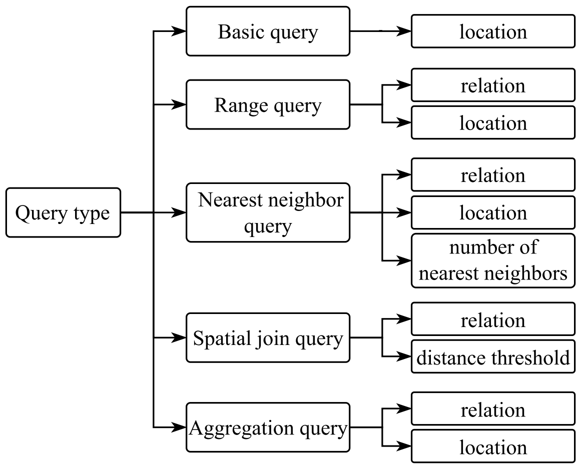


Fig. 6. Entities required for each query type.

exists between *E* and *SLM*.

$$E \rightarrow SLM \tag{6}$$

Our NALSpatial supports five types of queries including (i) *basic queries*, (ii) *range queries*, (iii) *nearest neighbor queries*, (iv) *spatial join queries* and (v) *aggregation queries*. Distinct types require specific entities, as depicted in Figure 6. We explain the process of constructing the structured language for each type of queries through illustrative examples of NLQs.

Basic query. When transforming a basic query, the required entity is *location*. The operators used in the query are (i) *distance*, (ii) *direction*, (iii) *size* and (iv) *area*. The *distance* operator returns the distance between two locations, the *direction* operator furnishes the angle from one location to another, the *size* operator provides the length of a line and the *area* operator computes the area of a region.

Range query. When transforming a range query, the necessary entities include (i) *relation* and (ii) *location*. If the spatial attribute of *relation* and the data type of *location* are line or region, the *intersects* operator is utilized to return all objects in *relation* that intersect *location*. If the spatial attribute of *relation* is point, the data type of *location* is restricted to region and the *ininterior* operator is utilized to return all objects in *relation* that are located within *location*.

EXAMPLE 7. *NLQ_5: "What are the roads that intersect Zhixing Road?"*

During the natural language understanding phase, we determine that the type is range query with the spatial relation being road and the location being Zhixing Road. The executable language for NLQ_5 is:

query road feed filter [.GeoData ***intersects*** *(road feed filter [.Name = "Zhixing Road"] extract [GeoData])] consume;*

In the executable language for NLQ_5, both the spatial attribute of *road* and the data type of *Zhixing Road* are line, and the *intersects* operator is used. According to the knowledge bases, *Zhixing Road* is stored in the relation *road*. To map ⟨*place*⟩ in the model, the following statement should be used instead of *"Zhixing Road"*.

(road feed filter [.Name = " Zhixing Road"] extract [GeoData])

Nearest neighbor query. When transforming a nearest neighbor query, the required entities encompass (i) *relation*, (ii) *location* and (iii) *the number of nearest neighbors*. The operators used in the query are (i) *createtree* and (ii) *distancescan*. The operator *createtree* creates a R-Tree for *relation*. The operator *distancescan* computes the *k* nearest objects in the *relation* to the *location*, based on the provided location, objects indexed by the R-tree and the number of nearest neighbors.

EXAMPLE 8. *NLQ_6: "Please tell me what are the 10 closest POIs to the border of Nanjing?"*

During the natural language understanding phase, we determine that the type is nearest neighbor query with the spatial relation being poi, the location being NJBorderLine and the number of nearest neighbors being 10. The executable language for NLQ_6 is:

query poi ***creatertree****[GeoData] poi* ***distancescan*** *[NJBorderLine, 10] consume;*

Spatial join query. When transforming a spatial join query, the essential entities involve (i) *relation* and (ii) *distance threshold*. The operators used in the query are (i) *symmjoin*, (ii) *intersects* and (iii) *distance*. The *symmjoin* operator combines objects from two relations, similar to the keyword JOIN in SQL. The operators *intersects* and *distance* are used to form the join condition of the *symmjoin* operator.

TABLE VI
THE GENERALITY OF STRUCTURED LANGUAGE MODELS

	SECONDO	PostGIS
L	query ⟨value expression⟩; let ⟨identifier⟩ = ⟨value expression⟩; delete ⟨identifier⟩;	SELECT ⟨value expression⟩; CREATE ⟨object⟩ ⟨identifier⟩; DROP ⟨object⟩ ⟨identifier⟩;
O	distance area intersects intersection	ST_Distance ST_Area ST_Intersects ST_Intersection
E	$E_k \cup E_d \cup E_{place} \cup E_{relation}$	
SLM	query distance (⟨place1⟩, ⟨place2⟩); query ⟨relation⟩ feed filter [.GeoData intersects ⟨place⟩] consume;	SELECT ST_Distance (⟨place1⟩, ⟨place2⟩); SELECT * FROM ⟨relation⟩ WHERE ST_Intersects (⟨ relation ⟩.GeoData, ⟨place⟩);

EXAMPLE 9. *NLQ_7: "What are the POIs within 1.5 kilometers from the center of each district in Nanjing?"*

During the natural language understanding phase, we determine that the type is distance join query, involving the spatial relations poi and district, and the distance threshold is 1500 meters. The executable language for NLQ_7 is:

*query poi feed a district feed b **symmjoin** [**distance** (.GeoData_a, ..GeoData_b) ≤ 1500.0] consume;*

Aggregation query. When transforming an aggregation query, the required entities consist of (i) *relation* and (ii) *location*. The operators used in the query are (i) *count*, (ii) *sum*, (iii) *sortby*, (iv) *head*, (v) *intersects* and (vi) *intersection*. The operator *count* is used with *intersects* to calculate the number of objects in the relation that intersect the location. The operators *sum* and *intersection* are collaboratively utilized to compute the total area of the intersection between the relation and the location. The operator *sortby* is used with *head* to determine the maximum of an attribute in the relation.

EXAMPLE 10. *NLQ_8: "Can you tell me the total area of the districts in Nanjing intersecting with the Yangtze River Basin?"*

During the natural language understanding phase, we determine that the type is aggregation query for sum, with the spatial relation being district and the location being Yangtze River Basin. The executable language for NLQ_8 is:

*query district feed extend [IntersectionArea: **area** (**intersection** (.GeoData, Yangtze River Basin))] **sum** [IntersectionArea];*

C. Discussion

The generality of structured language models. The structured language models in Table IV can be applied to spatial database systems other than SECONDO, as long as the operators and functions are replaced with operators and functions of the counterpart functionality in other systems. For example, the correspondence between SECONDO and PostGIS is given in Table VI. In the CREATE and DROP statements of PostGIS, ⟨*object*⟩ denotes operation objects such as FUNCTION, INDEX, TABLE and TYPE. For PostGIS, the structured language model for a range query can be obtained from Table VI as follows:

SELECT ***
FROM *⟨relation⟩*
WHERE *ST_Intersects(⟨relation⟩.GeoData, ⟨place⟩);*

In the model, the operator *ST_Intersects* is used.

When transforming the natural language of a nearest neighbor query, the operators used in PostGIS are (i) *ST_Distance*, (ii) *ORDER BY* and (iii) *LIMIT*. The *ST_Distance* operator is used to calculate the distance. This operator is integrated into the query alongside the *ORDER BY* clause, which facilitates sorting results in ascending order by the calculated distances. To further refine the output, the *LIMIT* clause is applied, restricting the results to the top *k* spatial objects.

TABLE VII
SPATIAL DATA STATISTICS

Dataset	#tables	#points	#lines	#regions
berlintest	50	3040	4708	330
nanjingtest	6	9000	887	13
chinawater	2	0	8399	2907

VI. EXPERIMENTAL EVALUATION

The framework is implemented in a computer (Intel(R) Core(TM) i5-10210U CPU, 1.60 GHz, 8 GB memory, 512 GB disk) running Ubuntu 20.04 (64 bits, kernel version 5.14.0-1051-oem). NALSpatial is integrated into the SECONDO system as an algebra module[4], and an operator named *spatial_nl* is developed. Within the system, users can use the operator to transform natural language queries over spatial data into executable database queries.

A. Datasets

Three datasets are used, and the statistics are reported in Table VII. (i) The *berlintest* dataset encapsulates urban geographic data of Berlin, encompassing public transport, POIs and rivers, with 50 spatial relations and 8078 locations. The locations are comprised of 3040 points, 4708 lines and 330 regions. The dataset is available in SECONDO by default. (ii) The *nanjingtest* dataset stores districts, part of roads and POIs information in Nanjing, with 6 spatial relations and 9900 locations. The locations consist of 9000 points, 887 lines and 13 regions. (iii) The *chinawater* dataset stores partial water system in China, including diverse water bodies such as rivers, lakes and ponds, with 2 spatial relations and 11306 locations. The locations involve 8399 lines and 2907 regions.

Test cases. We construct 100 test cases, including (i) *20 range queries*, (ii) *20 nearest neighbor queries*, (iii) *30 spatial join queries* and (iv) *30 aggregation queries*. The structure and semantics of basic queries are relatively straight forward compared to the other four types of queries over spatial data. The test cases do not contain instances of basic queries due to simplicity. The detailed query list is provided in Appendix.

[4]Our code is publicly available at https://github.com/nuaaer16/NALSpatial.

B. Evaluation metrics

We define three metrics to evaluate the transformation method including (i) *response time*, (ii) *translatability* and (iii) *translation precision*.

DEFINITION 8 (Translatability). Given the set of executable queries generated by the system and the set of input natural language queries, denoted by E and N, respectively, the translatability is defined by $T = \frac{|E|}{|N|}$.

DEFINITION 9 (Translation precision). Given the set of executable queries that meet the expected results and the set of natural language queries input into the system, denoted by ER and N, respectively, the translation precision is defined by $TP = \frac{|ER|}{|N|}$.

Response time denotes the duration necessary for the system to translate the input NLQ into the executable language. Translatability characterizes the likelihood of the system accurately translating NLQs into executable database queries. The metric is quantified as the ratio of accurately translated queries to the overall number of queries presented to the system. Translation precision refers to the probability that the output result of the translated executable language matches the expected outcome, and it is defined as the ratio of executable queries producing the expected results to the total number of queries.

C. Experimental results

We conduct two experiments: (i) *comparative experiment* and (ii) *performance verification experiment*.

1) Comparative experiment: We implement three baseline methods: SpatialNLI [47], IRNet [48] and ATHENA++ [54]. Our framework is compared with the three methods in terms of response time, translatability and translation precision. Additionally, we conduct a comparison with GPT-4o, focusing on translatability and translation precision.

In the test cases, the average response time, translatability and translation precision of NALSpatial, SpatialNLI, IRNet and ATHENA++ are shown in Figure 7. Due to the spatial comprehension model, SpatialNLI has the most average response time and the highest translation precision. IRNet demonstrates an average response time comparable to NALSpatial, yet exhibits lower translatability and translation precision than NALSpatial. Based on the identification of key entity information using the spatial domain ontology, ATHENA++ has the least average response time but the lowest translatability and translation precision. Considering the average response time, translatability and translation precision comprehensively, NALSpatial achieves the best performance.

Performance of NALSpatial. The experimental results show that (i) *the average response time is about 2.5 seconds*, (ii) *the translatability is 95%* and (iii) *the translation precision is 92%*. The queries that the method fails to translate accurately are Q6, Q14, Q17, Q41 and Q54. The reason for Q6 is that the method mistakenly identifies *Road* as a spatial relation and fails to recognize *Lingrui Road* as a location. The reason for Q14 is that *spaCy* incorrectly identifies *flaechens* as a verb, and the method fails to recognize the *Flaechens* relation. The reason for Q17 is that the *berlintest* dataset stores street information in the *strassen* relation, and there is no *street* relation. The method is unable to recognize the semantic similarity between *street* and *strassen*. The reason for Q41 is that the method mistakenly identifies *radius* as a location *Radis* in the *POI* relation. The reason for Q54 is that the model for identifying query types incorrectly classifies the query as a range query, and the query is a spatial join query. The executable languages generated by the method for Q21, Q31 and Q79 do not match the expected results. The reason for Q21 is that the method mistakenly interprets the intention of the query as finding the 10 nearest POIs to the city center of Nanjing, rather than the boundary line of Nanjing. The reason for Q31 is that the method mistakenly identifies *Mehringdamm* as a location in the relation *UBahnhof* instead of the intended relation *strassen*. The reason for Q79 is that the method mistakenly interprets the intent as an aggregation for sum, and the query is an aggregation query for quantity.

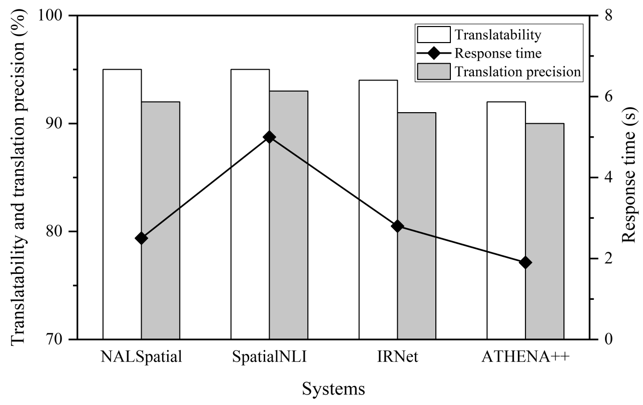


Fig. 7. The average response time, translatability and translation precision of the systems.

SpatialNLI. The experimental results show that (i) *the average response time is about 5 seconds*, (ii) *the translatability is 95%* and (iii) *the translation precision is 93%*. The queries that SpatialNLI does not accurately translate are Q6, Q14, Q41, Q54 and Q93. The reasons are the same as NALSpatial. For Q93, the method mistakenly identifies *rbahn* as the *UBahn* relation using the edit distance. Using cosine similarity, the method can recognize the semantic similarity between *street* and *strassen*, allowing Q17 to be accurately translated into an executable database query. The executable languages generated by the method for Q21 and Q79 do not match the expected results. The spatial comprehension model identifies *Mehringdamm* as a location in the *strassen* relation, and the executable language corresponding to Q31 aligns with the expected results.

IRNet. The experimental results show that (i) *the average response time is about 2.8 seconds*, (ii) *the translatability is 94%* and (iii) *the translation precision is 91%*. The queries that IRNet is unable to accurately translate are Q14, Q17, Q41, Q42, Q50 and Q54. The reasons for Q14, Q17, Q41 and Q54 are the same as in NALSpatial. The reasons for Q42 and Q50 are that the NLP tool *NLTK* fails to recognize the

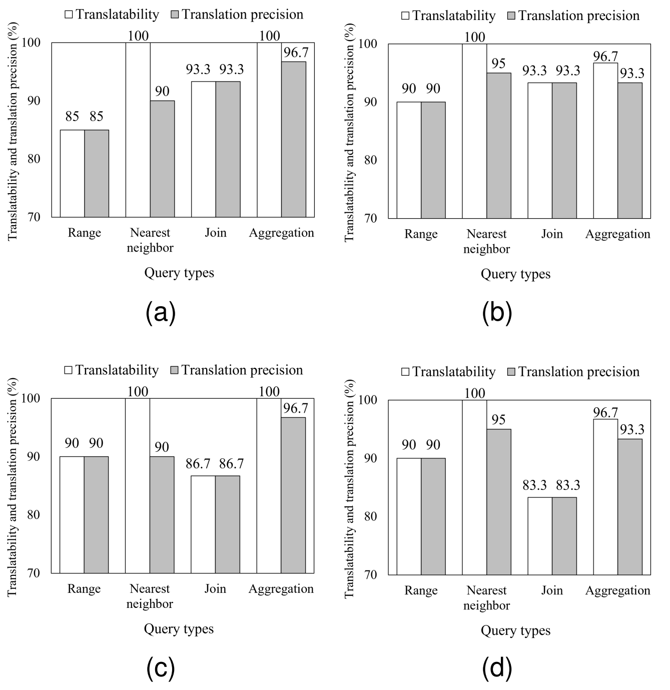


Fig. 8. Translatability and translation precision of each query type. (a) NALSpatial. (b) SpatialNLI. (c) IRNet. (d) ATHENA++.

phrase consisting of decimals and meter or kilometer as a quantity. The method is able to identify *Lingrui Road* as a location and Q6 can be accurately translated into an executable language. In the generated executable languages, the NLQs that yield results inconsistent with expectations are the same as in NALSpatial, and the same reasons also apply here.

ATHENA++. The experimental results show that (i) *the average response time is about 1.9 seconds*, (ii) *the translatability is 92%* and (iii) *the translation precision is 90%*. The method fails to translate the queries Q14, Q17, Q41, Q42, Q50, Q54, Q61 and Q88. The reasons for the first six queries are the same as in IRNet. The reasons for Q61 and Q88 are that the spatial domain ontology and the Relational Store cannot recognize *PLZBoundaries* as a spatial relation. The executable languages generated by the method for Q31 and Q79 do not match the expected results. The reasons are the same as in NALSpatial. Leveraging the spatial domain ontology, the method accurately understands the intent of Q21 (finding the 10 nearest POIs to the boundary of Nanjing), and the executable language corresponding to Q21 aligns with the expected results.

The translatability and translation precision of each query type in the four methods are shown in Figure 8. Range queries are simple in structure but heavily rely on the extraction of locations. In all methods, the translatability and translation precision of the range query do not exceed 90%. Nearest neighbor queries require the largest number of entities with complex structures, and their formats are similar. In the test cases, the nearest neighbor queries can be well translated into executable statements. The translatability and translation precision of spatial join queries are significantly influenced by NLP tools. Aggregation queries demonstrate relatively stable translation precision, ranging from 93% to 97%.

TABLE VIII
EVALUATION OF NALSPATIAL AND GPT-4O

Framework	RQ		NNQ		SJQ		AQ	
	T	TP	T	TP	T	TP	T	TP
NALSpatial	85%	85%	100%	90%	93.3%	93.3%	100%	96.7%
GPT-4o	45%	45%	75%	75%	83.3%	83.3%	76.7%	76.7%
GPT-4o+1SL	65%	65%	85%	85%	93.3%	93.3%	90%	90%

RQ, Range query; NNQ, Nearest neighbor query; SJQ, Spatial join query; AQ, Aggregation query; T, Translatability; TP, Translation precision.

Furthermore, we compare NALSpatial and GPT-4o in terms of the translatability and translation precision. The evaluation conducted by *Gao et al.* [55] emphasizes that OpenAI Demonstration Prompt for GPT-4 and GPT-3.5-TURBO is a good choice in the zero-shot scenario. Moreover, GPT-4 demonstrates proficiency in learning mappings from question-SQL pairs [56]. Considering the effectiveness of prompts and the complexity of formulation, GPT-4o is prompted using the 1-Shot Learning (1SL) strategy, which provides table schemas (annotated with the "#" sign) and a golden example of transforming natural language into SQL. To illustrate this, the prompt for the *chinawater* dataset is provided in Appendix. The experimental results demonstrate that the 1SL strategy has the potential to enhance the performance of GPT-4o, as illustrated in Table VIII. GPT-4o can accurately represent locations with the assistance of prompts. Nevertheless, the selection of an appropriate operator to determine the spatial relationship between the locations remains a significant challenge. In contrast, NALSpatial employed the knowledge bases to extract and represent locations without additional user interaction, and also permitted the utilization of spatial operators that were already incorporated into the database. The transformation of range queries is contingent upon the identification of locations and the application of spatial operators, making GPT-4o less proficient in transforming range queries than NALSpatial. **R1**.C2 **R3**.C2

2) Performance verification: We evaluate the translation process by configuring the following parameters: (i) *the proportion of the range to the entire data space*, (ii) *the number of neighbors*, (iii) *the join condition*, (iv) *the statistical function* and (v) *the size of the dataset*. The experimental results show that the translatability and translation precision of NALSpatial are not affected by the factors demonstrating the stability of the framework. Furthermore, ablation experiments are conducted to validate the effectiveness of each module in the framework.

The effect of the range size. We progressively increase the range size according to area and construct 10 range queries for each district, targeting the same geographic information. The NLQs are divided into four groups by the range {*Xuanwu*, *Yuhuatai*, *Qixia*, *Jiangning District*}. The average response time is reported in Figure 9a. The translatability is 100% and the translation precision is 90%.

The effect of the number of neighbors. We select 10 nearest neighbor queries and expand to 40 queries by varying the number of nearest neighbors from the domain {1, 8, 16, 32}. The average response time is reported in Figure 9b. The

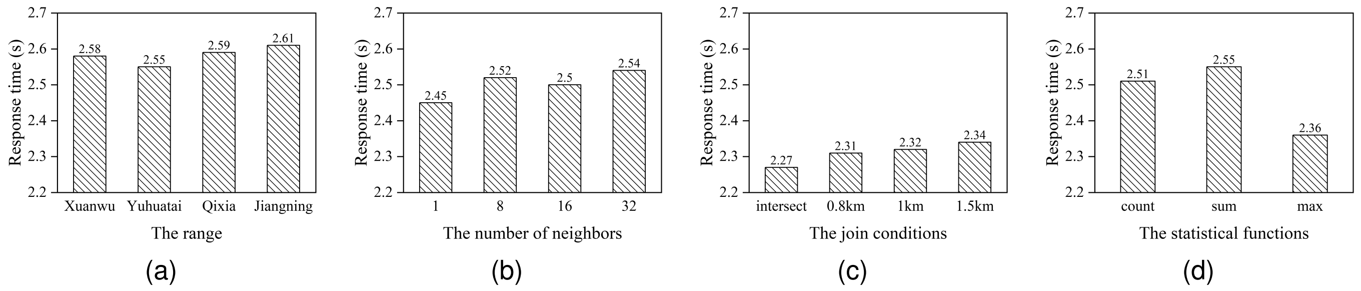


Fig. 9. Response time. (a) The effect of the range size. (b) The effect of the number of neighbors. (c) The effect of join conditions. (d) The effect of statistical functions.

translatability is 100% and the translation precision is 100%.

The effect of join conditions. We select 10 spatial join queries and expand to 40 queries by varying the join conditions to location intersection, distance less than 800 meters, distance less than 1 kilometer or distance less than 1.5 kilometers. The NLQs are divided into four groups according to the join conditions. As Figure 9c shows, the average response time of each group ranges from 2.25s to 2.35s. The translatability and translation precision are both 90%.

The effect of statistical functions. We select 10 aggregation queries and expand to 30 queries by varying the aggregation functions to *count*, *sum* or *max*. The NLQs are divided into 3 groups by the aggregation functions {*count*, *sum*, *max*}. As Figure 9d shows, the translatability is 100% and the translation precision is 90% for all groups. The function *sum* incurs the most average response time, while the function *max* has the least average response time. Aggregation queries with the *sum* function involve location identification and calculation of the intersection area, resulting in the most response time. Aggregation queries with the *max* function mainly involve relation identification, resulting in the least response time.

The effect of the dataset size. The *chinawater* dataset is expanded using HydroSHEDS[5] data. Four groups of tests are conducted by the number of locations in the dataset {10k, 100k, 1M, 10M}. In the test cases, the average response time for each group is approximately 2.5 seconds, the translatability is 95% and the translation precision is 92%. Furthermore, given the *chinawater* dataset with 10k locations, we scale the test cases by the set {10, 20, 40, 80}. The experimental results show that the average response time of each group remains around 2.5 seconds. We conclude that the time cost of the translation process is independent of the dataset size. R1.C1

Ablation experiments. We evaluate the entity information extraction rate *EI*, query type recognition rate *QT*, translatability *T*, translation precision *TP*, and response time *RT* of the ablated frameworks, and the experimental results in the test cases are shown in Figure 10, where -KB, -Corpus, -MR, and -SLM represent the elimination of the knowledge bases, corpus, mapping rules, and structured language models. The experimental results show that when the knowledge bases are removed, the framework is forced to rely on NLP tools to understand semantics, which significantly impairs the ability to extract entities. When eliminating the corpus, only 13 queries are transformed by randomly assigning query types. The framework is unable to achieve efficient generation of structured languages due to the absence of mapping rules. The elimination of the spatial relation knowledge base or structured language models results in the failure to construct structured languages. Therefore, the modules in the framework are interdependent and exhibit optimal conversion performance when operating as a unified system. R2.C2 R3.C1

[5]https://www.hydrosheds.org/products

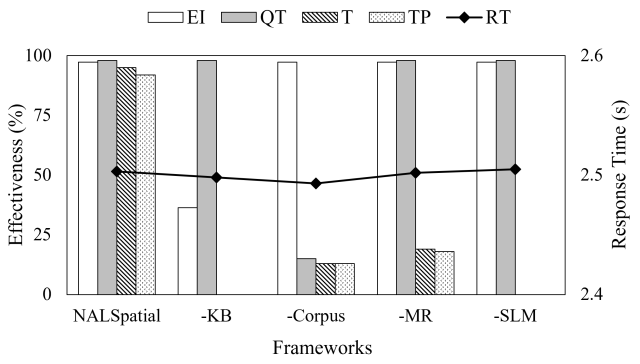


Fig. 10. Results of ablation experiments.

VII. Conclusion

The paper proposes a framework named NALSpatial for transforming NLQs over spatial data into executable database query statements. NALSpatial supports a variety of spatial query types including (i) *basic queries*, (ii) *range queries*, (iii) *nearest neighbor queries*, (iv) *spatial join queries* and (v) *aggregation queries*. We conduct extensive experiments to evaluate NALSpatial in comparison with alternatives. The experimental results indicate that NALSpatial is capable of effectively transforming natural language queries over spatial data into executable database query statements. Future research directions include supporting novel types of spatial queries and extending the coverage of executable query statement.

References

[1] V. Pandey, A. van Renen, A. Kipf, and A. Kemper, “How good are modern spatial libraries?” *Data Sci. Eng.*, vol. 6, no. 2, pp. 192–208, 2021.

[2] Y. Li and D. Rafiei, “Natural language data management and interfaces: Recent development and open challenges,” in *SIGMOD*, 2017, pp. 1765–1770.

[3] A. Shehab, A. Elashry, A. Aboul-Fotouh, and A. M. Riad, “An enhanced partitioning approach in spatialhadoop for handling big spatial data,” *Int. J. Comput. Intell. Syst.*, vol. 16, no. 1, p. 15, 2023.

[4] K. Hori, Y. Sasaki, D. Amagata, Y. Murosaki, and M. Onizuka, “Learned spatial data partitioning,” in *aiDM@SIGMOD*, 2023, pp. 3:1–3:8.

[5] S. Zhang, S. Ray, R. Lu, and Y. Zheng, “SPRIG: A learned spatial index for range and knn queries,” in *SSTD*, 2021, pp. 96–105.

[6] J. Wu, Y. Zhang, S. Chen, Y. Chen, J. Wang, and C. Xing, “Updatable learned index with precise positions,” *Proc. VLDB Endow.*, vol. 14, no. 8, pp. 1276–1288, 2021.

[7] M. H. Moti, P. Simatis, and D. Papadias, “Waffle: A workload-aware and query-sensitive framework for disk-based spatial indexing,” *Proc. VLDB Endow.*, vol. 16, no. 4, pp. 670–683, 2022.

[8] F. Zardbani, N. Mamoulis, S. Idreos, and P. Karras, “Adaptive indexing of objects with spatial extent,” *Proc. VLDB Endow.*, vol. 16, no. 9, pp. 2248–2260, 2023.

[9] J. Zhao, Y. Gao, G. Chen, and R. Chen, “Towards efficient framework for time-aware spatial keyword queries on road networks,” *ACM Trans. Inf. Syst.*, vol. 36, no. 3, pp. 24:1–24:48, 2017.

[10] P. Ahmed, A. Eldawy, V. Hristidis, and V. J. Tsotras, “Reverse spatial top-k keyword queries,” *VLDB J.*, vol. 32, no. 3, pp. 501–524, 2023.

[11] C. Luo, Q. Liu, Y. Gao, L. Chen, Z. Wei, and C. Ge, “TASK: an efficient framework for instant error-tolerant spatial keyword queries on road networks,” *Proc. VLDB Endow.*, vol. 16, no. 10, pp. 2418–2430, 2023.

[12] Q. Wu, Y. Li, H. Li, D. Zhang, and G. Zhu, “AMRAS: A visual analysis system for spatial crowdsourcing,” *Proc. VLDB Endow.*, vol. 15, no. 12, pp. 3690–3693, 2022.

[13] B. Li, Y. Cheng, Y. Yuan, Y. Yang, Q. Jin, and G. Wang, “ACTA: autonomy and coordination task assignment in spatial crowdsourcing platforms,” *Proc. VLDB Endow.*, vol. 16, no. 5, pp. 1073–1085, 2023.

[14] X. Lin, K. Wei, Z. Li, J. Chen, and T. Pei, “Aggregation-based dual heterogeneous task allocation in spatial crowdsourcing,” *Frontiers Comput. Sci.*, vol. 18, no. 6, p. 186605, 2024.

[15] A. Popescu, O. Etzioni, and H. A. Kautz, “Towards a theory of natural language interfaces to databases,” in *IUI*, 2003, pp. 149–157.

[16] Y. Li, I. Chaudhuri, H. Yang, S. Singh, and H. V. Jagadish, “Danalix: a domain-adaptive natural language interface for querying XML,” in *SIGMOD*, 2007, pp. 1165–1168.

[17] Z. Jia, A. Abujabal, R. S. Roy, J. Strötgen, and G. Weikum, “TEQUILA: temporal question answering over knowledge bases,” in *CIKM*, 2018, pp. 1807–1810.

[18] Y. Amsterdamer, A. Kukliansky, and T. Milo, “A natural language interface for querying general and individual knowledge,” *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1430–1441, 2015.

[19] S. Bird, “NLTK: the natural language toolkit,” in *ACL*, 2006, pp. 69–72.

[20] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, “The stanford corenlp natural language processing toolkit,” in *ACL*, 2014, pp. 55–60.

[21] P. Qi, Y. Zhang, Y. Zhang, J. Bolton, and C. D. Manning, “Stanza: A python natural language processing toolkit for many human languages,” in *ACL*, 2020, pp. 101–108.

[22] X. Zhou, Z. Sun, and G. Li, “DB-GPT: large language model meets database,” *Data Sci. Eng.*, vol. 9, no. 1, pp. 102–111, 2024.

[23] H. Kim, B. So, W. Han, and H. Lee, “Natural language to SQL: where are we today?” *Proc. VLDB Endow.*, vol. 13, no. 10, pp. 1737–1750, 2020.

[24] R. H. Güting, T. Behr, and C. Düntgen, “SECONDO: A platform for moving objects database research and for publishing and integrating research implementations,” *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 56–63, 2010.

[25] M. Liu, X. Wang, J. Xu, and H. Lu, “Nalspatial: An effective natural language transformation framework for queries over spatial data,” in *SIGSPATIAL/GIS*, 2023, pp. 57:1–57:4.

[26] A. Eldawy and M. F. Mokbel, “The era of big spatial data: A survey,” *Found. Trends Databases*, vol. 6, no. 3-4, pp. 163–273, 2016.

[27] J. Yu and M. Sarwat, “Geosparkviz: a cluster computing system for visualizing massive-scale geospatial data,” *VLDB J.*, vol. 30, no. 2, pp. 237–258, 2021.

[28] J. Xie, Z. Chen, J. Liu, and et al., “Ganos: A multidimensional, dynamic, and scene-oriented cloud-native spatial database engine,” *Proc. VLDB Endow.*, vol. 15, no. 12, pp. 3483–3495, 2022.

[29] X. Pan, Y. Tong, C. Xue, and et al., “Hu-fu: A data federation system for secure spatial queries,” *Proc. VLDB Endow.*, vol. 15, no. 12, pp. 3582–3585, 2022.

[30] G. Cong, C. S. Jensen, and D. Wu, “Efficient retrieval of the top-k most relevant spatial web objects,” *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 337–348, 2009.

[31] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, “The case for learned index structures,” in *SIGMOD*, 2018, pp. 489–504.

[32] P. Li, H. Lu, Q. Zheng, L. Yang, and G. Pan, “LISA: A learned index structure for spatial data,” in *SIGMOD*, 2020, pp. 2119–2133.

[33] Y. Tong, Z. Zhou, Y. Zeng, L. Chen, and C. Shahabi, “Spatial crowdsourcing: a survey,” *VLDB J.*, vol. 29, no. 1, pp. 217–250, 2020.

[34] Y. Tong, Y. Zeng, B. Ding, L. Wang, and L. Chen, “Two-sided online micro-task assignment in spatial crowdsourcing,” *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 5, pp. 2295–2309, 2021.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in *NIPS*, 2017, pp. 5998–6008.

[36] B. Min, H. Ross, E. Sulem, and et al., “Recent advances in natural language processing via large pre-trained language models: A survey,” *ACM Comput. Surv.*, vol. 56, no. 2, pp. 30:1–30:40, 2024.

[37] T. B. Brown, B. Mann, N. Ryder, and et al., “Language models are few-shot learners,” *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[38] L. Ouyang, J. Wu, X. Jiang, and et al., “Training language models to follow instructions with human feedback,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 730–27 744, 2022.

[39] X. Qiu, T. Sun, Y. Xu, and et al., “Pre-trained models for natural language processing: A survey,” *Sci. China Technol. Sci.*, vol. 63, no. 10, p. 1872–1897, 2020.

[40] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: pre-training of deep bidirectional transformers for language understanding,” in *NAACL-HLT*, 2019, pp. 4171–4186.

[41] C. Raffel, N. Shazeer, A. Roberts, and et al., “Exploring the limits of transfer learning with a unified text-to-text transformer,” *J. Mach. Learn. Res.*, vol. 21, pp. 140:1–140:67, 2020.

[42] M. Lewis, Y. Liu, N. Goyal, and et al., “BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension,” in *ACL*, 2020, pp. 7871–7880.

[43] F. Li and H. V. Jagadish, “Nalir: an interactive natural language interface for querying relational databases,” in *SIGMOD*, 2014, pp. 709–712.

[44] F. Li and H. V. Jagadish, “Constructing an interactive natural language interface for relational databases,” *Proc. VLDB Endow.*, vol. 8, no. 1, pp. 73–84, 2014.

[45] D. Saha, A. Floratou, K. Sankaranarayanan, and et al., “ATHENA: an ontology-driven system for natural language querying over relational data stores,” *Proc. VLDB Endow.*, vol. 9, no. 12, pp. 1209–1220, 2016.

[46] X. Wang, M. Liu, J. Xu, and H. Lu, “NALMO: transforming queries in natural language for moving objects databases,” *GeoInformatica*, vol. 27, no. 3, pp. 427–460, 2023.

[47] J. Li, W. Wang, W. Ku, Y. Tian, and H. Wang, “Spatialnli: A spatial domain natural language interface to databases using spatial comprehension,” in *SIGSPATIAL/GIS*, 2019, pp. 339–348.

[48] J. Guo, Z. Zhan, Y. Gao, Y. Xiao, J. Lou, T. Liu, and D. Zhang, “Towards complex text-to-sql in cross-domain database with intermediate representation,” in *ACL*, 2019, pp. 4524–4535.

[49] U. Brunner and K. Stockinger, “Valuenet: A natural language-to-sql system that learns from database information,” in *ICDE*, 2021, pp. 2177–2182.

[50] S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer, “Learning a neural semantic parser from user feedback,” in *ACL*, 2017, pp. 963–973.

[51] P. Tong, Q. Zhang, and J. Yao, “Leveraging domain context for question answering over knowledge graph,” *Data Sci. Eng.*, vol. 4, no. 4, pp. 323–335, 2019.

[52] K. Affolter, K. Stockinger, and A. Bernstein, “A comparative survey of recent natural language interfaces for databases,” *VLDB J.*, vol. 28, no. 5, pp. 793–819, 2019.

[53] H. Liu, T. Zhang, F. Li, M. Yu, and G. Yu, “A probabilistic generative model for tracking multi-knowledge concept mastery probability,” *Frontiers Comput. Sci.*, vol. 18, no. 3, p. 183602, 2024.

[54] J. Sen, C. Lei, A. Quamar, and et al., “ATHENA++: natural language querying for complex nested SQL queries,” *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 2747–2759, 2020.

[55] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, “Text-to-sql empowered by large language models: A benchmark evaluation,” *Proc. VLDB Endow.*, vol. 17, no. 5, pp. 1132–1145, 2024.

[56] S. Sun, Y. Zhang, J. Yan, Y. Gao, D. Ong, B. Chen, and J. Su, “Battle of the large language models: Dolly vs llama vs vicuna vs guanaco vs bard vs chatgpt - A text-to-sql parsing comparison,” in *EMNLP*, 2023, pp. 11 225–11 238.