软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(3):1121−1142 [doi: 10.13328/j.cnki.jos.007514] [CSTR: 32375.14.jos.007514]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于大语言模型的空间数据库自然语言查询转换方法*

刘孟怡[1], 许建秋[1], 童咏昕[2]

[1](南京航空航天大学 计算机科学与技术学院/软件学院, 江苏 南京 211106)

[2](北京航空航天大学 计算机学院, 北京 100191)

通信作者: 许建秋, E-mail: jianqiu@nuaa.edu.cn

摘　要: Text2SQL 技术通过减少非专业用户与关系数据库交互的技术障碍, 已发展为数据分析和数据库管理的重要工具. 以 GPT 为代表的大语言模型 (large language model, LLM) 的引入, 进一步提升了 Text2SQL 系统的性能. 然而, 由于空间数据涉及复杂的几何关系、多样化的查询类型和对高精度语义理解的需求, 现有的 Text2SQL 技术难以直接适用于空间数据库领域. 为了解决上述问题, 降低普通用户与空间数据库的交互门槛, 提出了面向空间数据库的自然语言查询 (natural language query, NLQ) 转换方法. 该方法有两个核心阶段: (1) 自然语言理解; (2) 可执行语言生成. 在阶段 (1) 中使用实体信息提取算法提取关键查询实体, 并基于大语言模型构建空间数据查询语料库进而确定查询类型. 在阶段 (2) 中根据查询类型选择结构化语言模型 (structured language model, SLM), 然后将实体映射到结构化语言模型中, 得到最终的空间数据库可执行语言. 在多组真实数据集上的实验结果表明, 该方法可以实现从用户的自然语言查询到空间数据库可执行语言的高效转换.

关键词: 空间数据库; 自然语言接口; 数据库自然语言接口; 语义解析; 查询处理

中图法分类号: TP311

中文引用格式: 刘孟怡, 许建秋, 童咏昕. 基于大语言模型的空间数据库自然语言查询转换方法. 软件学报, 2026, 37(3): 1121–1142. http://www.jos.org.cn/1000-9825/7514.htm

英文引用格式: Liu MY, Xu JQ, Tong YX. Natural Language Query Transformation Method for Spatial Databases Based on Large Language Models. Ruan Jian Xue Bao/Journal of Software, 2026, 37(3): 1121–1142 (in Chinese). http://www.jos.org.cn/1000-9825/7514.htm

Natural Language Query Transformation Method for Spatial Databases Based on Large Language Models

LIU Meng-Yi[1], XU Jian-Qiu[1], TONG Yong-Xin[2]

[1](College of Computer Science and Technology/College of Software, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China)

[2](School of Computer Science and Engineering, Beihang University, Beijing 100191, China)

Abstract: Text2SQL has evolved into a significant tool for data analysis and database management by reducing the technical barriers for non-expert users to interact with relational databases. The introduction of large language models (LLMs), represented by GPT, further improves the performance of Text2SQL systems. However, existing Text2SQL techniques are difficult to apply directly to the spatial database domain because spatial data involves complex geometric relationships, diverse query types, and the demand for high-precision semantic understanding. To address these issues and lower the threshold for interaction between non-experts and spatial databases, a natural language query (NLQ) transformation method for spatial databases is proposed. The method consists of two core phases: (1) natural language understanding; (2) executable language generation. In phase (1), an entity information extraction algorithm is employed to extract

* 基金项目: 国家自然科学基金 (62472217, U23A20296)
本文由“向量数据库及 DB4LLM 技术”专题特约编辑高宏教授、李国良教授、张蓉教授推荐.
收稿时间: 2025-05-06; 修改时间: 2025-06-30; 采用时间: 2025-08-20; jos 在线出版时间: 2025-09-02
CNKI 网络首发时间: 2026-01-08

key query entities, and a spatial data query corpus is constructed based on large language models to determine the query type. In phase (2), a structured language model (SLM) is selected according to the query type, and the entities are then mapped into the structured language model to generate the final executable language for spatial databases. Experimental results on multiple real-world datasets demonstrate that the proposed method enables efficient transformation from natural language queries to executable languages of spatial databases.

Key words: spatial database; natural language interface; natural language interface for databases; semantic parsing; query processing

空间数据库为地图服务、物流管理和城市规划提供了重要的地理信息数据管理和分析支持. 地理信息技术的飞速发展和广泛应用, 不仅使空间数据的获取变得可行且具有成本效益, 而且推动了对高效处理空间数据的需求[1]. 近年来, 诸如 GeoSpark[2]、Ganos[3]和 SECONDO[4]等空间数据库系统相继涌现, 这些系统能够支持多种空间数据查询, 包括基础空间查询、范围查询、最近邻居查询、空间 Join 查询和聚合查询. 然而, 空间数据查询的高技术门槛与应用用户日益增长的需求之间存在显著鸿沟. 一方面, 空间查询通常需要构造高度技术化、结构明确的查询语句, 这对用户的专业知识提出了较高要求. 另一方面, 随着空间数据的快速积累与广泛传播, 非专家用户逐渐成为空间数据库的主要用户群体, 他们缺乏深入的领域知识, 难以有效地利用空间数据库的潜力. 为了使非技术用户能够通过自然语言直接查询空间数据库, 需要一个数据库自然语言接口 (natural language interface for database, NLIDB). 虽然研究人员对空间数据库进行了大量的研究, 包括数据分区[5]、索引结构[6]和空间众包[7], 但对空间数据库自然语言接口的研究显著不足.

尽管 NLIDB 在关系数据库、XML 数据库和 RDF 问答系统等领域已有广泛研究, 但针对空间数据库的研究仍处于起步阶段. 由于空间数据查询的独特复杂性, 尤其是在涉及最近邻居搜索、空间聚合等高级任务时, 传统的 Text2SQL 框架难以直接应用于空间数据库可执行语言的生成. GPT-4o 等大语言模型 (large language model, LLM) 为空间数据库自然语言接口提供了新思路. GPT-4o 支持空间数据上的自然语言查询 (natural language query, NLQ), 并生成语法上正确的可执行语言, 但这些查询通常在语义上是不准确的.

NLQ_1: What universities are there in Jiangning District of Nanjing?

以 NLQ_1 为例, 转换过程如图 1 所示, GPT-4o 错误地提取实体 “Jiangning District”, 使用 ininterior 运算符来判断大学是否完全位于江宁区里, 但实际上应该使用 intersects 运算符进行判断. 同时, GPT-4o 生成的可执行语言没有考虑查询优化.

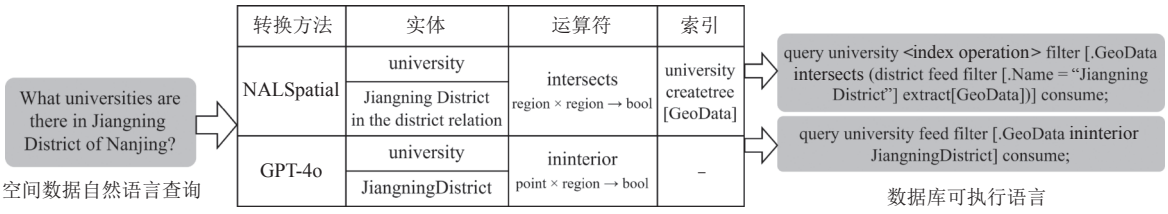

图 1 空间数据库自然语言查询转换示例

构建空间数据库的自然语言接口需要解决以下两大关键挑战.

挑战 1. 空间关系和位置描述涉及复杂的几何和拓扑概念, 用户的查询可能以多种方式表达, 查询语义难以理解. 如果不能准确理解用户的意图, 将导致查询结果的偏差甚至完全错误. 现有的一些自然语言处理 (natural language processing, NLP) 工具, 如 Stanford CoreNLP[8]、spaCy[9]、NLTK[10]和 Stanza[11], 在解析常规文本时表现出色. 然而, 由于空间数据固有的复杂几何关系和多样化查询类型, 这些工具在准确解析空间数据查询方面的效果并不理想. 最近, 大语言模型在空间数据自然语言查询的语义解析方面展现了巨大潜力[12], 然而缺乏专门针对空间数据查询的高质量训练语料库. 这种数据稀缺性导致模型在生成可执行语言时需要依赖提示工程, 不仅降低了转换精度, 还可能增加开发成本. 虽然微调模型可以部分解决这一问题, 但设计和创建高质量的微调语料库往往需要大量的人力和资源投入, 在实际应用中难以广泛推广.

挑战 2. 可执行查询语句需遵循特定的系统级规则, 如何选择合适的数据表和操作符仍是一个尚未解决的问

题[13]. SQL 在处理包含多维数据的复杂空间查询时效率不高. 现有的 NLIDB 系统通常依赖查询优化器在将自然语言转换为 SQL 后生成最终查询结果. 这种方式虽然简化了用户的操作, 但在查询优化和性能控制方面存在不足. 为了更好地支持空间数据查询, 有必要研究如何将自然语言直接转换为空间数据库的可执行语言. 可执行语言能够支持空间数据运算符的选择和组合, 从而为查询优化提供更多控制能力. 此外, 通过显式生成查询计划, 用户可以更深入地理解查询的执行过程, 并在必要时调整数据操作步骤, 改善查询性能.

为了解决上述挑战, 本文提出一种基于大语言模型的空间数据库自然语言查询转换方法, 命名为 NALSpatial, 允许用户直接使用自然语言进行空间数据库查询. NALSpatial 以其独特的两阶段方法兼具可扩展性和通用性, 保证了物理布局与语义解耦. 在自然语言理解阶段, 通过应用知识库和基于大语言模型构建的语料库, 提取出关键实体和查询类型. 这种设计使 NALSpatial 具备可扩展性, 适应各种空间数据类型和查询操作, 为不断演进的数据需求提供了灵活且可持续的解决方案. 在可执行语言生成阶段, 根据查询类型选择结构化语言模型 (structured language model, SLM), 基于实体映射规则构造数据库可执行查询. 通过动态调整结构化语言模型, 可以实现对底层数据库的高度适应性. 这种设计确保了 NALSpatial 的通用性, 使其能够无缝集成并运行于任一空间数据库之上, 无需过多的修改. 此外, NALSpatial 不仅是轻量级的, 对于用户查询的语义限制较小, 还是模式弱相关的, 无须为不同的数据库重新训练模型, 部署代价小.

综上所述, 本文的贡献可以总结如下.

(1) 对用户输入的自然语言查询进行深入解析, 提取关键的空间实体信息并准确识别查询类型. 首先设计空间数据查询实体提取算法, 利用自然语言处理技术的分词和命名实体识别功能, 结合空间关系和地点知识库, 对查询中的空间实体信息进行全面解析. 然后, 基于大语言模型构建涵盖多种空间查询类型的高质量语料库, 使用 BiLSTM 模型进行训练, 从而实现查询类型的识别.

(2) 通过预定义的结构化语言模型与映射规则, 将自然语言的查询语义准确转化为数据库可执行语言. 首先为基础空间查询、范围查询、最近邻居查询、空间 Join 查询和聚合查询分别构造结构化语言模型, 然后基于映射规则将空间数据查询实体中的每个元素映射到模型实体槽中对应的元素, 进而生成可执行语言. 最后结合空间索引和查询计划优化技术, 对生成的查询语句进行优化.

(3) 在数据库系统 SECONDO 中开发了 NALSpatial, 并基于真实的地理信息数据集进行了全面的实验验证. 实验结果表明, NALSpatial 的平均响应时间约为 2.5 s, 可翻译性为 95%, 翻译精度为 92%. 综合考虑评估指标, NALSpatial 优于基于传统规则或机器学习的 3 种方法和基于大语言模型 GPT-4o 的方法. 此外, 验证了 NALSpatial 的性能稳定性及其各个模块的有效性.

1 相关工作

Text2SQL 技术已经取得了显著进展, 以应对自然语言处理和结构化语言生成方面的重要挑战. 为了解决这些问题, 研究人员提出了 3 种方法: 基于规则的方法、基于机器学习的方法、混合的方法.

基于规则的方法需要明确的规则, 以便将自然语言翻译成结构化语言, 但会限制用户提出的自然语言查询的语义表示. 代表系统是 PRECISE[14]和 NaLIR[15]. PRECISE 定义了语义易处理性的概念, 以识别可以准确翻译成 SQL 的自然语言查询. 但是, 无法进行语义处理的自然语言查询将被 PRECISE 拒绝转换. PRECISE 使用推理规则将已解析的查询映射到数据库元素. 然后, 将生成的结构化语言片段组合在一起得到最终的结构化语言. NaLIR 使用 NLP 工具生成自然语言查询的解析树, 然后标识其中可以映射到 SQL 组件的节点, 并将语义覆盖表示为解析树的一个子集. NaLIR 不仅向用户解释了查询的处理方式, 而且还呈现多种理解方式供用户选择, 以减轻用户消除歧义的负担. 当查询超出语义范围时, NaLIR 可以为用户生成构造查询的建议.

基于机器学习的方法可以直接学习自然语言到语义表示的映射, 但该方法的性能在很大程度上取决于训练数据的质量[16]. 代表系统是 IRNet[17]和 ValueNet[18]. IRNet 在自然语言查询和数据库模式之间执行模式链接, 目标是识别 NLQ 中提到的列和表. 然后使用基于语法的神经模型来合成树形结构的中间查询语言, 并从中推断出 SQL.

ValueNet 使用基于编码器-解码器架构的神经模型来合成结构化语言. 编码器是 IRNet 编码器的扩展, 不仅接收有关数据库模式的信息, 还接收从数据库内容中提取的候选值. 解码器由 LSTM 架构和多个指针网络组成, 用于选择表、列和值. 最近, 大语言模型为基于机器学习的方法带来了全新的发展机遇. Kordjamshidi[19]利用大语言模型在自然语言理解阶段进行空间和时间推理, 构建了能够捕获关键实体及其活动的空间和时间信息的架构.

基于规则的方法使用广泛, 但只能解析特定领域的自然语言问题. 基于机器学习的方法支持跨知识库或跨语言的系统, 但高度依赖训练数据. 基于规则和机器学习混合的方法将规则和机器学习技术结合起来, 以充分利用它们的优势, 从而提高系统对自然语言查询的理解和处理能力. 代表系统是 NALMO[20]和 GAR[21]. NALMO 是一个用于移动对象的自然语言接口. 理解自然语言查询时, NALMO 先对它进行预处理, 然后使用语义解析算法得到包括最近邻居数、时间、查询对象在内的实体信息. 最后, 基于数据库生成一个位置知识库来确定位置信息. 同时为了提高查询效率, 系统构建了一个基于位置的前缀索引来匹配位置知识库. NALMO 支持 5 种移动对象查询, 包括时间间隔查询、范围查询、最近邻居查询、轨迹相似性查询和 Join 查询. GAR 从给定数据库上的一组示例 SQL 开始, 进行数据准备. 对于给定的 NLQ, GAR 查找在数据准备过程中生成的 NLQ, 并使用学习排序模型找到最接近的表达式, 从而获得转换结果.

大多数 Text2SQL 技术是为关系数据库开发的, 依赖关系型数据的结构化查询模式. 然而, 空间数据不仅具有高度的多样性, 还涉及复杂的空间拓扑关系, 以及适用于空间数据库的训练数据集的有限性, 使得现有的 Text2SQL 技术难以直接适用于空间数据库. 此外, 空间数据库的查询往往涉及复杂的空间算子, 表达方式与传统 SQL 查询存在很大差异. 因此, 如何有效地支持空间数据库的自然语言查询, 准确表达空间关系、提高查询的可解释性与可用性, 仍然是当前数据库自然语言查询转换研究中的关键挑战.

2 基于大语言模型的空间数据库自然语言查询转换方法

本节详细介绍所提出的基于大语言模型的空间数据库自然语言查询转换方法 NALSpatial, 图 2 是对 NALSpatial 的整体框架展示. 在自然语言理解阶段, 先对输入的查询进行自然语言处理, 然后结合预先构建的知识库提取关键实体, 利用基于大语言模型构建的语料库识别查询类型. 在可执行语言生成阶段, 根据查询类型选择结构化语言模型, 基于实体映射规则构造可执行语句.

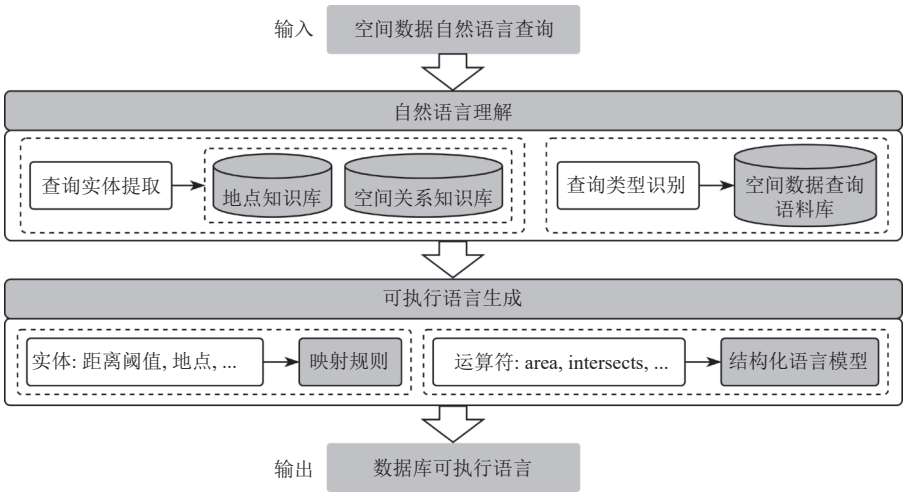

图 2 NALSpatial 框架

形式化 NALSpatial, 如算法 1 所示. 在自然语言理解过程中, 先根据空间数据库构建空间关系知识库和地点知识库 (第 1 行), 利用 NLP 工具 spaCy 对输入的查询进行处理 (第 2 行), 结合知识库提取关键实体 (第 3、4 行). 然

后基于大语言模型 GPT-4o 构建高质量的空间数据查询语料库 (第 5 行), 使用 BiLSTM 网络训练语料库, 对查询的类型进行预测 (第 6、7 行). 在可执行语言生成过程中, 先利用数据库中的运算符和查询语法为所有类型的查询生成结构化语言模型 (第 8 行), 根据已识别的类型确定合适的结构化语言模型 (第 9 行), 然后基于映射规则将实体映射到结构化语言模型中, 构造可执行的数据库查询并返回 (第 10、11 行).

算法 1. 空间数据自然语言查询转换.

输入: 自然语言查询 *NLQ*; 数据库信息 *DB*;
输出: 可执行的数据库查询 *EDQ*.

1. $KBs \leftarrow generateKB(DB)$
2. $doc \leftarrow nlp(NLQ)$
3. **for** $word_i \in doc$ **do** // 实体提取
4. $\quad E \leftarrow entityExtraction(KBs)$
5. $Corpus \leftarrow generateCorpus()$
6. $model \leftarrow BiLSTM\ (Corpus)$ // 查询类型识别
7. $QType \leftarrow predictQueryType(model, doc)$
8. $SLM \leftarrow generateSLM(DB)$
9. $slm \leftarrow selectSLM(SLM, QType)$
10. $EDQ \leftarrow constructEDQ(E, slm)$ // 查询映射
11. **return** *EDQ*

3 空间数据自然语言查询理解

自然语言理解阶段的任务是对输入的空间数据 NLQ 进行解析, 创新性工作包括实体信息提取算法和空间数据查询语料库.

(1) 实体信息提取算法的创新点在于采用双层候选实体生成策略, 先提取数字列表和信息列表, 确保涵盖不同类型的查询要素, 然后结合空间关系知识库和地点知识库进行筛选与剪枝, 从而提升实体识别的准确性和可解释性. 相比于传统方法, 该算法不仅提取查询地点, 还能识别最近邻居数、距离阈值和空间关系, 适用于多种复杂的空间查询类型, 增强了系统的泛化能力和对空间语义的理解能力.

(2) 空间数据查询语料库的创新点体现在基于领域论文抽取真实查询, 确保语料的专业性, 并结合 GPT-4o 进行数据增强, 通过关键数据实体替换、句法结构变化和语言风格调整等方式生成变体, 提升查询的多样性和覆盖范围. 通过这种方法, 语料库得以扩展至 3 500 条高质量查询, 相比传统的手工构造或规则生成方法, 能够提高系统在不同查询场景下的适应性和鲁棒性.

3.1 相关定义

定义 1 (空间数据查询实体). 在空间数据自然语言查询中, 关键实体是一个元组 $E = (k, d, relation, location)$, 其中, k 存储最近邻居数, d 存储距离阈值, *relation* 是一个存储空间关系的字典数组, *location* 是一个存储查询地点的字符串数组.

定义 2 (最近邻居数). 令 k 是最近邻居数, *word_NN* 是集合{“nearest”, “closest”, “neighbor”}中的一个英文单词. 如果自然语言查询不包含 *word_NN* 中任一单词, 则 $k = 0$. 反之, 令 K 是自然语言查询中的基数集合, 则 k 按照如下 3 种情况取值.

(1) 如果 $|K| = 0$, $k = 1$.

(2) 如果 $|K| = 1$, $k = k_1 \in K$.

(3) 如果 $|K| > 1$, 令 $D(word_1, word_2)$ 是 $word_1$ 和 $word_2$ 相隔的单词数, 则 $k = n \in K$, 对于 $\forall k_i \in K$, n 满足:

$$D(n, word\_NN) = \min_{1 \leqslant i \leqslant |K|} D(k_i, word\_NN) \tag{1}$$

NLQ_2: List the 12 closest parks to the border of Nanjing.

NLQ_3: List the 12 closest parks to the road 3.

以 NLQ_2 为例, 它包含 “closest”, $K = \{12\}$. 由定义 2 可得 $k = 12$. 以 NLQ_3 为例, 它包含 “closest”, $K = \{12, 3\}$. 对于 K 中所有基数词来说, 12 和单词 “closest” 之间的距离是最小的. 由定义 2 可得 $k = 12$.

定义 3 (距离阈值). 给定短语 $D = (N, U)$, N 是一个数字, U 表示距离单位 (如 kilometer、meter), 每个单位 U 对应一个到 meter 的转换因子 f_U, 则距离阈值 $d = N \times f_U$.

$$f_U = \begin{cases} 1, & U = \text{meter} \\ 1000, & U = \text{kilometer} \end{cases} \tag{2}$$

NLQ_4: Please provide a list of POIs within 2 kilometers of each district in Nanjing.

以 NLQ_4 为例, $D = (2, \text{kilometer})$, 由定义 3 可得 $d = 2 \times 1000 = 2000$.

定义 4 (空间关系). 令 *relation* 是存储空间关系的字典数组, $relation = \{R_1, R_2, \ldots, R_n\}$, 则空间关系 R_i 的完整结构为:

$$R_i = \{\text{“ID”}: id_i, \text{“Name”}: name_i, \text{“Attribute”}: attribute_i\} \tag{3}$$

其中, ID 属性表示空间关系的唯一标识符, Name 属性表示空间关系的名称, Attribute 属性用于描述空间关系的几何类型, $attribute_i \in \{\text{point}, \text{line}, \text{region}\}$.

定义 5 (查询地点). 令 *location* 是存储查询地点的字符串数组, $location = \{L_1, L_2, \ldots, L_m\}$, 其中 L_i 是有效地理位置, 如城市、区域、建筑物等.

3.2 查询实体提取

在提取实体信息之前, 基于空间数据库构造空间关系和地点知识库. 地点知识库存储地理空间实体的信息, 而空间关系知识库则用于记录空间数据库中的表信息. 地点知识库的主要功能是完整表达每个地点的语义信息, 因此需要包含地点的名称、类型以及其存储方式. 地点的存储方式可能是几何对象, 如区域 (region)、线 (line) 或点 (point), 也可能是某个空间关系表中的记录. 空间关系知识库存储空间数据库中所有空间关系表的信息, 包括唯一标识符、名称和空间属性. 地点知识库中的地点如果是通过某一关系表存储的, 那么地点知识库中的该条记录会引用空间关系知识库中的表标识符. 将地点和空间关系的信息分别存储为 CSV 文件, 便构成了地点知识库和空间关系知识库. 为了提升知识库的搜索效率, 采用哈希表结构进行管理. 使用链表存储冲突的元素, 并使用动态调整容量的策略来提高性能和内存利用率. 给定元素数量 *size* 和哈希表容量 *capacity*, 负载因子被定义为 $load\_factor = size/capacity$. 当负载因子 *load_factor* 超过阈值或过低时, 扩展或缩小哈希表容量并重新哈希所有元素.

空间数据自然语言查询中的关键实体信息提取如算法 2 所示. 首先, 对输入的 NLQ 进行初步处理, 利用 NLP 工具识别句子中潜在的关键信息, 并将如 “10-minute walk”“nearby” 等模糊自然语言表达转换为明确的空间距离阈值. NLP 工具需满足以下条件: 高效的处理速度、广泛的功能支持、易于集成的开发环境. 基于这些考虑, spaCy 是优先选择, 主要因为其在 Python 环境中的优秀性能及丰富的功能集. spaCy 不仅支持高效的分词和命名实体识别, 还能够对复杂的语法结构进行解析, 特别适合处理多样化的自然语言查询. 为了提高系统的鲁棒性, NLTK 作为替代方案. 利用 spaCy 对输入的 NLQ 进行处理, 生成两类候选实体的集合.

算法 2. 实体信息提取算法.

输入: 自然语言查询 *Q*; 地点知识库 *LKB*; 空间关系知识库 *SRKB*;
输出: 空间数据查询实体 *E*.

1. $doc \leftarrow nlp(Q)$
2. **for** $word_i \in doc$ **do**

```
3.  if word_i 是最近邻居数 then // 基于定义 2 进行判断
4.      E.k ← word_i // 提取最近邻居数
5.  if word_i 是距离阈值 then // 基于定义 3 进行判断
6.      E.d ← get_threshold(word_i) // 提取距离阈值
7.  if word_i.pos=NOUN 或者 word_i.pos=PROPN then
8.      noun_list.append(word_i.text)
9. for noun_i∈noun_list do
10.  if search(noun_i, SRKB.name) then
11.      E.relation.append(noun_i) // 提取空间关系
12.  if search(noun_i, LKB.name) then
13.      E.location.append(noun_i) // 提取查询地点
14. return E
```

(1) 数字列表 $N\_L=\{n_1,n_2,\ldots,n_k\}$, 存储被识别为数字、基数或数量的实体. 数字实体通常与查询中的数量或距离相关, 代表查询条件中重要的参数.

(2) 信息列表 $I\_L=\{i_1,i_2,\ldots,i_m\}$, 包含词性为名词或专有名词的单词. 这些词通常代表空间实体或地点名称等信息, 是查询的核心空间实体.

以 NLQ_2 为例, N_L 中包含 “12”, I_L 中包含 “parks”“border”“Nanjing”.

通过细粒度实体提取对候选实体进行剪枝, 获得实体 E=(k, d, *relation*, *location*). 具体来说, 对于列表 N_L 中的每个数字, 根据定义 2 进行判断, 如果一个数字在 NLQ 中与关键字 “nearest”, “closest” 或 “neighbor” 在语义距离上很近, 则将该数字识别为最近邻居数 k. 例如, 在 NLQ_2 中, “12” 紧邻 “closest”, 因此被提取为查询中的最近邻居数. 距离阈值用于限制查询范围, 以确保查询结果符合用户的期望. 该阈值的提取过程依赖于列表 N_L 中的数量短语, 这些短语通常由数字和距离单位 (如 meter、kilometer 等) 组成. 例如, 在 NLQ_4 中, 短语 “2 kilometers” 被解析为距离阈值, 通过将单位转换为 meter, 系统可以获得精确的距离阈值 d. 最后, 通过判断列表 I_L 中的单词是否位于知识库中来确定空间关系 *relation* 和查询地点 *location*. 例如, 在 NLQ_2 中, “parks” 被识别为空间关系, “Nanjing” 被识别为地点.

● 算法 2 的时间复杂度分析. 假设一个 NLQ 中包含的单词数为 *len*, 地点知识库中包含的地点数为 m, 空间关系知识库中包含的关系数为 n, 可以推断 m 大于 n (因为一个空间关系至少包含一个地点信息). 利用哈希表查找知识库的时间复杂度为 O(1), 因此算法 2 的时间复杂度为 O(*len*).

3.3 查询类型识别

空间查询的种类繁多, 这种多样性使得为每种查询设计通用的翻译模板难以实现. 因此, 准确识别查询类型成为生成可执行语言的前提和重要步骤. 针对这一需求, 先利用大语言模型构建涵盖丰富空间查询类型的高质量语料库, 确保语料库内容在表达多样性、语法一致性和语义清晰度等方面的高标准. 同时, 通过对比 TextCNN、LSTM、BiLSTM、DistilBERT 和 BERT 的基本原理以及在实际场景中的应用, 从而确定最适合训练语料库的模型为 BiLSTM. 虽然此方法增加了模型设计的成本与复杂性, 但显著提升了自然语言查询到可执行语言转换的精度.

语料库包含 3 500 条自然语言查询, 覆盖了 5 种常见的空间数据查询类型: 基础空间查询、范围查询、最近邻居查询、空间 Join 查询和聚合查询. 每条查询都严格标注了类型, 保证了语料库在训练任务中的分类准确性. 这些查询是从相关领域的论文中抽取出来并进行数据增强得到的. 语料库的构建过程如下.

● 查询提取. 先从空间数据处理和自然语言查询相关领域的文献中抽取 60 条高质量的空间数据自然语言查询. 查询的选择过程涉及多层筛选和领域专家的指导, 以保证每条查询都能够涵盖空间数据库的典型需求. 在选择过程中, 领域专家检查了查询在语义上是否符合自然语言查询的表达方式, 以及是否能够被转化为有效的空间数

据库查询.

● 数据增强. 利用 GPT-4o 通过替换查询中的关键数据实体、改变语句结构或使用不同的语言风格来进行数据增强, 将语料库扩展至 3 500 条查询. 表 1 展示了部分数据增强示例. GPT-4o 能够快速生成多样化的查询, 显著加快了语料库的构建速度. 其次, GPT-4o 生成的查询在语法和语言风格上具有一致性, 确保了语料库的统一性和专业性. 在此步骤中, 制定了多维度的增强策略, 具体包含如下几个方面.

NLQ_5: Calculate the distance from Mount Tai to Mount Huang.

表 1 数据增强示例

查询类型	原始查询	数据增强示例
基础空间查询	Calculate the distance from Mount Huang to Mount Tai.	What's the distance from Mount Huang to Mount Tai?
范围查询	What restaurants are located in Jiangning District?	Could you provide a list of restaurants located in Jiangning District?
最近邻居查询	Find the two nearest lakes to West Lake.	Find the five nearest lakes to West Lake.
空间Join查询	What restaurants are within 2 kilometers of each subway station?	What parks are within 2 kilometers of each subway station?
聚合查询	How many amusement parks are there in Nanjing?	Please provide the number of amusement parks in Nanjing.

NLQ_6: Find the two nearest lakes to West Lake.

(1) 数据实体替换: GPT-4o 通过替换查询中的关键实体, 如地理位置和空间对象, 生成多种查询变体. 例如, 对于 NLQ_5, 将 "Mount Tai" 和 "Mount Huang" 替换为其他地理位置, 可创建不同的查询.

(2) 语言结构转换: 利用 GPT-4o 生成表达同一含义的多样化语言结构, 从而增强语料库的语言多样性. 例如, 将 NLQ_5 转化为 "What's the distance from Mount Tai to Mount Huang?", 可以保证查询在不同语言风格下的表达一致性和连贯性. 语言结构转换能够提升模型对自然语言表达的鲁棒性.

(3) 查询属性的数量参数调整: 对于数量相关的查询属性, 如最近邻居数和距离阈值, 通过调整参数来生成多种样例. 例如 NLQ_6 可通过调整数量来生成新的查询, 如 "Find the five nearest lakes to West Lake.".

● 查询检查. 在语料库构建的最终阶段, 为确保 GPT-4o 生成的空间数据查询能够有效支持自然语言到数据库查询的转化过程, 挑选了 3 位具备丰富经验的专家对查询进行严格检查. 这些专家均熟悉自然语言查询的构造原则、查询类型的分类标准以及空间数据可执行语句的设计规范. 因此, 他们不仅能够确保生成的查询在学术和技术上的高质量, 还能够对查询在实际应用场景中的可操作性做出专业判断. 通过严格的人工检查和校对, 最终构建的语料库具备高质量和广泛适用的特点, 不仅可以用于空间数据查询类型识别模型的训练, 还能作为微调模型或领域适配的关键资源, 从而帮助大语言模型理解空间语义.

在训练空间数据查询语料库时, 任务特点是小样本、多类别、短文本分类. 这类任务对模型的准确性和训练效率提出了很高的要求, 以适应语料库的持续扩展与更新. TextCNN 结构简单、训练速度快, 适合短文本分类任务, 但基于卷积的局部特征提取机制难以建模查询语句中的上下文依赖, 在语义理解方面存在明显局限. LSTM 通过引入记忆单元, 有效捕捉语句中的时序信息, 弥补了 TextCNN 的不足. BiLSTM 在 LSTM 基础上进一步引入双向建模能力, 可同时捕捉前后语义依赖, 显著提升了对查询语句语法结构与空间语义关系的理解能力, 在语料库持续更新场景中具有很强的泛化能力与适应性. 虽然 BERT 在自然语言处理任务中的表现十分强大, 但庞大的参数量导致训练时间显著增加, 不适合小样本任务. DistilBERT 作为轻量级预训练模型, 具备较强的语义建模能力, 但参数规模依然较大, 训练时间显著高于传统模型, 不利于语料库持续更新下的快速训练. 理论上, BiLSTM 在保持较强上下文理解能力的同时具备良好的训练效率, 特别适合应对空间数据查询语料库持续扩展与动态更新的需求, 是本任务的优选模型结构.

基于查询实体的提取结果, 当一个 NLQ 中包含超过两个空间关系, 且被识别为空间 Join 查询时, 系统将调用 GPT-4o 来判断各对空间关系之间的连接逻辑是基于空间拓扑 (如包含、相交) 还是基于距离约束 (如 "within 1.5 kilometers""within a 15-minute walk"), 并将生成的连接信息作为输入传递至可执行语言生成阶段.

4 空间数据自然语言查询的可执行语言生成

可执行语言生成阶段的任务是基于关键语义信息构建可执行语言, 创新性工作是结构化语言模型的设计. 首先, 自然语言理解阶段确定的查询类型可以精准指导关键实体和运算符的选择. 其次, 创新性地设计了实体槽位机制, 将待定实体值与关键实体一一对应, 形成对查询上下文的结构化抽象, 这种方法使得自然语言中的复杂语义能够转化为清晰的逻辑结构, 确保了结构化语言模型的可解释性和易操作性. 最后, 基于数据库可执行语言的语法规则进行组合, 生成结构清晰且逻辑明确的查询表达式.

4.1 相关定义

令 $\text{Type}(p) \in \{\text{point}, \text{line}, \text{region}\}$ 表示地点 p 的数据类型, 函数 $compLoc$ 判断地点 p 和 q 是否有公共部分:

$$compLoc(p,q) = \begin{cases} \text{intersects}(p,q), & \text{Type}(p) \in \{\text{line}, \text{region}\} \wedge \text{Type}(q) \in \{\text{line}, \text{region}\} \\ \text{within}(p,q), & \text{Type}(p) = \text{point} \wedge \text{Type}(q) = \text{region} \end{cases} \tag{4}$$

定义 6 (空间范围查询). 给定一个空间对象集合 R 和一个查询地点 p, 空间范围查询返回满足特定空间谓词条件的所有对象, 用集合 R' 表示:

$$R' = \{r | r \in R \wedge compLoc(r,p)\} \tag{5}$$

定义 7 (最近邻居查询). 给定一个空间对象集合 R, 一个查询地点 p 和一个正整数 k, 最近邻居查询返回集合 $R' = \{r'_1, \ldots, r'_k\} \subseteq R$, 对于 $\forall r \in R - R'$, 满足:

$$dist(r,p) \geqslant \max_{1 \leqslant i \leqslant k} dist(r'_i, p) \tag{6}$$

定义 8 (空间 Join 查询). 根据连接条件将空间 Join 查询分为两类.

(1) 给定两个空间对象集合 R 和 S, 空间 Join 查询返回集合 T:

$$T = \{(p,q) | p \in R \wedge q \in S \wedge compLoc(p,q)\} \tag{7}$$

(2) 给定两个空间对象集合 R 和 S, 一个距离阈值 d, 距离 Join 查询返回集合 T:

$$T = \{(p,q) | p \in R \wedge q \in S \wedge dist(p,q) \leqslant d\} \tag{8}$$

定义 9 (空间聚合查询). 根据统计函数将空间聚合查询分为 3 类.

(1) 给定一个空间对象集合 R 和一个查询地点 p, 统计数量的空间聚合查询返回一个整数 $n = |R'|$, $R' = \{r | r \in R \wedge compLoc(r,p)\}$. 对两个空间对象集合 R 和 S 进行数量统计的空间聚合查询的结果用集合 R' 表示:

$$R' = \{r' | r' \in R \wedge r'.Cnt = |S'|,\ S' = \{s | s \in S \wedge compLoc(s,r')\}\} \tag{9}$$

(2) 给定一个空间对象集合 $R = \{r_1, \ldots, r_n\}$ 和一个查询地点 p, 统计总和的空间聚合查询返回一个浮点数 f:

$$f = \sum_{i=1}^{n} \text{area}(\text{intersection}(r_i, p)) \tag{10}$$

(3) 给定两个空间对象集合 R 和 S, 统计最大值的空间聚合查询返回一个对象 $p \in R'$, 其中, $R' = \{r' | r' \in R \wedge r'.Cnt = |S'|, S' = \{s | s \in S \wedge compLoc(s,r')\}\}$, 满足:

$$p.Cnt = \max_{1 \leqslant i \leqslant |R|} r'_i.Cnt \tag{11}$$

定义 10 (结构化语言模型). 结构化语言模型基于语法规则, 通过组合运算符和实体来表示可执行的数据库查询. $SLM = (L, O, E)$, 其中, L 是数据库查询语言的语法规则集合; O 是数据库的运算符集合; $E = E_k \cup E_d \cup E_{\text{rel}} \cup E_{\text{loc}}$, 是实体的集合, 其中, E_k、E_d、E_{rel}、E_{loc} 分别表示最近邻居数、距离阈值、空间关系和查询地点的集合.

4.2 结构化语言模型

为了高效地生成空间数据库可执行语言, 设计结构化语言模型, 具体过程如图 3 所示.

基于 SECONDO 系统, 为基础空间查询、空间范围查询、最近邻居查询、空间 Join 查询和聚合查询分别构造结构化语言模型. SECONDO 支持空间和时空数据, 为用户提供了广泛的基本操作和专业操作, 常用查询如下.

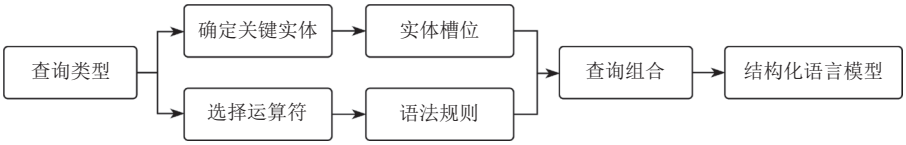

图 3　结构化语言模型的构造

(1) query <value expression>;

该命令计算给定的值表达式 (value expression) , 并将结果显示给用户.

(2) let <identifier> = <value expression>;

该命令也计算给定的值表达式, 但不同于第 1 个命令, 该命令并不直接显示值表达式的计算结果, 而是存储到名为 identifier 的对象中.

(3) delete <identifier>;

该命令从当前数据库中删除名为 identifier 的对象, 通常配合第 2 个命令使用.

SECONDO 系统中用于处理空间数据的运算符的用法和含义如表 2 所示. 不同查询类型的结构化语言用到的实体信息不同. 使用<k>表示待定的最近邻居数, <d>表示待定的距离阈值, <relation>表示待定的空间关系, <location>表示待定的查询地点. 基于数据库可执行语言的语法规则, 组合实体槽位和运算符, 可以得到基于 SECONDO 系统的结构化语言模型. 当查询地点存储于一个空间关系中时, 要将模型中的<location>替换为如下语句, 其中, <tmp_relation>表示该空间关系, <name>表示该查询地点的名称:

(<tmp_relation> feed filter [.Name = "<name>"] extract[GeoData])

表 2　处理空间数据的运算符

运算符	语法	含义
distance	{point, line, region} × {point, line, region} → real	计算两个对象之间的距离
direction	point × point → real	计算两点之间的方向
size	line → real	计算line的长度
area	region → real	计算region的面积
intersects	{line, region} × {line, region} → bool	判断两个对象是否相交
intersection	{point, line, region} × {point, line, region} → *T*, 如果point是参数, *T*为point, 否则*T*为维度较小的参数	计算两个对象的相交部分
distancescan	rtree × relation × object × int *k* → stream	计算对象的*k*个最近邻居
sortby	stream × attribute × asc / desc → stream	按给定的属性列表对元组流排序
head	stream × int *n* → stream	从流中获取前*n*个元素
filter	stream × filter condition → stream	通过谓词过滤流中的元素

基础空间查询包括如下典型应用场景.

(1) 距离查询, 计算两个空间对象之间的欧几里得距离, 需要的实体信息为查询地点, 运算符为 distance, 结合可执行语言和运算符的语法规则, 可得结构化语言模型为:

query distance (<location1>, <location2>);

(2) 方向查询, 基于方位角, 返回两个空间对象之间的相对方向, 需要的实体信息为查询地点, 运算符为 direction, 结构化语言模型为:

query direction (<location1>, <location2>);

(3) 几何查询, 针对单个空间对象, 计算其长度或面积, 需要的实体信息为查询地点, 运算符为 size 或 area, 结构化语言模型为:

query size (<location>);

query area (<location>);

根据空间对象与查询范围的关系, 空间范围查询可分为如下两类.

(1) 相交查询, 返回与查询地点具有非空交集的所有空间对象, 如检索穿过某一区域的道路或河流. 该类查询需要的实体信息为空间关系和查询地点, 运算符为 intersects, 结构化语言模型为:

query <relation> feed filter [.GeoData intersects <location>] consume;

(2) 包含查询, 提取完全位于查询范围内的空间对象, 如查询某区域内的建筑物. 该类查询需要的实体信息为空间关系和查询地点, 运算符为 within, 结构化语言模型为:

query <relation> feed filter [.GeoData within <location>] consume;

最近邻居查询需要的实体信息为空间关系、查询地点和最近邻居数, 运算符为 creatertree 和 distancescan. 运算符 creatertree 为空间关系构建 R 树索引. 运算符 distancescan 根据查询地点、存储在 R 树中的对象和最近邻居数 k, 返回 R 树中对象距离查询地点的 k 个最近邻居. 结构化语言模型为:

query <relation> creatertree [GeoData] <relation> distancescan [<location>, <k>] consume;

空间 Join 查询的核心在于基于空间关系 (如位置、距离等) 进行连接, 包括如下具体形式.

(1) 基于位置关系的 Join, 返回两个空间对象集合中所有满足空间相交条件的对象对, 例如分析每个行政区的公园. 该类查询需要的实体信息为空间关系, 运算符为 symmjoin 和 intersects. 运算符 symmjoin 将两个空间关系的元组结合起来, 等同于 SQL 中的 JOIN 关键字. 运算符 intersects 用于构成 symmjoin 运算符的连接条件. 结构化语言模型为:

query <relation1> feed {a} <relation2> feed {b} symmjoin [.GeoData_a intersects ..GeoData_b] consume;

(2) 基于距离关系的 Join, 返回两个空间对象集合中距离在给定阈值内的对象对, 例如查找距离工业园区一定范围内的住宅区. 该类查询需要的实体信息为空间关系, 运算符为 symmjoin 和 distance. 运算符 distance 用于构成 symmjoin 运算符的连接条件. 结构化语言模型为:

query <relation1> feed {a} <relation2> feed {b} symmjoin [distance (.GeoData_a, ..GeoData_b) ⩽ <d>] consume;

根据聚合类型的不同, 空间聚合查询包括如下典型形式.

(1) 数量统计, 统计与查询地点相交的空间对象数量, 例如评估某一生态保护区内的树木分布密度. 该类查询需要的实体信息为空间关系和查询地点, 运算符为 intersects 和 count, 结构化语言模型为:

query <relation1> feed extend [Cnt: fun(t: TUPLE) <relation2> feed filter [.GeoData intersects attr(t, GeoData)] count] consume;

(2) 总和统计, 计算空间对象集合中与查询地点相交部分的面积总和, 例如估算某一洪水区的受灾土地面积. 该类查询需要的实体信息为空间关系和查询地点, 运算符为 intersection 和 sum, 结构化语言模型为:

query <relation> feed extend [IntersectionArea: area (intersection (.GeoData, <location>))] sum [IntersectionArea];

(3) 最大值统计, 返回空间对象集合中, 与另一集合对象相交数量最多的对象, 例如查询南京市中拥有最多公园的行政区. 查询需要的实体为空间关系, 运算符为 intersects、sortby 和 head, 结构化语言模型为:

query <relation1> feed extend [Cnt: fun(t: TUPLE) <relation2> feed filter [.GeoData intersects attr (t, GeoData)] count] sortby [Cnt desc] head [1] consume;

结构化语言模型的设计基于两个关键原则: 查询逻辑的普适性和功能映射的灵活性. 查询逻辑是空间数据查询的基础, 它描述了查询的目标和过程, 具有很强的通用性. 此外, 不同空间数据库系统在运算符和函数的命名、参数形式上可能有所不同, 但功能本质上是相通的. 因此, 只要目标空间数据库系统具备类似功能的运算符和函数, 上述构造的结构化语言模型即可通过简单的运算符和函数替换, 适配新系统. 以 PostGIS 系统为例, 在结构化语言模型中, 和 SECONDO 系统的对应关系如表 3 所示. 在 PostGIS 的 CREATE 和 DROP 语句中, <object> 代表操作对象, 例如 FUNCTION、INDEX、TABLE 和 TYPE. 根据表 3 可得 PostGIS 的范围查询的结构化语言模型为:

SELECT * FROM <relation> WHERE ST_Intersects(<relation> .GeoData, <location>);

当转换最近邻居查询的自然语言时, PostGIS 使用的运算符为 ST_Distance, ORDER BY 和 LIMIT. ST_Distance

运算符用于计算距离. 该运算符与 ORDER BY 子句一起集成到查询中, 便于根据计算的距离按升序对结果排序. 为了进一步细化输出, 应用了 LIMIT 子句, 将结果限制为前 k 个空间对象.

表 3 SECONDO 和 PostGIS 的结构化语言模型

结构化语言模型要素	SECONDO	PostGIS
数据库查询语言L	query <value expression>;	SELECT <value expression>;
	let <identifier> = <value expression>;	CREATE <object> <identifier>;
	delete <identifier>;	DROP <object> <identifier>;
运算符O	distance	ST_Distance
	area	ST_Area
	intersects	ST_Intersects
	intersection	ST_Intersection
实体E	$E = E_k \cup E_d \cup E_{\text{rel}} \cup E_{\text{loc}}$	
结构化语言模型SLM	query distance (<location1>, <location2>);	SELECT ST_Distance (<location1>, <location2>);
	query <relation> feed filter [.GeoData intersects <location>] consume;	SELECT * FROM <relation> WHERE ST_Intersects (<relation> .GeoData, <location>);

4.3 可执行语言生成案例

基于自然语言理解阶段识别的查询类型选择匹配的结构化语言模型, 然后通过映射规则将空间数据查询实体 E 中的每个元素对应到模型实体槽集合{<k>, <d>, <relation>, <location>}中相应的元素, 生成可执行语言. 最近邻居数 k、距离阈值 d 和查询地点 *location* 在查询中是唯一的, 所以唯一对应于模型中的<k>, <d>和<location>. 空间关系 *relation* 在最近邻居查询和范围查询中只有一个, 在空间 Join 查询和聚合查询中有多个. 当查询涉及多个不同的空间关系时, 要根据关系的空间属性和运算符确定映射规则. 每类查询的可执行语言生成示例详见附录 A.

在生成空间数据库可执行语言后, 通过结合空间索引和查询计划优化技术, 能够显著提高查询的执行效率和性能. 在 SECONDO 系统中, 使用 derive 命令创建空间索引, 使用 delete 命令删除索引, 语法如下.

derive <r-tree name> = <relation> creatertree[<attribute>];

delete <r-tree name>;

其中, <relation> 为索引的空间关系, <r-tree name> 为索引名称, <attribute> 为 <relation> 用于索引的属性.

以南京市地理信息数据集为例, 空间关系如下.

district(Name:string, GeoData:region)

university(Name:string, GeoData:region)

road(Name:string, GeoData:line, Rid:int)

junction(R1id:int, R2id:int, GeoData:point)

poi(Name:string, Type:string, GeoData:point)

park(Name:string, GeoData:point)

案例 1: What are the roads that intersect the Zhixing Road?

自然语言理解阶段得到的类型为范围查询, 空间关系为 road, 查询地点为 Zhixing Road. 基于映射规则, 将关键实体信息映射到结构化语言模型中, 生成可执行语言为:

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

在该查询中, road 的空间属性和 Zhixing Road 的数据类型均为 line, 所以使用运算符 intersects 判断是否相交. 由地点知识库可得 Zhixing Road 存储于关系 road 中, 那么使用 (road feed filter [.Name = “Zhixing Road”] extract[GeoData]) 映射到结构化语言模型中的<location>, 而不是 Zhixing Road.

接下来进行查询优化, 先为空间关系创建 R 树索引, 用于快速定位可能与查询地点相交的空间对象.

derive tmp_rtree = road creatertree[GeoData];

然后使用 bbox 运算符返回查询地点的边界框, 使用 windowintersects 运算符基于 R 树索引查找空间关系中

与查询地点边界矩形框相交的对象, 初步筛选出候选子集. 最后对候选子集使用 intersects 运算符, 进行精确的空间交集计算, 确定与查询地点真实相交的对象.

query tmp_rtree road windowintersects [bbox(road feed filter [.Name = “Zhixing Road”] extract[GeoData])] filter [.GeoData intersects (road feed filter [.Name = “Zhixing Road”] extract[GeoData])] consume;

案例 2: Can you tell me the total area of the districts of Nanjing intersecting with the Yangtze River Basin?

自然语言理解阶段得到的类型为统计总和的聚合查询, 空间关系为 district, 查询地点为 Yangtze River Basin. 基于映射规则, 将关键实体信息映射到结构化语言模型中, 生成可执行语言为

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

接下来进行查询优化, 先为空间关系创建 R 树索引.

derive tmp_rtree = district creatertree[GeoData];

然后使用 bbox 和 windowintersects 运算符得到候选子集. 最后使用 intersection 运算符精确计算候选子集中每个对象与查询地点相交的部分, 使用 area 运算符计算相交部分的面积, 使用 sum 运算符对所有相交面积进行汇总, 得到总面积.

query tmp_rtree district windowintersects [bbox(Yangtze River Basin)] extend [IntersectionArea: area (intersection (.GeoData, Yangtze River Basin))] sum [IntersectionArea];

案例 3: Please find the POIs in each district that are located within a 15-minute walk of a university.

该查询可用于支持城市规划与公共服务评估任务, 帮助相关部门分析高校周边约 1 000 m (按平均步行速度 4 km/h 估算的 15 min 步行距离) 范围内的商业、生活及文化设施的空间分布情况, 进而评估高校是否处于功能完备的区域. 这类信息对于大学城发展规划、教育产业园区选址以及服务资源均衡配置具有重要参考价值. 自然语言理解阶段得到的类型为空间 Join 查询, 空间关系为 poi、district 和 university, 距离阈值为 1 000, poi 和 district 之间的连接条件为包含关系, poi 和 university 之间的连接条件为距离约束 “within a 15-minute walk”. 基于映射规则, 将关键实体信息映射到结构化语言模型中, 生成以下两个候选可执行查询.

EXE1: query poi feed {a} university feed {b} symmjoin [distance(.GeoData_a, ..GeoData_b) <= 1 000.0] district feed {c} symmjoin [.GeoData_a within ..GeoData_c] consume;

EXE2: query poi feed {a} district feed {b} symmjoin [.GeoData_a within ..GeoData_b] university feed {c} symmjoin [distance(.GeoData_a, ..GeoData_c) <= 1 000.0] consume;

上述两个查询在语义上等价, 但在执行顺序上存在差异. 为了优化查询性能, 系统对两个候选查询进行代价估算, 主要依据第 1 次连接后的中间结果规模和第 2 次连接时的计算开销. 具体估算如下.

$Cost_{EXE1} = |poi \bowtie university| \times (1 + |district|)$

$Cost_{EXE2} = |poi \bowtie district| \times (1 + |university|)$

通过实际数据库中的查询执行结果可得 $|poi \bowtie university| = 397$, $|poi \bowtie district| = 1000$, $|district| = 11$, $|university| = 89$, 进而计算得到 $Cost_{EXE1} = 4764$, $Cost_{EXE2} = 90000$. 由于 $Cost_{EXE1} < Cost_{EXE2}$, 系统最终选择 EXE1 作为最优可执行查询返回给用户.

5 实验分析

在一台运行 Ubuntu20.04 (64 位, 内核版本 5.14.0-1051-oem) 的电脑 (Intel(R) Core(TM) i5-10210U CPU, 1.60 GHz, 8 GB 内存, 512 GB 硬盘) 上对 NALSpatial 进行开发并进行实验评估. 在 SECONDO 系统中以代数模块的形式实现了 NALSpatial, 并通过该代数实现一个运算符, 名为 spatial_nl. 用户可以在 SECONDO 数据库上使用运算符 spatial_nl 将空间数据的自然语言查询转换为可执行语言.

5.1 数据集与评估指标

- 数据集. 使用 berlintest, nanjingtest, londontest 和 chinawater 数据集, 空间数据统计如表 4 所示. berlintest 数

据集存储了柏林市的城市地理信息, 包括公共交通、POI 和河流. nanjingtest 和 londontest 数据集分别存储了南京市和伦敦市的行政区、部分道路和 POI 信息. chinawater 数据集构建自全球水文数据集 HydroSHEDS[22], 包含中国部分水系的地理信息, 如河流、湖泊、池塘等水体.

表 4 空间数据统计

数据集	空间关系表数	point个数	line条数	region个数
berlintest	50	3040	4708	330
nanjingtest	6	9000	887	13
londontest	6	9032	9728	12669
chinawater	2	0	8399	2907

● 评估指标. 定义 3 个指标来评估空间数据自然语言查询转换框架的性能, 包括可翻译性 T、翻译精度 TP 和响应时间 RT.

定义 11 (**可翻译性**). 给定系统成功生成的可执行语言集合 Q_{success}, 输入的自然语言查询集合 Q_{total}, 可翻译性 $T = |Q_{\text{success}}|/|Q_{\text{total}}|$.

定义 12 (**翻译精度**). 给定系统生成的执行结果符合预期的可执行语言集合 Q_{correct}, 输入的自然语言查询集合 Q_{total}, 翻译精度 $TP = |Q_{\text{correct}}|/|Q_{\text{total}}|$.

定义 13 (**响应时间**). 给定系统接收到自然语言查询的时间 t_{start}, 完成可执行查询生成的时间 t_{end}, 响应时间 $RT = t_{\text{end}} - t_{\text{start}}$.

● 测试用例. 基于 berlintest, nanjingtest, londontest 和 chinawater 数据集构建 100 条测试用例, 包含 20 条范围查询, 20 条最近邻居查询, 30 条空间 Join 查询和 30 条聚合查询. 结合数据集的实际应用场景, 测试用例涵盖了不同类型的空间查询需求, 可通过 https://pan.baidu.com/s/1fze_iUjRJbMzhrbeY9Z8XQ?pwd=ujwb 获取.

5.2 对比实验

为了全面评估 NALSpatial 的性能与优势, 设计两组对比实验: (1) 与基于传统规则或机器学习的转换方法对比; (2) 与基于大语言模型的转换方法对比. 这两组实验旨在从不同技术路径出发, 验证 NALSpatial 在有效性和用户友好性等方面的表现.

5.2.1 对比实验设置

在第 1 组实验中, 选择了具有代表性的 3 种转换方法: SpatialNLI[23]、IRNet 和 ATHENA++[24]. 首先在实验环境中复现了这 3 种方法的关键模块, 然后结合 NALSpatial 的查询类型识别模型、映射规则和结构化语言模型, 最终生成 SECONDO 系统的可执行语言.

(1) SpatialNLI 关键模块的实现: 首先检测潜在的关键字和数据元素, 通过字符串的精确匹配、根据编辑距离计算单词间的相似性、采用余弦相似度衡量两个单词在语义上的相近性. 然后构造空间理解模型, 根据上下文来确定有歧义的空间短语的含义. 对输入的自然语言查询进行预处理, 将有歧义的空间短语用特殊的符号包围起来, 以表明它比问题中的其他符号具有更大的影响力. 输入预处理后的自然语言查询和有歧义的短语的一个语义 t, 如果模型返回 true, 则有歧义的空间短语的语义为 t.

(2) IRNet 关键模块的实现: 首先识别自然语言查询中的实体, 包括列、表和值. 使用基于统计语言模型的算法 N-Gram 和字符串匹配的方法来识别查询中的实体. 然后根据问题中提到的方式为列分配不同的类型. 如果被识别为列的 n-gram 与数据库模式中的列名完全匹配, 则为这些列分配一个类型为 EXACT MATCH, 否则为一个类型 PARTIAL MATCH.

(3) ATHENA++关键模块的实现: 首先根据对象知识库、关系知识库和地点知识库构建空间领域本体. 然后基于构建的空间领域本体和 Relational Store, 利用自然语言处理工具 Stanza 提取自然语言查询中的关键实体信息. 如果自然语言查询中的一个单词映射到多个本体元素, 则为每种情况生成相应的可执行语言.

在第 2 组实验中, 选择了当前主流的大语言模型 GPT-4o 作为对比基准. 为实现与大语言模型的公平对比, 设

计了用于大语言模型的提示词, 以优化其在空间数据自然语言查询转换任务中的表现. Gao 等人[25]的评估强调, 在 0-shot 场景下, 对 GPT-4 和 GPT-3.5-TURBO 进行 OpenAI Demonstration 提示是很好的选择. 此外, GPT-4 可以有效地从 NLQ-SQL 对中学习映射[26]. 综合考虑提示的有效性和制定难度, 使用 1-Shot Learning (1SL) 策略对 GPT-4o 进行提示, 它提供了表模式 (使用 “#” 号注释) 和一个 NL2SQL 的黄金示例. 以 chinawater 数据集为例, 提示内容如下.

SQLite SQL tables , with their properties :

#

water(ID, Type, Name, GeoData)

waterway (ID, Type, Name, GeoData)

#

Translate text to SQL: What are the waterways that intersect the Agongdian River? →

SELECT * FROM waterway WHERE ST_Intersects(GeoData, (SELECT GeoData FROM water WHERE Name = ‘Agongdian River’));

What are the waterways that pass through each body of water? →

5.2.2 对比实验结果分析

在第 1 组实验中, 对比了 NALSpatial、SpatialNLI、IRNet 和 ATHENA++在平均响应时间 *RT*、可翻译性 *T* 和翻译精度 *TP* 上的表现, 实验结果如图 4 所示. SpatialNLI 的平均响应时间是最长的, 翻译精度是最高的. 由于空间理解模型的使用, SpatialNLI 在解析复杂空间查询时表现出卓越的翻译精度. 然而, 这种高精度的代价是较长的平均响应时间. IRNet 的平均响应时间与 NALSpatial 接近, 它在查询解析和转换效率方面具有一定优势. 然而, 由于其通用性设计在空间领域的适配能力有限, 可翻译性和翻译精度均低于 NALSpatial. ATHENA++利用空间领域本体对查询中的关键实体信息进行快速识别, 因而具有最低的平均响应时间. 这一特点使其在对响应速度要求极高的场景下具有潜在优势. 然而, 可翻译性和翻译精度较低表明其对复杂空间查询的支持能力有限. NALSpatial 未能准确翻译的原因主要有 3 个: (1) spaCy 错误识别单词的词性; (2) 无法识别单词间的语义相似性; (3) 查询的类型未被准确识别. 实验结果表明, 在测试用例覆盖的场景下, NALSpatial 在平均响应时间、可翻译性和翻译精度 3 项指标上实现了较为理想的平衡, 综合性能优于其他 3 种方法.

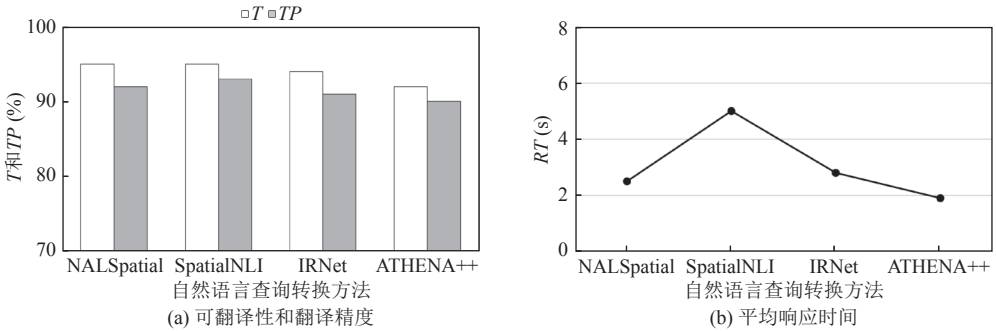

图 4 NALSpatial 与基于传统规则或机器学习的转换方法对比的实验结果

此外, 对 NALSpatial、SpatialNLI、IRNet 和 ATHENA++在不同类型查询上的可翻译性和翻译精度进行了全面对比, 实验结果如图 5 所示. 范围查询的语句结构简单, 但严重依赖地点的提取. 实验结果表明, 在所有系统中范围查询的可翻译性和翻译精度都未超过 90%. 最近邻居查询涉及的实体信息最多, 语句结构复杂, 但由于其格式相对固定, 测试数据集上的最近邻居查询都能很好地被转换为可执行语句. 空间 Join 查询中距离阈值的提取受 NLP 工具影响较大. 实验结果表明, 4 种系统在此类查询的可翻译性和翻译精度上存在较大差异. 聚合查询主要涉及统计计算, 语义解析较为直接, 因此其翻译精度在 4 种系统中均处于 93%–97% 的稳定范围内.

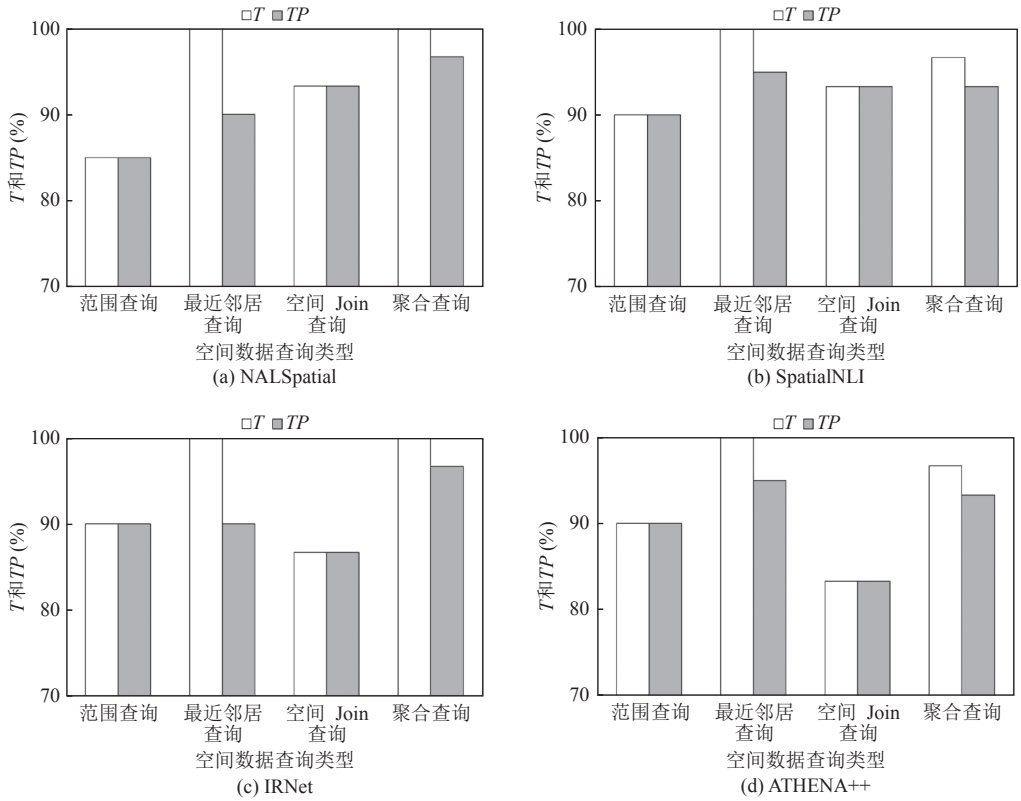

图 5 转换方法在不同类型查询上的性能对比

在第 2 组实验中, 针对可翻译性和翻译精度对比了 NALSpatial 和 GPT-4o, 实验结果如表 5 所示. GPT-4o 依赖大规模预训练模型, 通过上下文提示解析自然语言查询. 实验结果表明, 1SL 策略在一定程度上提升了 GPT-4o 的性能, 特别是在空间查询中位置表示的准确性上表现突出. 然而, GPT-4o 在选择合适的空间运算符来判断位置间关系时依然面临挑战. 相比之下, NALSpatial 无需额外的用户交互, 利用知识库来准确地提取和表示位置, 还可以使用数据库中新增的空间运算符, 在处理空间位置和关系的解析上表现更为可靠. 特别是在范围查询的转换中, 由于查询依赖于对空间位置和运算符的精确选择, NALSpatial 能够通过知识库和运算符优化进行高效解析. 而 GPT-4o 在这类查询中的表现则受限于提示的质量和对空间关系的理解深度, 翻译结果往往不够精确或存在逻辑缺陷.

表 5 NALSpatial 与 GPT-4o 对比的实验结果 (%)

转换方法	范围查询		最近邻居查询		空间Join查询		聚合查询	
	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

5.3 消融实验

为了验证 NALSpatial 中各个模块的有效性, 进行消融实验, 系统性评估了在去除部分模块的情况下框架的性能变化. 具体而言, 通过对框架的可翻译性 *T*、翻译精度 *TP* 和响应时间 *RT* 进行测量, 从不同维度探讨各模块的作用和整体系统的依赖性. 实验结果如图 6 所示, 其中, −KB、−Corpus、−MR、−SLM 分别表示去除知识库模块、语料库模块、映射规则模块和结构化语言模型模块的 NALSpatial 框架配置.

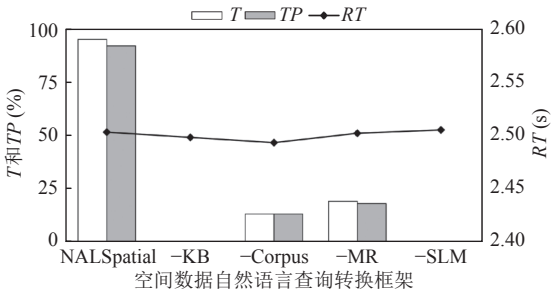

图 6 NALSpatial 的消融实验结果

实验发现, 当去除知识库模块时, NALSpatial 只能依赖基础的自然语言处理工具和算法完成语言理解. 这种情况下, 系统无法提取出空间关系. 测试用例包含空间范围查询、最近邻居查询、空间 Join 查询和聚合查询, 在生成数据库可执行语言时均依赖对空间关系的精准提取. 由于缺少知识库模块支撑, 自然语言查询转换框架在测试用例中的可翻译性和翻译精度均为 0, 无法生成任何可执行语言.

当去除语料库模块时, NALSpatial 无法依赖数据驱动的查询类型预测, 而是使用随机分配方式推测查询类型. 此策略导致系统的查询类型识别精度显著下降, 仅能正确识别出 15 条自然语言查询的类型, 其中由于未能准确提取测试用例 Q6 和 Q41 的实体信息, 最终系统仅能成功转换 13 条自然语言查询为可执行语言, 且所有转换结果均符合预期. 因此, 此配置下的可翻译性和翻译精度均为 13%, 显著低于完整的转换框架.

Q6: Can you give me details about the districts that intersect with Lingrui Road?

转换失败的原因是系统错将 Road 识别为一个空间关系, 没有识别出 Lingrui Road 地点.

Q41: Which POIs are located within a one kilometer radius of each POI?

转换失败的原因是系统错将 radius 识别为 POI 关系中的一个地点 Radis.

当去除映射规则模块时, NALSpatial 只能将提取出的实体信息随机映射到结构化语言模型中, 在一定程度上降低了翻译性能. 实验中, 系统正确映射了 19 条查询, 但其中一条查询 Q21 的转换结果不符合预期, 最终系统只能成功生成 18 条符合预期的数据库可执行语言. 因此, 此配置下的可翻译性为 19%, 翻译精度为 18%.

Q21: What are the top 10 POIs closest to the boundary line of Nanjing?

不符合预期的原因是系统误将该查询的意图理解为求距离南京市中心最近的 10 个 POI, 而不是南京市边界线.

当去除结构化语言模型时, NALSpatial 完全失去生成可执行语言的能力. 尽管仍能提取查询中的关键语义信息, 但缺乏语言生成能力的系统无法完成自然语言查询的最终转换. 因此, 在该配置下, 系统的可翻译性和翻译精度均为 0.

此外, 无论是否去除模块, 系统的响应时间始终保持在 2.45–2.55 s 之间, 表明单个模块的缺失不会对系统的实时性产生明显影响. 消融实验结果表明, 知识库模块对空间关系提取至关重要, 缺失直接导致系统无法生成任何有效的可执行语言. 语料库模块显著影响查询类型识别的准确性, 缺失使系统的翻译性能大幅下降. 映射规则模块是生成精确数据库查询的关键, 缺失会导致系统映射准确率降低, 进而影响整体翻译精度. 结构化语言模型模块是实现查询转换的核心, 缺失使系统完全失去可执行语言生成能力. 完整的 NALSpatial 框架能够通过各模块的协同工作, 全面实现高精度、高可翻译性和高效能的空间数据自然语言查询转换功能.

5.4 性能验证实验

为了验证 NALSpatial 的性能稳定性, 设计并实施了两组性能验证实验.

(1) 通过调整范围查询、最近邻居查询、空间 Join 查询和聚合查询中的关键参数, 分析可翻译性 T、翻译精度 TP 和响应时间 RT 的变化情况, 从而评估 NALSpatial 对多样化查询任务的适应能力.

(2) 通过扩展数据集的规模和调整测试用例的数量, 分析平均语义理解时间的变化情况. 实验数据集选用

chinawater, 并借助 HydroSHEDS 数据进行扩充, 以评估 NALSpatial 在不同数据负载条件下的处理效率和稳定性.

● 范围大小的影响. 按照区域面积依次倍增, 选取南京市的玄武区、雨花台区、栖霞区和江宁区进行范围查询. 基于测试用例, 为每个行政区构造 10 个对相同地理信息 (包括 POI、road、park 等) 的范围查询. 根据区域的不同, 将这些 NLQ 分为 4 组: A 组是玄武区, B 组是雨花台区, C 组是栖霞区, 和 D 组是江宁区. 分别测试每组的可翻译性、翻译精度和平均响应时间. 实验结果如图 7(a) 所示, 每组的可翻译性都为 100%, 翻译精度都为 90%, 平均响应时间保持在 2.55–2.65 s 之间.

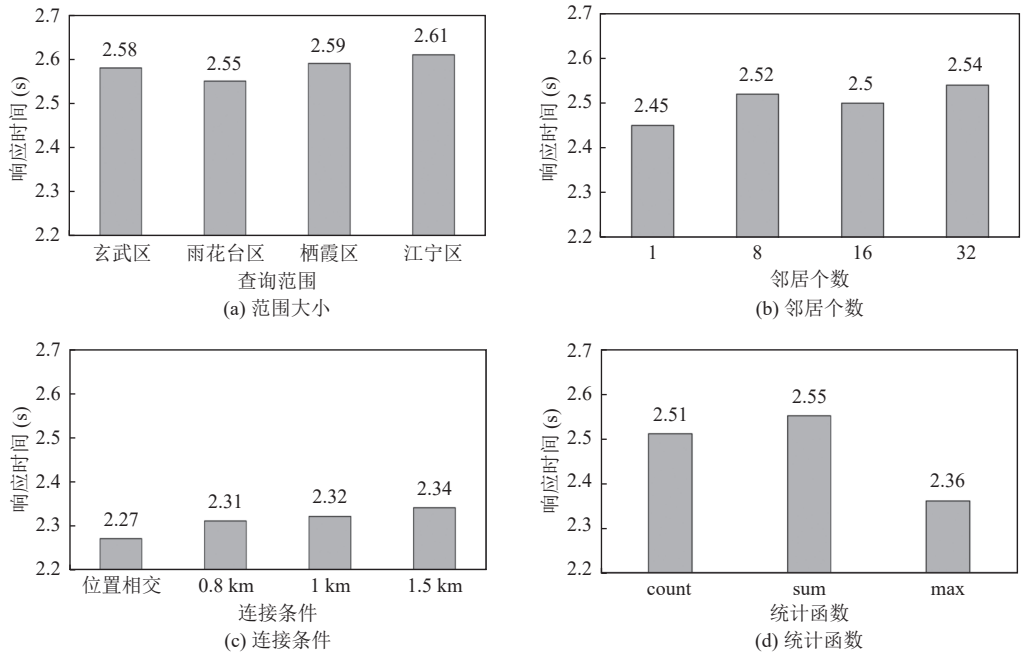

图 7 NALSpatial 的第 1 组性能验证实验结果

● 邻居个数的影响. 基于测试用例, 选取 10 个最近邻居查询, 通过将最近邻居数修改为 1、8、16 或 32 扩展到 40 个查询. 根据最近邻居数的不同, 将这些 NLQ 分为 4 组: A 组是 1 个, B 组是 8 个, C 组是 16 个, 和 D 组是 32 个. 分别测试每组的可翻译性、翻译精度和平均响应时间. 实验结果如图 7(b) 所示, 每组的可翻译性都为 100%, 翻译精度都为 100%, 平均响应时间保持在 2.45–2.55 s 之间.

● 连接条件的影响. 基于测试用例, 选取 10 个空间 Join 查询, 通过将连接条件修改为位置相交、距离小于 800 m、距离小于 1 km 或距离小于 1.5 km 扩展到 40 个查询. 根据连接条件的不同, 将这些 NLQ 分为 4 组: A 组是位置相交, B 组是距离小于 800 m, C 组是距离小于 1 km, 和 D 组是距离小于 1.5 km. 分别测试每组的可翻译性、翻译精度和平均响应时间. 实验结果如图 7(c) 所示, 每组的可翻译性都为 90%, 翻译精度都为 90%, 平均响应时间保持在 2.25–2.35 s 之间.

● 统计函数的影响. 基于测试用例, 选取 10 个聚合查询, 通过将统计函数修改为 count、sum 或 max 扩展到 30 个查询. 根据统计函数将的不同, 将这些 NLQ 分为 3 组: A 组是 count 函数, B 组是 sum 函数, 和 C 组是 max 函数. 分别测试每组的可翻译性、翻译精度和平均响应时间. 实验结果如图 7(d) 所示, 每组的可翻译性都为 100%, 翻译精度都为 90%. 系统的平均响应时间随聚合查询的统计函数的改变而不同. 统计函数为 sum 的聚合查询涉及地点的识别和相交面积的计算, 响应时间是最大的. 统计函数为 max 的聚合查询的关键实体信息提取只涉及空间关系, 响应时间是最小的.

第 1 组性能验证实验结果表明, NALSpatial 在多种空间查询任务下均表现出良好的稳定性. 空间范围查询、最近邻居查询、空间 Join 查询和聚合查询的参数调整未对系统的整体可翻译性、翻译精度和响应时间造成显著影响.

● 数据集规模扩展. 分别将 chinawater 数据集中的地点数量扩展至 4 个不同规模, 即 10 000 (10k)、100 000 (100k)、1 000 000 (1M) 和 10 000 000 (10M). 使用测试用例对每组数据进行测试, 计算平均语义理解时间. 实验结果表明, 随着数据集规模的增大, 平均语义理解时间始终保持在 2.35–2.55 s 之间, 如图 8(a) 所示. 得益于为知识库构建的高效哈希表结构, NALSpatial 在进行地点和空间关系查找时具备 O(1) 的时间复杂度, 从而在大规模数据集上可以实现快速的语义理解.

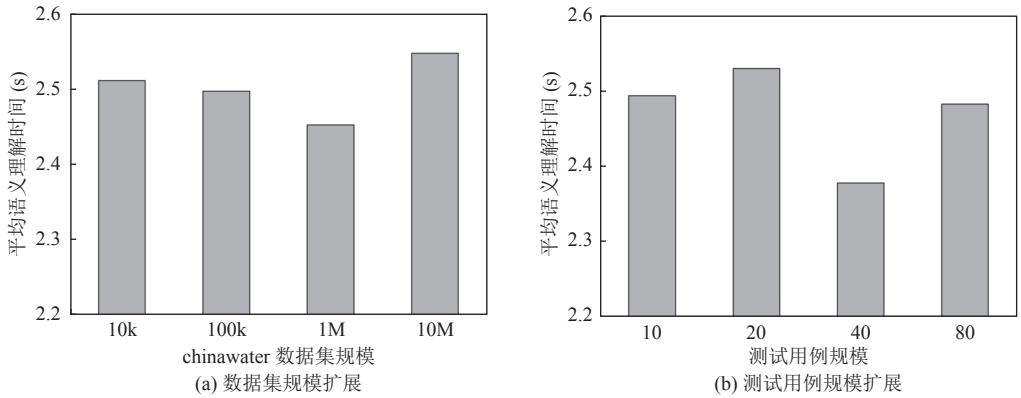

图 8 NALSpatial 的第 2 组性能验证实验结果

● 测试用例规模扩展. 固定 chinawater 数据集中的地点数量为 10k, 考察不同测试用例规模对系统性能的影响. 设置测试用例的规模分别为 10、20、40 和 80 条查询, 记录并分析平均语义理解时间, 结果如图 8(b) 所示. 实验结果表明, 无论测试用例规模如何变化, 平均语义理解时间保持在 2.35–2.55 s 间, 验证了 NALSpatial 在高负载测试用例条件下的响应性能.

第 2 组性能验证实验结果表明, 自然语言理解过程的时间成本主要取决于哈希表的索引效率和实体信息提取算法, 与数据集和测试用例的规模并无直接线性关系. 因此, NALSpatial 具备良好的横向扩展能力, 能够适应大规模空间数据集的应用需求.

此外, 为了验证 BiLSTM 在查询类型识别任务中的表现, 对 TextCNN、LSTM、BiLSTM、DistilBERT 和 BERT 模型进行对比实验. 将语料库以 8:2 的比例划分为训练集和测试集, 记录每个模型的平均每轮训练时间和测试集准确率, 实验结果如表 6 所示. 对比准确率, BiLSTM、DistilBERT 与 BERT 均显著优于 LSTM 和 TextCNN, 其中, BiLSTM 在传统模型中表现最佳, 准确率接近预训练模型水平. 对比训练时间, TextCNN 最短, 但准确率最低, 难以满足语义理解需求. 和 LSTM 相比, BiLSTM 的训练时间略长, 但准确率提升显著, 在性能与效率之间实现了更优平衡. DistilBERT 和 BERT 的准确率虽高, 但训练时间很长, 不利于语料库的持续更新. 相比之下, BiLSTM 在保持高准确率的同时具备较低训练开销, 综合性能最优, 是训练语料库的最优选择.

表 6 查询类型识别模型的性能对比

模型	平均每轮训练时间 (s)	测试集准确率 (%)
TextCNN	2.43	88.50
LSTM	3.35	94.29
BiLSTM	5.03	98.57
DistilBERT	78.86	99.49
BERT	134.58	99.58

6 总结与展望

本文提出了一种基于大语言模型的空间数据库自然语言查询转换方法 NALSpatial, 支持基础空间查询、范围查询、最近邻居查询、空间 Join 查询和聚合查询. NALSpatial 的架构主要分为两个核心模块: 自然语言理解和可执行语言生成. 首先, 通过应用知识库和基于大语言模型构建的语料库, 提取出关键实体和查询类型. 然后, 根据查询类型选择结构化语言模型, 将实体映射到模型中构造数据库可执行语言. 实验结果表明, NALSpatial 能够有效地将空间数据自然语言查询转换为可执行的数据库查询语句. 本文所提方法主要聚焦于文本形式的自然语言查询, 未来可以探索多模态数据的自然语言查询, 如结合语音或图像的查询解析.

References

[1] Liu ZD, Lin WX, Wu KS. A spatial structure matching algorithm for large spatial-textual datasets. Chinese Journal of Computers, 2022, 45(6): 1261–1275 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2022.01261]

[2] Yu J, Sarwat M. GEOSPARKVIZ: A cluster computing system for visualizing massive-scale geospatial data. The VLDB Journal, 2021, 30(2): 237–258. [doi: 10.1007/S00778-020-00645-2]

[3] Xie J, Chen Z, Liu JW, Wang F, Li FF, Chen ZD, Liu YP, Cai SL, Fan ZH, Xiao F, Chen Y. Ganos: A multidimensional, dynamic, and scene-oriented cloud-native spatial database engine. Proc. of the VLDB Endowment, 2022, 15(12): 3483–3495. [doi: 10.14778/3554821.3554838]

[4] Güting RH, Behr T, Düntgen C. SECONDO: A platform for moving objects database research and for publishing and integrating research implementations. IEEE Data Engineering Bulletin, 2010, 33(2): 56–63.

[5] Hori K, Sasaki Y, Amagata D, Murosaki Y, Onizuka M. Learned spatial data partitioning. In: Proc. of the 6th Int'l Workshop on Exploiting Artificial Intelligence Techniques for Data Management. Seattle: ACM, 2023. 3. [doi: 10.1145/3593078.3593932]

[6] Zardbani F, Mamoulis N, Idreos S, Karras P. Adaptive indexing of objects with spatial extent. Proc. of the VLDB Endowment, 2023, 16(9): 2248–2260. [doi: 10.14778/3598581.3598596]

[7] Lin XC, Wei KM, Li ZT, Chen JP, Pei TR. Aggregation-based dual heterogeneous task allocation in spatial crowdsourcing. Frontiers of Computer Science, 2024, 18(6): 186605. [doi: 10.1007/S11704-023-3133-6]

[8] Manning CD, Surdeanu M, Bauer J, Finkel JR, Bethard S, Mcclosky D. The Stanford CoreNLP natural language processing toolkit. In: Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations. Baltimore: ACL, 2014. 55–60. [doi: 10.3115/V1/P14-5010]

[9] Algamdi S, Albanyan A, Shah SK, Tariq Z. Twitter accounts suggestion: Pipeline technique spaCy entity recognition. In: Proc. of the 2022 IEEE Int'l Conf. on Big Data. Osaka: IEEE, 2022. 5121–5125. [doi: 10.1109/BIGDATA55660.2022.10020570]

[10] Potočár M, Kvet M. Comparison of unigram, HMM, CRF and Brill's part-of-speech taggers available in NLTK library. In: Proc. of the 33rd Conf. of Open Innovations Association. Zilina: IEEE, 2023. 226–235. [doi: 10.23919/FRUCT58615.2023.10143061]

[11] Qi P, Zhang YH, Zhang YH, Bolton J, Manning CD. Stanza: A Python natural language processing toolkit for many human languages. In: Proc. of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations. ACL, 2020. 101–108. [doi: 10.18653/V1/2020.ACL-DEMOS.14]

[12] Li SC, Wang ZQ, Zhou GD. LLM enhanced cross domain aspect-based sentiment analysis. Ruan Jian Xue Bao/Journal of Software, 2025, 36(2): 644–659 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/7156.htm [doi: 10.13328/j.cnki.jos.007156]

[13] Kim H, So BH, Han WS, Lee H. Natural language to SQL: Where are we today? Proc. of the VLDB Endowment, 2020, 13(10): 1737–1750. [doi: 10.14778/3401960.3401970]

[14] Popescu AM, Etzioni O, Kautz H. Towards a theory of natural language interfaces to databases. In: Proc. of the 8th Int'l Conf. on Intelligent User Interfaces. Miami: ACM, 2003. 149–157. [doi: 10.1145/604045.604070]

[15] Li F, Jagadish HV. Understanding natural language queries over relational databases. ACM SIGMOD Record, 2016, 45(1): 6–13. [doi: 10.1145/2949741.2949744]

[16] Pan X, Xu SH, Cai XR, Wen YL, Yuan XJ. Survey on deep learning based natural language interface to database. Journal of Computer Research and Development, 2021, 58(9): 1925–1950 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2021.20200209]

[17] Guo JQ, Zhan ZC, Gao Y, Xiao Y, Lou JG, Liu T, Zhang DM. Towards complex text-to-SQL in cross-domain database with intermediate representation. In: Proc. of the 57th Conf. of the Association for Computational Linguistics. Florence: ACL, 2019. 4524–4535. [doi: 10.18653/V1/P19-1444]

[18] Brunner U, Stockinger K. ValueNet: A natural language-to-SQL system that learns from database information. In: Proc. of the 37th IEEE Int'l Conf. on Data Engineering. Chania: IEEE, 2021. 2177–2182. [doi: 10.1109/ICDE51399.2021.00220]

[19] Kordjamshidi P. Spatial and temporal reasoning with LLMs for natural language comprehension and grounding. In: Proc. of the 2nd Int'l

Workshop on Spatio-temporal Reasoning and Learning Co-located with the 32nd Int'l Joint Conf. on Artificial Intelligence. 2023.

[20] Wang XY, Liu MY, Xu JQ, Lu H. NALMO: Transforming queries in natural language for moving objects databases. GeoInformatica, 2023, 27(3): 427–460. [doi: 10.1007/S10707-023-00494-5]

[21] Fan YK, He ZY, Ren TH, Guo DJ, Chen L, Zhu RS, Chen GD, Jing YN, Zhang K, Wang XS. GAR: A generate-and-rank approach for natural language to SQL translation. In: Proc. of the 39th IEEE Int'l Conf. on Data Engineering. Anaheim: IEEE, 2023. 110–122. [doi: 10.1109/ICDE55515.2023.00016]

[22] Lehner B, Verdin K, Jarvis A. New global hydrography derived from spaceborne elevation data. Eos, Trans., American Geophysical Union, 2008, 89(10): 93–94. [doi: 10.1029/2008EO100001]

[23] Wang WL, Li JJ, Ku WS, Wang HX. Multilingual spatial domain natural language interface to databases. GeoInformatica, 2024, 28(1): 29–52. [doi: 10.1007/S10707-023-00496-3]

[24] Sen J, Lei C, Quamar A, Özcan F, Efthymiou V, Dalmia A, Stager G, Mittal A, Saha D, Sankaranarayanan K. ATHENA++: Natural language querying for complex nested SQL queries. Proc. of the VLDB Endowment, 2020, 13(12): 2747–2759. [doi: 10.14778/3407790.3407858]

[25] Gao DW, Wang HB, Li YL, Sun XY, Qian YC, Ding BL, Zhou JR. Text-to-SQL empowered by large language models: A benchmark evaluation. Proc. of the VLDB Endowment, 2024, 17(5): 1132–1145. [doi: 10.14778/3641204.3641221]

[26] Sun S, Zhang YC, Yan JH, Gao YZ, Ong D, Chen B, Su J. Battle of the large language models: Dolly vs LLaMA vs Vicuna vs Guanaco vs Bard vs ChatGPT—A text-to-SQL parsing comparison. In: Findings of the Association for Computational Linguistics: EMNLP 2023. Singapore: ACL, 2023. 11225–11238. [doi: 10.18653/V1/2023.FINDINGS-EMNLP.750]

附中文参考文献

[1] 刘志丹, 林维鑫, 伍楷舜. 一种面向大规模空间文本数据的空间结构匹配算法. 计算机学报, 2022, 45(6): 1261–1275. [doi: 10.11897/SP.J.1016.2022.01261]

[12] 李诗晨, 王中卿, 周国栋. 大语言模型驱动的跨领域属性级情感分析. 软件学报, 2025, 36(2): 644–659. http://www.jos.org.cn/1000-9825/7156.htm [doi: 10.13328/j.cnki.jos.007156]

[16] 潘璇, 徐思涵, 蔡祥睿, 温延龙, 袁晓洁. 基于深度学习的数据库自然语言接口综述. 计算机研究与发展, 2021, 58(9): 1925–1950. [doi: 10.7544/issn1000-1239.2021.20200209]

附录 A. 可执行语言生成示例

空间数据自然语言查询的可执行语言生成示例如表 A1 所示.

表 A1 可执行语言生成示例

自然语言查询	查询类型	转换后的可执行语言
Returns the distance between Mehringdamm and Alexanderplatz.	基础空间查询	query distance (Mehringdamm, Alexanderplatz);
What's the direction from Mehringdamm to Alexanderplatz?		query direction (Mehringdamm, Alexanderplatz);
How long is the border of Nanjing?		query size (NJBorderLine);
Could you tell me the area of Nanjing University?		query area (university feed filter [.Name = "Nanjing University"] extract[GeoData]);
What are the roads that intersect the Zhixing Road?	空间范围查询	query road feed filter [.GeoData intersects (road feed filter [.Name = "Zhixing Road"] extract[GeoData])] consume;
Please tell me what are the 10 closest POIs to the Nanjing border line?	最近邻居查询	query poi createrree [GeoData] poi distancescan [NJBorderLine, 10] consume;
What are the POIs in each district of Nanjing?	空间Join查询	query poi feed {a} district feed {b} symmjoin [.GeoData_a within ..GeoData_b] consume;
What are the POIs within 1.5 kilometers from the center of each district in Nanjing?		query poi feed {a} district feed {b} symmjoin [distance (.GeoData_a, ..GeoData_b) <= 1500.0] consume;
Please find the POIs in each district that are located within a 15-minute walk of a university.		query poi feed {a} university feed {b} symmjoin [distance (.GeoData_a, ..GeoData_b) <= 1000.0] district feed {c} symmjoin [.GeoData_a within ..GeoData_c] consume;

表 A1 可执行语言生成示例 (续)

自然语言查询	查询类型	转换后的可执行语言
How many POIs are there in each district of Nanjing?	空间聚合查询	query district feed extend [Cnt: fun(t: TUPLE) poi feed filter [.GeoData within attr(t, GeoData)] count] consume;
Can you tell me the total area of the districts of Nanjing intersecting with the Yangtze River Basin?		query district feed extend [IntersectionArea: area (intersection (.GeoData, Yangtze River Basin))] sum[IntersectionArea];
Which district in Nanjing has the most POIs?		query district feed extend [Cnt: fun(t: TUPLE) poi feed filter [.GeoData within attr(t, GeoData)] count] sortby[Cnt desc] head[1] consume;

作者简介

刘孟怡, 博士生, CCF 学生会员, 主要研究领域为空间数据库.

许建秋, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为数据软件, 空间数据库, 移动对象数据库, 可扩充数据库.

童咏昕, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为联邦学习, 时空大数据分析处理, 智慧城市, 众包计算, 群体智能, 隐私保护.