单位代码：10610
送审编号：2020223040092

四川大學

硕士研究生学位论文

（学术学位）

题目： **基于数据库模式信息扩展和视图的 Text2SQL 方法**

学位类别：工学硕士

学科名称：软件工程

学位论文完成时间：二〇二三年十月

摘 要

现实世界中大量数据保存在关系型数据库中，用户通过应用系统提供的界面访问这些数据，系统开发者负责将用户请求转化成数据库查询引擎所支持的 SQL 语句。由于受限于应用系统所提供的界面，用户无法灵活利用数据库中的数据，任何数据访问需求的变更都必须依赖于应用系统开发者设计新的 SQL 程序。为此，Text2SQL 任务被提出，旨在将用户提出的自然语言查询转化为可执行的 SQL 语句。现有的 Text2SQL 方法仅仅依赖数据库模式捕捉数据库建模的领域知识，导致模型严重依赖表名和列名在自然语言查询中的显式提及，在同物异名的实际应用场景中准确率急剧下降。现有研究引入了序列化、图神经网络等技术，以更好地捕捉数据库模式的语义，从而更好地生成复杂 SQL 语句。然而，现有研究的多表查询翻译准确率仍然明显低于单表查询，当数据库内仅包含一张表时，模型的单表查询准确率可以达到最高水平。这表明，表和列的包含关系以及表名会在 SQL 语句中的哪些位置出现会增加模型的学习难度。为此，本文分别提出了基于数据库模式信息扩展和基于视图的 Text2SQL 方法，通过词典扩展数据库模式信息以更准确地捕捉数据库建模的领域知识，将多表查询和单表查询转化为生成视图上的 SQL 骨架（skeleton），以提高 SQL 语句的生成质量。本文具体工作包括：

（1）针对现有方法仅依赖数据库模式捕捉数据库建模的领域知识的问题，本文提出利用词典扩展数据库模式信息的 Text2SQL 方法，该方法从数据库表名和列名解析出其中的单词或短语，查询词典获取这些单词或短语的语义解释，将这些解释看成是相应表名或列名的扩展内容，与表名、列名及其他数据库模式信息（主键、外键等）相结合，作为模型的输入，从而使模型能够更全面地学习数据库建模的应用领域知识。在 Spider-syn 和 Spider 数据集上进行的实验说明了所提出方法的有效性，即使自然语言查询中使用的表名和列名与数据库模式中对应的表名和列名完全不同，本文方法也能够得到较好的 SQL 翻译结果。

（2）现有研究的多表查询翻译准确率明显低于单表查询，当数据库内仅包含一张表时，模型的准确率可以达到最高水平。因此，本文提出了基于视图的 Text2SQL 方法。该方法在常见的查询场景中事先创建视图，将多表查询和单表查询归结为单视图查询。在这种情况下，模型仅将当前视图的列名和自然语言查询作为输入，既不需要学习表和列之间的包含关系，也无需判断表名会在 SQL 语句中的哪些位置出现。同时，模型从生成基本表上的 SQL 语句转化为生成视图上不包含表名和 from 子句的 SQL 骨架，有效地降低了生成难度。在 Spider-syn 和 Spider 数据集上进行了一系列实验，实验结果证明，视图查询能提高模型的 Text2SQL 准确率。

关键词：数据库模式；语义扩展；视图；Text2SQL

Abstract

In the real world, a significant amount of data is stored in relational databases. Users access this data through interfaces provided by application systems, and it is the responsibility of system developers to translate user requests into SQL statements supported by the database query engine. However, due to the limitations imposed by the interfaces provided by application systems, users often cannot efficiently leverage the data in the database. Any changes to data access requirements require developers to design new SQL programs. To address this issue, the Text2SQL task was introduced, aiming to convert user-provided natural language queries into executable SQL statements. Existing Text2SQL methods primarily rely on database schemas to capture domain knowledge related to database modeling. This results in models heavily depending on explicit mentions of table and column names in natural language queries. In practical scenarios where the same thing has the different name, this approach leads to a sharp decrease in accuracy. To overcome these challenges, recent research has introduced techniques like serialization and graph neural networks to better capture the semantic aspects of the database schema, thus improving the generation of complex SQL statements. However, the accuracy of existing research in translating multi-table queries remains significantly lower than single-table queries. When the database contains only one table, the accuracy of the model in single-table queries reaches its highest level. This indicates that the relationships between tables and columns and where table names appear in SQL statements increase the complexity of model learning. We present two methods for Text2SQL: one based on database schema information extension and another based on view generation. The methods aim to enhance the modeling of a database by extending schema information using dictionaries to capture domain knowledge more accurately and convert multi-table queries and single-table queries into the generation of SQL skeletons on views, improving the quality of SQL statement generation. Our contributions include:

（1）In response to the limitations of existing methods that rely solely on database schema to capture domain knowledge for database modeling, we propose a Text2SQL method that extends database schema information using dictionaries. This method extracts words or phrases from table and column names, queries the dictionary to obtain the semantic explanations, and treats these interpretations as extensions of the corresponding table and column names and combined with other database schema information such as primary keys and foreign keys. As a result, the model gains a more comprehensive understanding of domain-specific knowledge

related to database modeling. Experiments conducted on the Spider-syn and Spider datasets demonstrate the effectiveness of the proposed approach. Even when natural language queries use table and column names that are entirely different with the database schema, the method is capable of producing high-quality SQL translation results.

（2）Existing research has shown that the accuracy of multi-table query translation is significantly lower than that of single-table queries. When the database contains only one table, model accuracy can reach its highest level. Therefore, we introduce a view-based Text2SQL method to address this issue. This method creates views in advance in common query scenarios, and reduces multi-table queries and single-table queries into single-view queries. In such cases, the model takes only the column names of the current view and the natural language query as input. There is no need to learn the inclusion relationships between tables and columns, and it does not require determining where table names will appear in the SQL statement. Furthermore, the model transforms from generating SQL statements on base tables to generating SQL skeletons on views that do not contain table names or FROM clauses. This transformation effectively reduces the generation complexity. A series of experiments conducted on the Spider-syn and Spider datasets show that view-based queries can improve the Text2SQL accuracy.

Key words: database schema; semantic extension; view; Text2SQL

目 录

图目录

表目录

第1章 绪论

1.1 选题背景及意义

随着互联网技术的广泛传播与不断演进，与人们日常生活紧密相连的各个领域都在经历电子化和信息化的变革，导致对数据存储的需求不断增长。关系型数据库不仅具备存储大规模数据的能力，而且有效确保数据的一致性和完整性，同时满足数据共享和安全性等多重需求。此外，它还具备对于逻辑和物理错误的卓越恢复能力。这些特性使其成为现有互联网生态系统中不可或缺的部分，在各行各业广泛应用。

在实际的应用场景中，操作者通过应用系统提供的用户界面进行数据访问。然而，在这种数据访问方式下，用户无法以自由灵活的方式利用数据库中的数据。此外，对于任何数据访问需求的变更，都需要系统开发人员重新设计 SQL 程序和前端页面。即使对于有经验的开发人员，编写正确的 SQL 语句同样是一项具有挑战的工作，因此利用自然语言来访问数据库对于专业开发者和一般用户来说都具有极其重要的价值。

因此，Text2SQL 任务应运而生[1-3]，旨将在自然语言文本（Text）转换成结构化查询语言 SQL，为数据库系统提供自然语言查询接口。在这一背景下，用户和开发人员无需掌握 SQL 语言，可以直接使用自然语言表达数据访问需求，由查询接口自动将自然语言查询翻译成相应的 SQL 语句。采用自然语言表达数据库查询请求的方法，不仅有助于减轻开发人员的工作负担，还能为普通用户提供自然灵活的数据库访问方式，因此 Text2SQL 任务受到学术界和工业界的广泛重视。

图 1.1 展示了瓜子二手车直卖网的用户界面实例。该系统为用户提供了多个预置的选项，以帮助用户设置筛选条件。用户可以通过点击预置按钮和填写数值的方式来使用系统，但这种方式限制了用户表达个性化查询需求的灵活性。当新能源汽车的市场占有率大幅提高，汽车售卖网站必须在原有功能的基础上添加筛选动力类型功能，对网站的前后端都需要进行对应的修改。然而，当用户可以使用自然语言来表达查询需求时，应用系统无需设计新的 SQL 程序，从而大幅度降低了系统更新的成本。

图 1.1 应用系统界面实例

Figure 1. 1 An example of application system

1.2 研究概述

Text2SQL 的相关研究可以追溯到早在 1973 年[4]-[7]，当时研究者致力于创建一个查询系统，以让用户使用自然语言来访问存储月球岩石样本信息的数据库。尽管这种研究的应用场景十分受限，仅能支持特定数据库上的单表查询，但它标志着 Text2SQL 任务的研究开始受到广泛关注，并被认为具有重要的研究价值。。

与人工智能的其他领域一样，Text2SQL 的主流方法也是数据驱动的深度学习方法。为此，研究者发布了一系列基准（benchmark）数据集，其中典型的有 ATIS[8]、GeoQuery[9]、WikiSQL[10]、Spider[11]等。这些数据集不仅为模型训练和测试提供了黄金标注数据，而且分别针对不同的应用场景定义了三种具体的 Text2SQL 任务，如图 1.2 所示。

图 1.2 不同阶段的 Text2SQL 研究

Figure 1.2 Different stages of Text2SQL research

数据集 ATIS[8]和 GeoQuery[9]针对相同数据库构造了大量自然语言查询-SQL 语句对，定义了单数据库 Text2SQL 任务。其目的是为了训练深度学习模型，使其能够捕捉不同自然语言查询与对应的 SQL 语句之间的关系。这个任务相对简单，无需在不同的数据库模式之间进行泛化，通常可以通过一般的序列-序列模型获得良好的效果[12-16]。然而，当需要将模型部署到新的生产环境或者数据库模式发生变化时，这个任务要求为每个需要查询的数据库提供大量训练用的自然语言查询-SQL 语句对，并且为不同的数据库训练不同的 Text2SQL 模型。这使得开发数据库自然语言查询接口成为一项费时费力的工作，其适用范围受限。

相比之下，数据集 WikiSQL[10]和 Spider[11]定义了多数据库 Text2SQL 任务。这两个数据集中均包含多个数据库，每个数据库有多个自然语言查询-SQL 语句对。不同之处在于，前者的每个数据库仅包含单张表，而后者的一个数据库可能含有多张表，并且查询可能涉及多表连接或嵌套的情况[11]。由于多数据库多表情况更具一般性，同时 SQL 语句的编写难度更高，因此它们目前是研究的关注重点。

现有工作针对多数据库多表 Text2SQL 研究，普遍采用编码器-解码器架构[22]，并且在编码器中显式或隐含地堆叠模式链接器，用于捕捉 SQL 查询的对象-表名和列名。在

编码器中引入模式链接器的方法主要有两种，一种是基于图神经网络的方法[23-27]，另一种是基于注意力的序列建模方法[28-30]。基于图神经网络的编码器以数据库模式元素（表名、列名）和自然语言查询中的词（Token）作为图的结点，根据表名、列名与 Token 之间的字面相似性或语义嵌入表达相似性确定结点之间的边或作为边的权重。通过图神经网络的训练，模型学习得到融合了自然语言查询信息的数据库模式元素表达以及融合了数据库模式信息的自然语言查询表达。基于注意力的序列编码器[30]将数据库模式和自然语言查询线性序列化（序列中包含表名、列名、主键和外键信息以及自然语言查询中的 Token），通过 BERT[31]中的自注意力机制捕捉序列中不同类型元素之间的软对齐关系，其中表名和列名与 Token 之间的注意力权重隐含地实现了模式链接器的功能，或者通过字面相似性确定自然语言查询与数据库模式之间的对齐关系，然后在序列编码过程中编码这些对齐关系。现有工作在设计 Text2SQL 解码器时多数采用了指针网络，不同之处在于一些工作将输入编码解码成 SQL 语句的抽象语法树[32]，另外一些则直接解码成 SQL 语句的 Token 序列[30]。

上述方法在 Spider 数据集上取得了大约 70%的 SQL 语句精确匹配正确率。然而，正如 Gan 等人[33]指出的那样，这些方法严重依赖表名和列名在自然语言查询中的显式提及，以及模式链接器对这些提及的准确捕捉。Gan[33]进一步分析了上述方法在 Spider 数据集上表现良好的原因，发现 Spider 数据集是有偏的，其中被查询的表名和列名经常在自然语言查询中明确出现，而这显然不符合实际的应用场景。在实际应用中，普通用户很难精确记忆数据库模式中的表名和列名。此外，尽管现有方法利用不同的编码方式帮助模型对数据库模式和自然语言查询进行联合建模，但其多表查询翻译准确率要远低于单表查询。并且上述方法虽然在 spider 数据集的单表查询上表现较好，但它们的最终 SQL 翻译准确率仍然低于模型在数据集 WikiSQL 上的表现，而两个数据集之间的主要差异就是数据库中是否含有多张表。

综上所述，尽管现有方法虽然在 spider 数据集上取得了良好的 Text2SQL 性能，但它们过于依赖模式链接器和数据库模式的显式提及，而这在实际应用场景中是难以实现的。此外，现有方法的多表查询准确率明显低于单表查询，将模型应用在多表数据库和单表数据库时，单表查询准确率也有明显差距。因此，减少模型对模式链接器和数据库模式显式提及的依赖，并将模型从利用多张基本表来生成 SQL 语句转化为生成特定单表（视图）上的查询，将成为提高模型在实际应用场景中 Text2SQL 性能的关键。

1.3 本文主要工作

本文主要针对自然语言转 SQL 语句进行研究，首先梳理了该领域的发展脉络，并对多数据库多表 Text2SQL 研究的基准数据集 spider 以及现有方法的实验结果进行了详细分析。针对现有研究严重依赖自然语言查询对数据库模式显式提及的问题，和多表查

询的准确率远低于单表查询，以及模型在多表数据库和单表数据库上的单表查询准确率存在明显差距的问题，本文进行了深入的探讨，具体工作如下：

（1）为了帮助模型更好地捕捉数据库领域建模知识，从而能够在同物异名的实际应用场景中确定 SQL 查询对象，本文扩展了数据库模式的解释信息，作为领域知识的补充。具体而言，本文将表名和列名解析成由词和短语组成的序列，并通过词典查询序列中每个元素的解释，然后按顺序将它们拼接成一段文本，作为当前表名或列名的解释信息。模型将查询中的词、表名和列名建立成图神经网络中的结点，借助解释信息来获得数据库模式结点的嵌入表达，并通过比较自然语言查询和解释信息的语义相似性，构建带权重的边，以辅助模型识别出现在 SQL 语句中的表名和列名。在 Spider-syn 和 Spider 数据集进行了实验，实验证明本文方法在自然语言查询使用同义词提及数据库模式时，本文方法具有更好的抗同义词替换攻击鲁棒性。

（2）多数据库多表 Text2SQL 研究基准数据集 Spider 中的复杂 SQL 查询，使得现有研究聚焦于如何建模数据库模式和复杂 SQL 语句的生成。然而，本文分析了现有方法的实验结果，并发现单表查询的 SQL 翻译准确率远高于多表查询。当针对确定的表生成 SQL 语句时，模型表现出卓越的准确率，甚至超越了人类表现水平。因此，本文提出了基于视图的 Text2SQL 方法，旨在针对常见的查询场景事先创建视图。对于每一个查询，首先判断其是否能够被归结成某个视图上的查询。如果是，则本文先为其生成视图上的 SQL 骨架，然后再为其添加 from 子句等内容，以获得可执行的 SQL 语句。在 Spider-syn 和 Spider 数据集上的实验证明，生成视图上的 SQL 骨架对模型来说更加简单，能有效地提高模型的 SQL 翻译准确率。

1.4 本文组织结构

本文一共分为五个章节，每个章节的内容安排如下所示：

第一章绪论，阐述了 Text2SQL 任务的研究背景、选题意义以及研究价值，并归纳和总结了现有研究的发展路线以及不足，阐述了本文的两部分工作的主要内容和价值，在本章的最后对本文的组织结构进行了介绍。

第二章相关工作，首先介绍了 Text2SQL 任务，并详细地介绍了不同阶段的 Text2SQL 研究方法及其优缺点，然后对近期研究进行归纳和总结，分析现有工作的不足和提出本文的改进思路。

第三章提出了本文的第一部分工作，即利用词典扩展数据库模式信息的 Text2SQL 方法。该部分内容首先论述了仅利用自然语言查询和数据库模式生成 SQL 语句的局限性，然后提出了本文为表名和列名扩展解释信息来为模型补充数据库领域建模知识的思想并介绍本文模型，最后描述了本文的实验环境和使用的超参数，展示并分析了实验结果。

第四章详细介绍了本文的第二部分工作，即基于视图的 Text2SQL 方法。本章内容主要包括了该任务的出发点和从生成基本表上的 SQL 语句转为生成视图上的 SQL 骨架所带来的好处，介绍了本文模型框架和如何获得生成 SQL 骨架和生成 SQL 语句的模型，最后介绍了实验设计并分析实验结果。

第五章总结与展望，主要对本文所做的工作和取得的成果进行了总结，并对下一步的研究进行思考和展望。

第2章 相关工作

Text2SQL 任务是自然语言处理领域的研究热点之一，是实现数据库的智能接口和提高数据库分析效率的关键技术。目前 Text2SQL 的主流方法是数据驱动的深度学习方法，研究者针对三种场景创建了标注数据集并开展了相关研究：单数据库、多数据库单表和多数据库多表，本章将对三种研究场景进行详细分析。

2.1 单数据库 Text2SQL 研究

为了创建数据库的自然语言查询接口，Price 标注了 ATIS[8]数据集定义了早期的单数据库 Text2SQL 任务。单数据库场景假定被查询的数据库有大量的训练数据（Text-SQL 对），深度学习算法只需要从这些训练数据学习特定数据库的 Text2SQL 模型，无需在不同的数据库模式之间进行泛化。在这样的场景下，一般的 Seq2Seq 模型即可完成从自然语言到 SQL 语句的翻译工作，唯一的难点是模型的词汇表中可能不会包含编写 SQL 语句需要的条件值。因此研究者在一般的 Seq2Seq 模型中加入注意力机制[34]和拷贝机制[35]，采用生成加拷贝的方式来获得 SQL 语句，在 ATIS 和 GeoQuery 数据集上取得了极大的成功，达到超过 80%的翻译准确率。这种场景虽然降低了任务的复杂性，但通过这种方式训练的模型仅能应用在特定数据库上，针对新的应用环境必须重新构造自然语言查询-SQL 对作为训练样例，并重新训练的 Text2SQL 模型，使得开发数据库的自然语言接口成为一项成本极高的工作。

2.1 多数据库单表 Text2SQL 研究

随着数据库在各行各业的应用逐渐普及，针对每个数据库构造自然语言查询-SQL 语句对来训练模型的成本较高，如何将模型在不同的数据库模式和自然语言查询之间泛化成为研究的重点。因此 Zhong 标注了 WikiSQL[10]数据集定义了多数据单表 Text2SQL 任务。在该任务中，模型不仅需要拟合自然语言查询和 SQL 语句间的映射关系，还需要学习当前查询会命中哪些数据库模式。在 ATIS、GeoQuery 数据集上表现良好的模型，在 WikiSQL 数据集上只能达到 23.3%准确率，这是因为模型在不同的数据库模式之间的泛化能力较差，难以判断哪些列名与当前问题相关。

为了建立自然语言查询和数据库模式之间的对齐关系，研究者多利用注意力机制来为其获取包含全局上下文信息的嵌入表示，从而帮助模型确定在 SQL 语句中更可能出现的列名，不同方法之间的区别在于，一些研究[17,18]在获得自然语言查询和数据库模式的嵌入表达之后，再利用注意力捕捉它们之间的关联，而另一些[19-21]将查询和模式组织

成序列后，利用 BERT 中的注意力模块来捕捉不同元素之间的软对齐关系，然后将其嵌入表达输入解码器中。多数据库单表 Text2SQL 研究的基准数据集 WikiSQL 中作为标签的 SQL 语句均为单表查询且格式相对固定。因此研究者为其总结出了适用性较强的模板，采用槽填充（slot filling）的方法[17-21]来将 SQL 语句的生成任务转化成对槽（slot）进行预测的分类任务。通过这种方式训练的模型可以很好地捕捉子句之间的联系，在 WikiSQL 的测试集上取得了超越人类表现的准确率[20]。但仅能接受单张表作为输入和生成的 SQL 语句被预先定义的槽所限制等缺点，导致模型难以应用在实际场景中，因此在后续的多数据库多表 Text2SQL 研究不再采用槽填充方法来生成 SQL 语句。

2.3 多数据库多表 Text2SQL 研究

多数据库多表 Text2SQL 研究和多数据库单表 Text2SQL 研究的相同点在于，均假设相同的 Text2SQL 模型可以为不同的数据库提供自然语言到 SQL 的翻译，数据库模式和自然语言查询共同作为 Text2SQL 模型的输入，模型需要捕捉数据库模式和自然语言查询之间的关联，并在不同的数据库模式和自然语言查询之间泛化，不同之处在于，多数据库多表 Text2SQL 研究允许一个数据库含有多张表，可以多表连接或嵌套查询，因此适用面更广，是目前研究的热点，典型的数据集是 Spider[11]。

多数据库多表 Text2SQL 目前面临的主要挑战是捕捉自然语言查询与数据库模式之间的关联，确定 SQL 查询对象-表名、列名，和生成包含连接查询、嵌套查询等子句的复杂查询。为了应对上述挑战，研究者提出了一系列编码器方法，包括基于自然语言查询和数据库模式线性化的 BERT 编码方法和基于图神经网络的编码方法，然后在解码的过程中采用指针网络来直接生成 SQL 语句，或者利用基于语法[45-47]的解码器生成树形的抽象语法树 AST[32]后，再将其翻译成 SQL 语句。

在基于 BERT 的编码方法中，研究者[29,30]在自然语言查询中划分成长度为 1-6 的 gram，根据 gram 和表名、列名或者数据库中存储的值间字符匹配的结果，在序列中添加特殊字符，来利用 BERT 的自注意力层来捕捉哪些数据库模式与当前问题相关，然后利用前馈神经网络在列名的嵌入表达中融合当前列是否为主键，外键等信息，或者使用额外的多层感知机作为模式链接器[28]，根据其输出的概率为自然语言查询加权数据库模式的嵌入表达。

在基于图神经网络的编码方法中，一些研究者[23,24]将表名和列名建模成图神经网络中的结点，利用数据库模式中的主键、外键等信息在图中建立不同类型边，利用门控图神经网络(gated GNNs)[41]计算结点间相关性得分，使用标准 GRU[42]加权聚合图神经网络更新前后的语义来为结点获得全局表达(global representation)[43,44]，Wang [26]将自然语言查询中的 gram 也建模成图神经网络中的结点，来将其与数据库模式进行联合建模，利用关系感知注意力[48]更新结点的嵌入表达，从而更好地感知全图的拓扑结构，但这种

在图迭代的过程中不区分结点的局部邻居和非局部邻居的方法，可能导致过度平滑（over-smoothing）问题[49]。因此 Cao[27]在保留结点图的基础上，额外添加了以边为中心的对称图，将结点图中结点的嵌入表达作为边图中边结点的嵌入表达，在每轮图迭代的过程中，利用注意力[50]同时更新结点和边的嵌入表达，在多数据库多表 Text2SQL 研究的基准数据集 spider 上准确率超过了 70%。

这些方法的核心都是基于字面或语义相似性建立硬对齐关系，或通过数据库模式与自然语言查询之间的交互注意力建立软对齐关系的模式链接(Schema Linking)机制[36-39]。这些严重依赖模式链接的方法虽然在 Spider 数据集上表现优异，但 Gan 等人[33]发现，Spider 数据集是有偏的，被查询数据库的表名和列名在自然语言查询中明确出现，这不代表实际应用的真实情况，普通用户很难理解数据库模式，更难准确记忆表名和列名。为了验证上述方法在真实应用场景下的 Text2SQL 性能，Gan 等人用同义词替换了 Spider 数据集中，自然语言查询显式提及的表名和列名构建了 Spider-syn 数据集，并将这种操作称为同义词替换对抗攻击。他们在 Spider-syn 上测试了上述模型中的三个典型代表[23,26,29]，发现它们的 Text2SQL 准确率大大降低，说明了现有方法在 Spider 上虚高性能的假象。

现有方法为了更好地生成 spider 数据集中包含的多表查询，采用了不同的序列化方法和图神经网络方法，来对数据库模式和自然语言查询进行联合建模，帮助模型在数据库模式的嵌入表达中融合自然语言查询信息，在自然语言查询表达中融合数据库模式信息。然而无论是在能够更好地模拟实际应用场景的 Spider-syn 数据集上，还是显式提及表名和列名的 Spider 数据集上，现有方法的多表查询准确率都远低于单表查询，并且现有方法的单表查询准确率在多表数据库和单表数据库上的差距较大。

2.4 现有工作的问题

本文对现有的 Text2SQL 研究进行了分析，发现其主要存在如下的问题：

（1）现有方法仅利用用户提出的自然语言查询和给定的数据库模式来生成 SQL 语句，然而即使是专业的开发者也需要借助数据库设计文档来辅助理解数据库建模的领域知识，领域知识的缺失导致模型难以理解数据库模式的语义。

（2）现有方法为了更好地生成复杂 SQL 语句，采用序列化和图神经网络等方法来建模数据库模式，但其多表查询的准确率仍然远低于单表查询，并且单表查询的准确率也未能达到模型在多数据库单表基准数据集上的表现。

对于上述的问题，本文的改进方向如下：

（1）本文提出了利用词典扩展数据库模式信息的 Text2SQL 方法，利用词典为数据库模式元素（表名和列名）扩展解释信息，来帮助模型进一步补充数据库建模领域知识来弥补自然语言查询和数据库模式之间的语义鸿沟，并利用图神经网络对二者进行联

合建模，将表名、列名和自然语言查询中的词作为图中的结点，利用查询和解释信息之间的语义相似性在图中建立带权边，在表名和列名的嵌入表达中融合其解释信息的语义，从而帮助模型更好地判断当前自然语言查询指向了哪些数据库模式，提高 SQL 翻译准确率。

（2）本文通过分析现有研究的实验结果发现，当数据库内仅包含一张表时，模型无需为 SQL 语句生成表名和 from 子句，从而能够获得最好的 SQL 翻译准确率。因此，本文提出了基于视图的 Text2SQL 方法，利用数据库模式中的信息为常见的查询场景创建视图。如果当前查询可以被归结为视图上的查询，则本文首先为其生成特定视图上的 SQL 骨架，然后再为 SQL 骨架添加表名和 from 子句来生成 SQL 语句，从而获得更高的 SQL 翻译准确率。

2.5 本章小结

本章梳理了 Text2SQL 研究的发展脉络，对该研究的三个阶段，即单数据库、多数据库单表和多数据库多表 Text2SQL 研究进行了归纳和总结，并着重介绍了更能代表一般情况，是目前研究焦点的多数据库多表 Text2SQL 研究，分析了现有方法存在的问题并提出了本文的解决方案。

第3章 利用词典扩展数据库模式信息的 Text2SQL 方法

现有方法仅依赖数据库模式捕捉数据库建模的领域知识，然而即使是经验丰富的开发人员也需要借助数据库设计文档，而不是仅利用数据库模式来构造 SQL 语句。这是因为结构化的数据库模式表达领域知识的能力有限，导致与自然语言查询之间存在着语义鸿沟。因此，在数据库设计文档缺失的情况下，本章在词典中为表名或列名查询解释信息作为模型输入的扩展内容，使得模型在同物异名的实际应用场景中具有更高的 SQL 翻译准确率。

3.1 引言

在目前的多数据库多表 Text2SQL 中，研究者提出了一系列编码器方法[23-30]包括基于 BERT[31]编码方法和基于图神经网络的编码方法，在 Spider[11]数据集上取得了近 70% 的 Text2SQL 准确率，但 Gan[33]发现 Spider 数据集是有偏的，在实际应用场景中， 被查询数据库的表名和列名通常不会在普通用户提出的自然语言查询中明确出现，Gan[33]针对 Spider 自然语言查询中显式提及的表名和列名，人工筛选同义词替换它们，并把这种操作称为同义词替换攻击，这样替换得到的新数据集叫做 Spider-syn，发现现有方法在同义词替换攻击下 Text2SQL 效果大大下降，说明其无法适用于真实的数据库应用场景。

为了解决所发现的问题，Gan 等人提出了两种解决方案，一种是以同义词扩展数据库模式中的表名和列名，将扩展后的数据库模式作为 Text2SQL 模型的输入，另外一种是采用对抗训练的方法，以对抗样本结合正常样本重新训练模型，并实验验证了这两种方法的同义词替换鲁棒性。在 Spider-syn 上进行的实验发现前一种方法效果更好，并且具有不需要重新训练模型的优点。

实质上，在数据库应用系统开发过程中，开发人员在不理解数据库建模的领域知识的情况下很难准确构造实现一个查询功能的 SQL 语句，而数据库模式作为结构化的元数据，其提供的领域知识是很有限的，有经验的开发人员也必须依赖数据库详细设计文档才能撰写 SQL 语句。同样，Text2SQL 模型仅以数据库模式和自然语言查询作为输入，也很难生成完成相应自然语言查询的 SQL 语句。例如，图 3.1 给出的数据库仅包含表“公司（收入，净收入，预算，…）”，对该数据库的自然语言查询“贵公司明年预计投入多少钱来完成业务目标？”，无论是有经验的开发人员还是 Text2SQL 模型，只有在理解“预算”的含义（代表为未来发展的投入）后才能确定上述查询涉及字段“预算”，而数据库模式并不包含这样的应用背景知识。

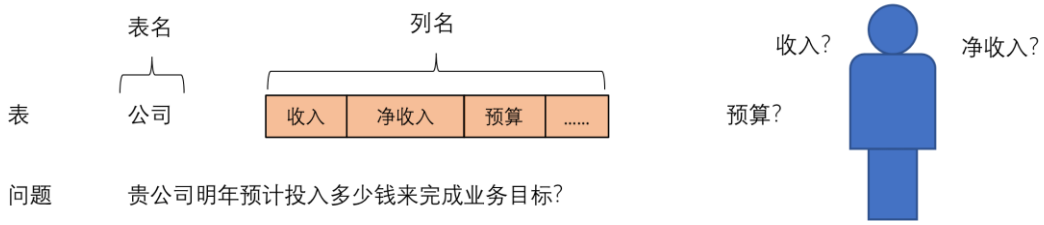

图 3. 1 应用场景实例

Figure 3.1Figure 3. 1 An example of application scenarios

本章主要针对现有研究仅利用用户提出的自然语言问句和给定的数据库模式来生成 SQL 语句，导致模型难以理解数据库模式语义的问题，本章提出利用词典扩展数据库模式信息的 Text2SQL 方法 ExSQL（Expansion Enhanced SQL），利用数据库模式中出现的 Token 的词条解释信息，丰富数据库建模的领域知识，进一步提高 Text2SQL 模型抵抗同义词替换攻击的能力，弥补数据库模式与自然语言查询之间的语义鸿沟（gap）。本章在相关数据集上的实验证明了 ExSQL 具有最好的同义词替换鲁棒性，更能适应实际的应用场景。

3.2 利用词典扩展数据库模式信息的 Text2SQL 方法 ExSQL

ExSQL 方法的模型架构如图 3.2 所示，由编码器、相似度模块、图神经网络、解码器和辅助任务五部分组成。

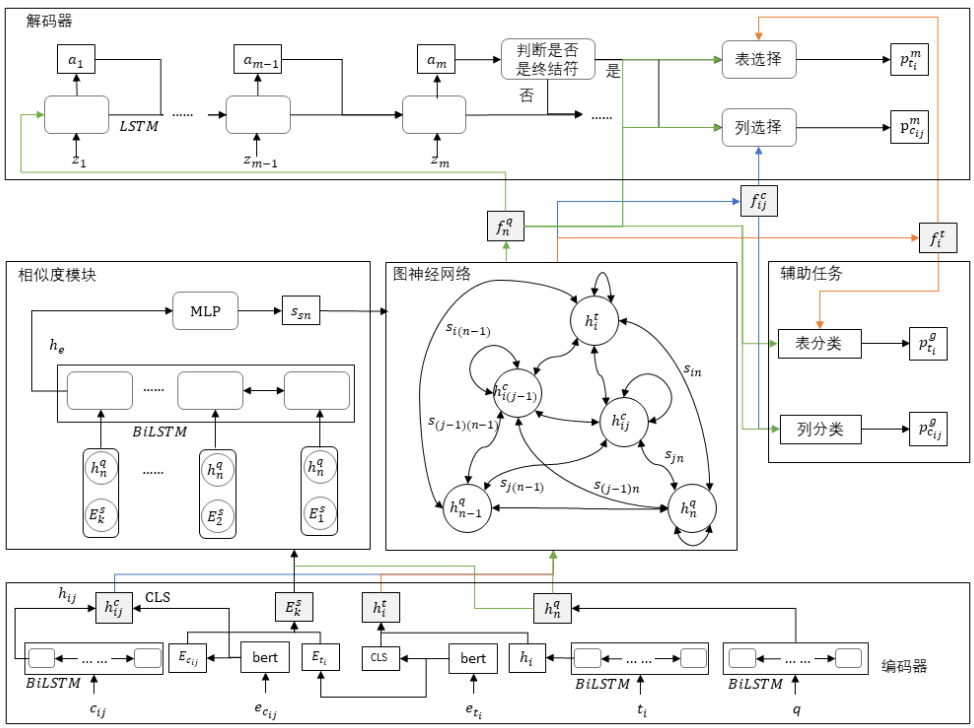

图 3. 2 ExSQL 模型架构

Figure 3. 2 Architecture of the ExSQL model

编码器将自然语言查询、数据库模式和解释信息作为输入，输出解释信息和自然语言查询的嵌入表达作为相似度模块的输入，输出表名和列名的嵌入表达作为图神经网络的输入。相似度模块利用解释信息和自然语言查询的嵌入表达获取图中边的权重，作为图神经网络模块的输入。图神经网络更新结点的嵌入表达后，将其输入解码器和辅助任务中。解码器使用有序神经元 LSTM（Ordered Neurons LSTM，On-LSTM）[51]解码生成 AST。辅助任务用来帮助模型学习哪些表名、列名更可能在 SQL 语句中出现。

3.2.1 问题定义

ExSQL 将给定的自然语言查询 Q、数据库模式 S、解释信息 E 作为输入，输出对应的 SQL 语句。自然语言查询$Q = \{q_1, \dots q_n, \dots q_{|Q|}\}$，其中|Q|表示自然语言查询中词的个数。数据库模式 S 主要包含表 T 和列 C 两部分，其中表$T = \{t_1, \dots, t_i \dots, t_{|T|}\}$, 列$C = \{c_{11}, \dots, c_{ij} \dots\}$，$|T|$代表当前数据库中数据库中表的数量，$t_i$是第$i$张表，$c_{ij}$是第$i$张表的第$j$个列.每个表$t_i$或列$c_{ij}$被进一步定义为.每个表名和列名可能由多个词组成，因此将t_i和c_{ij}进一步定义为$t_i = \{t_{i1}, \dots t_{iu}, \dots\}$, $c_{ij} = \{c_{ij1}, \dots c_{ijv}, \dots\}$,其中$t_{iu}$和$c_{ijv}$是该表名和列名中的词，然后利用$t_{i0}$和$c_{ij0}$标识当前字段为一张表或描述当前列的数据类型.。用于弥补数据库建模领域知识的解释信息$E = \{e_k^s\}$，其中上标代表第 s 个表名或列名的解释信息，下标表示当前解释信息中的第 k 个词。

ExSQL 将输入的自然语言查询和数据库模式定义成带权图$G_n = (V_n, R_n)$。图中共包含三种类型的结点，$V_n = Q \cup S = Q \cup T \cup C$，其中$Q$为自然语言结点，$T$为表名结点，$C$为列名结点。图中的边$R^n = R^{S\sim S} \cup R^{Q\sim S} \cup R^{Q\sim Q}$，其中$R^{S\sim S}$由预先定义的数据库模式决定，$R^{Q\sim S}$是通过自然语言查询和表名与列名之间的字面相似性建立起的匹配关系，$R^{Q\sim Q}$由自然语言查询决定（详见附录 A）。

3.2.2 模式信息扩展

Spider 数据集内的数据库中所有的表名和列名均使用完整英文单词命名，在词间使用下划线进行分隔。本章将表名或列名解析为由单词组成的字符串后，枚举出所有长度为 1-3 的 gram，将其按长度降序排列成一个待查询序列，然后在 oxford 词典中查询每个 gram 的解释。在当前 gram 的查询结果不为空时，本章会移除剩余的待查询序列中所有与当前查询对象存在重叠的 gram，当查询结果为空时，如果当前 gram 的长度大于 1 则继续向下查询，否则为了统一操作直接将当前 gram 本身作为解释使用，查询完成后，将所有 gram（未被移除）的解释按顺序拼接成一段文本，作为当前数据库模式元素的解释信息，解释信息的长度通常远大于原本的表名或列名。

例如列名 book_club_id，解析出的待查询序列为：book club id; book club; club id;

book; club; id;首先查询 book club id 的解释，结果为空则程序继续向下查询，book club 的查询结果不为空，所以移除 club id, book, club 三个与 book club 存在重叠部分的 gram，然后继续查询 id 的解释，最后将 book club 和 id 的解释拼接后作为当前列的解释信息。

3.2.3 编码器

编码器将自然语言查询、数据库模式和解释信息三部分作为输入，将自然语言查询和数据库模式按照如下顺序组织成查询-模式序列，并将每个表名或列名的解释信息作为独立的序列，将上述序列作为预训练模型 BERT 的输入。

$$[\mathrm{CLS}]q_1 q_2 \dots\dots [SEP] t_{10} t_1 c_{110} c_{11} c_{120} c_{12} \dots\dots t_{|T|0} t_{|T|} c_{|T|10} c_{|T|1} c_{|T|20} c_{|T|2} \dots\dots [SEP]$$

将上述查询-模式序列输入 BERT 后，每个词$w = \{w_1, \dots\dots, w_{|W|}\}$被拆分为$|w|$个子词，然后利用一个注意力层加权聚合所有子词的嵌入表达，获得查询-模式序列中每个词的语义，如式(3-1)-式(3-2)所示。

$$\alpha_i = softmax(tanh(w_i v_1)\, v_2) \tag{3-1}$$

$$w = \sum_{i=1}^{|W|} \alpha_i\, w_i \tag{3-2}$$

获得查询-模式序列中每个词的嵌入表达后，本章将自然语言查询的嵌入表达，输入双向$LSTM_1$中，拼接查询中每个词的前向和后向的隐藏层状态，作为该词也即后续图神经网络中自然语言结点的初始嵌入表达$h_n^q \in R^n$，如式(3-3)所示。

$$h_n^q = \left[\overrightarrow{LSTM}\left(\vec{h}_{n-1}^q, q_n\right); \overleftarrow{LSTM}\left(\overleftarrow{h}_{n+1}^q, q_n\right)\right] \tag{3-3}$$

图神经网络除自然语言结点外，还包含表名和列名两种类型的结点，其初始嵌入表达均由两部分组成，二者的计算方式完全相同，在下述公式中使用s表示当前字段。将表名或列名中所有词$t_i = \{t_{i1}, \dots t_{iu}, \dots\}$或 $c_{ij} = \{c_{ij1}, \dots c_{ijv}, \dots\}$的嵌入表达输入双向$LSTM_2$中，将前后向的最终时刻隐藏层状态拼接后，作为表名结点或列名结点的嵌入表达的第一部分$h_{s1} \in R^n$，如式(3-4)-式(3-5)所示。

$$h_i^s = \left[\overrightarrow{LSTM}\left(\vec{h}_{i-1}^s, s_i\right); \overleftarrow{LSTM}\left(\overleftarrow{h}_{i+1}^s, s_i\right)\right] \tag{3-4}$$

$$h_{s1} = \left[\vec{h}_N^s; \overleftarrow{h}_1^s\right] \tag{3-5}$$

对于解释信息，本章在其头尾处分别拼接 CLS 和 SEP 标识符后输入预训练模型 BERT 中，得到输出的 CLS 标识符的嵌入表达作为对应表名结点或列名结点语义表达的第二部分$h_{s2} \in R^n$，将h_{s1}和h_{s2}拼接后利用前馈神经网络g ($R^{2n} \rightarrow R^n$)将其映射回R^n维度，作为图中数据库模式结点的初始嵌入表达，如式(3-6)所示。

$$h_s = g([h_{s1}; h_{s2}]) \tag{3-6}$$

3.2.4 相似度模块

在编码器输出的解释信息中所有子词的嵌入表达E_k^s上，拼接自然语言查询中的第n

个词的嵌入表达h_n^q后，输入双向$LSTM_3$中，将前后向的最终时刻隐藏层状态拼接后输入打分函数MLP，获得当前表名或列名与自然语言查询中第n个词的语义相似度得分s_{sn}，计算方法如式(3-7)-式(3-9)所示。

$$E_k^{sn} = \left[h_n^q; E_k^s\right] \tag{3-7}$$

$$h_k^E = \left[\overrightarrow{LSTM}\left(\vec{h}_{k-1}^E, E_k\right); \overleftarrow{LSTM}\left(\overleftarrow{h}_{k+1}^E, E_k\right)\right] \tag{3-8}$$

$$s_{sn} = MLP\left(\left[\vec{h}_N^E; \overleftarrow{h}_1^E\right]\right) \tag{3-9}$$

3.2.5 图神经网络

将编码器输出的结点的嵌入表达、对预先定义的结点间关系进行编码获得的边的嵌入表达和相似度模块输出的相似度得分三部分，作为图神经网络的初始输入.在自然语言结点和数据库模式结点之间的边的嵌入表达上，乘以对应的语义相似性得分s_{sn}，从而利用带权边来影响结点间的图注意力权重.

在本章使用的图神经网络中，结点 X 由自然语言结点、表名结点和列名结点三部分组成，定义$X = \left(h_n^q; h_i^t; h_{ij}^c\right) = \{x_1 \dots x_p \dots x_q, \dots, x_{|X|}\}$，其中$x_p$和$x_q$是图中的第$p$个和第$q$个结点的嵌入表达，使用$r_{pq}$表示结点$x_p$和结点$x_q$之间的边的嵌入表达.当结点$x_p$和结点$x_q$分别是自然语言结点和数据库模式结点时，$s_{pq}$的值在 0-1 之间，否则将对应位置的$s_{pq}$置为 1，通过多头注意力[50]来衡量图中结点之间的关系α，计算方式如式(3-10)-式(3-14)所示.

$$e_{pq}^h = \frac{x_p W_Q^h \left(x_q W_K^h + s_{pq} * r_{pq}^K\right)^T}{\sqrt{d_z/H}} \tag{3-10}$$

$$\alpha_{pq}^h = softmax\left(e_{pq}^h\right) \tag{3-11}$$

$$\widetilde{x_p} = \mathop{\Big\|}_{h=1}^{H} \sum_{q=1}^{|X|} \alpha_{pq}^h \left(x_q * W_V^h + r_{pq}^V\right) \tag{3-12}$$

$$\widetilde{y_p} = LayerNorm\left(x_p + \widetilde{x_p} * W_o\right) \tag{3-13}$$

$$y_p = LayerNorm\left(\widetilde{y_p} + FFN\left(\widetilde{y_p}\right)\right) \tag{3-14}$$

在上述公式中，$\mathrm{W_Q^h}$,$\mathrm{W_K^h}$,$\mathrm{W_V^h} \in R^{n*d/H}$，$\mathrm{W_o} \in R^{n*d}$是可学习的参数，$H$是注意力头数，$FFN$表示前馈神经网络，$LayerNorm$表示归一化层[52]在图迭代的过程中，每个结点$x_p$在聚合图中其他结点的嵌入表达时采用$H$头注意力,其中一半的注意力头只聚合来自预先定义的一跳邻居结点的嵌入表达，剩余的一半注意力头聚合来自图中所有邻居结点的嵌入表达，以此来解决图神经网络的过渡平滑问题。经过 8 轮图迭代后，图神经网络输出所有结点的嵌入表达$f_p = (f_n^q; f_i^t; f_{ij}^c)$。

3.2.6 解码器

该模块利用 On-LSTM 解码生成 AST，将$z_m = [a_{m-1}; a_{p_m}; h_{p_m}; n_m]$ 作为输入，得到每个时刻的隐藏层状态h_m，其中z_m中$\mathrm{a_{m-1}}$是上一步解码生成的 action[27]、a_{p_m}是当前 action 的父 action、h_{p_m}是生成父 action 时 On-LSTM 的隐藏层状态、$\mathrm{n_m}$是待拓展结点的类型，将每个时刻的隐藏层状态h_m作为查询向量聚合图中所有结点的嵌入表达f_p获得上下文向量c，拼接h_m后输入MLP中映射回原本的维度获得$\mathrm{h_m^{att}}$，用于判断当前时间步如何拓展 AST，过程如式(3-15)-式(3-19)所示。

$$c_m, h_m = OnLSTM(z_m, c_{m-1}, h_{m-1}) \tag{3-15}$$

$$e_{mp}^h = \frac{h_m W_{cq}^h \left(f_p W_{ck}^h\right)^T}{\sqrt{d_z/H}} \tag{3-16}$$

$$\alpha_{mp}^h = softmax\left(e_{mp}^h\right) \tag{3-17}$$

$$c = \mathop{\Big\|}_{h=1}^{H} \sum_{p=1}^{|X|} \alpha_{mp}^h f_p \tag{3-18}$$

$$h_m^{att} = MLP([h_m; c]) \tag{3-19}$$

当待拓展的 action 不包含终结符时，使用式(3-20)所示的方式计算p_{a_m}，否则通过式(3-21)和式(3-22)的公式计算$p_{t_i}^m$和$p_{c_{ij}}^m$，在 AST 中添加表名和列名。

$$p_{a_m} = softmax(h_m^{att} W_R) \tag{3-20}$$

$$\zeta_{mi}^h = softmax\left(h_m^{att} W_{tq}^h\right)\left(f_i^t W_{tk}^h\right)^T \tag{3-21}$$

$$p_{t_i}^m = \frac{1}{H} \sum_{h=1}^{H} \zeta_{mi}^h \tag{3-22}$$

解码模块在上述公式中使用 H 头注意力，在计算$p_{c_{ij}}^m$时除参数外与计算$p_{t_i}^m$完全相同，在解码过程中使用如式(3-23)的负对数似然损失。

$$loss = -\sum_{p_m} log\, p_m \tag{3-23}$$

3.2.7 辅助任务模块

辅助任务模块利用带权图模块输出的图中结点的嵌入表达$f_p = （f_n^q; f_i^t; f_{ij}^c）$来判断每个表名或列名是否会出现在模型输出的 SQL 语句中，首先将表名结点或列名结点的嵌入表达f_i^t和f_{ij}^c作为查询向量聚合自然语言结点的嵌入表达f_n^q获得上下文向量c，然后将f_i^t或f_{ij}^c和c输入 biaffine[53]分类器中，计算当前表名或列名出现在 SQL 语句中的概率。在下述式(3-24)-式(3-27)中以表名为例，除参数外列名的计算和表名完全相同。

$$e_{in}^h = \frac{f_i^t W_{sq}^h \left(f_n^q W_{sk}^h\right)^T}{\sqrt{d_z/H}} \tag{3-24}$$

$$\alpha_{in}^h = softmax\left(e_{in}^h\right) \tag{3-25}$$

$$c = \left(\mathop{\Big\|}_{h=1}^{H} \sum_{n=1}^{|Q|} \alpha_{in}^{h} f_n^q W_{sv}^h \right) W_{so}^H \tag{3-26}$$

$$p_{t_i}^g = \sigma\left(f_i^t W_{s_1} c^T + [f_i^t; c] W_{s_2} + b_s\right) \tag{3-27}$$

在辅助任务中，使用如式(3-28)的交叉熵损失。

$$loss = -\sum_{t_i} [y_{t_i} log\ (p_{t_i}^g) + (1 - y_{t_i}) log\ (1 - p_{t_i}^g)] \tag{3-28}$$

3.3 实验设置

3.3.1 实验环境

本章实验均在 Ubuntu 18.04.4 上进行，使用 python 作为编程语言，模型基于 PyTorch 框架开发，具体的实验环境和开源库的版本见表 3.1。

表 3. 1 实验环境信息

Table 3. 1 Experimental environment information

事项	内容
操作系统	Ubuntu 18.04.4 LTS
编程语言	Python3.6
集成开发环境	Pycharm
第三方开源库	Pytorch 1.7.0+cu110，torchvision0.8.1+cu110
GPU	NVIIDA Ge-Force RTX 3090Ti GPU

3.3.2 实验参数

使用线性学习率预热的 AdamW [54]作为优化器，将初始学习率设置为 0.1，在训练过程中逐渐衰减为 2e-5。编码器和解码器使用的 LSTM 的丢弃率都是 0.2[55]，其余超参数见表 3.2。

表 3. 2 模型超参数

Table 3. 2 Hyperparameters setting

超参数描述	超参数值
图神经网络隐藏单元个数	512
图神经网络层数	8
LSTM 隐藏单元个数	512
MLP 隐藏单元个数	512
action 嵌入维度	128
AST 中结点类型嵌入维度	128

注意力头数	8
epoch	200
batch size	20
最大梯度范数	5

3.3.3 实验数据和评价指标

本章在 spider-syn 和 spider 两个数据集上进行模型的评估。

（1）spider 数据集中的每个自然语言查询都对应一个存储着多张表的数据库，并且训练集和验证集以及测试集中的数据对应的数据库之间没有重合，但测试集是不公开的，数据统计如表 3.3 所示。除此之外，Spider 数据集还额外提供了从 GeoQuery[9]、Restaurants[56,57]、Scholar[58]、Academic[59]、Yelp and IMDB[16]数据集中提取的 1965 条训练样例补充到数据集内，共涉及 6 个数据库。

（2）spider-syn 是由 spider 衍生出的数据集，二者的数据集规模、自然语言查询所表达的语义、执行 SQL 语句的数据库是完全相同的。区别之处在于，spider 数据集中的自然语言查询和数据库模式之间具有显式的对应关系，这种对应关系通常不会在实际的应用场景中出现，而在 spider-syn 数据集中，自然语言查询中使用的多是表名和列名的同义词。

表 3.3 数据集数据统计情况

Table 3.3 Statistics of the dataset

数据集	训练集		验证集		测试集	
	数据量	数据库数量	数据量	数据库数量	数据量	数据库数量
Spider	7000	140	1034	20	2147	40

Spider 数据集提供的测试集是不公开的，因此 Gan 等人[33]创建的 spider-syn 数据集不包含测试集。同理，本章也无法为测试集中的表名和列名扩展解释信息，在原验证集上进行模型的评估。本章划分出了原训练集的 17 个数据库及其对应的数据作为验证集，用来调整模型的参数，新的数据划分情况如表 3.4 所示。

表 3.4 重新划分后的数据集数据统计情况

Table 3.4 Statistics of the data set after repartitioning

数据集	训练集		验证集		测试集	
	数据量	数据库数量	数据量	数据库数量	数据量	数据库数量
Spider	6274	123	726	17	1034	20

Spider 数据集提供了两种评价指标来评估模型的性能，分别是精确匹配准确率(Exact Match，EM)和执行准确率(Execution Accuracy，EA)。

（1）使用精确匹配准确率 EM 作为评价指标，需要利用 Spider 数据集提供的评估程序将模型生成的 SQL 语句和黄金标注 SQL 语句转化成特殊的数据结构[11]，来判断模型的预测结果是否正确，值得注意的是这种评价方法对 SQL 语句中的 value 部分不敏感（sensitive），因此即使被评估程序判定为正确的 SQL 语句，其执行结果可能和标注 SQL 语句执行结果并不相同。现有研究将 Text2SQL 任务视为一种特殊的序列到序列任务，多使用 EM 作为评价指标，本章实验同样如此。

（2）执行准确率评价指标 EA 要求模型生成的 SQL 语句和作为标签的 SQL 语句在执行后获得的结果集相同，即 SQL 引擎对其进行语法分析时形成的抽象语法树相同。但不同的抽象语法树也可能获得相同的结果集。

3.4 实验分析

为了验证本章算法是否能帮助模型获得更好的实际应用效果，本章在 spider-syn 和 spider 两个基准数据集上进行了实验。

3.3.4 基线模型

为了验证模型的效果，本章在 spider-syn 和 spider 两个基准数据集上与以下基线模型进行比较。

GNN[23]：使用图神经网络编码数据库模式，在自然语言查询和表名与列名之间通过注意力进行交互。

Global-GNN[24]：利用表名与列名的嵌入表达选择可能出现在 SQL 语句中的 top-K 后，利用自然语言查询的语义对候选表名与列名进行重排序。

IRNet [29]：在自然语言查询中划分 gram 和表名与列名进行字面匹配，辅助识别出现在 SQL 中的表名和列名，并采用基于 AST 的解码器解码。

RAT-SQL[26]：将自然语言查询和表名与列名都看成图中的结点，利用关系感知注意力辅助图编码。

BRIDGE[30]：将自然语言查询和表名与列名拼接成一个序列，并根据被问句提及的数据库中的值在序列中添加特殊字符来捕捉当前查询和数据库模式间的关联。

LGESQL[27]：利用线性图挖掘问题、表、列之间的关系，而无需事先定义元路径。

S2SQL[25]：将自然语言查询中的词间依存关系建模成图中的边，并引入解耦约束解决边的耦合问题。

RoSQL[33]：创建了 spider-syn 数据集，并提出通过对抗训练和创建同义词词表两种方法提高模型的鲁棒性。

ETA[60]：挖掘了预训练模型学习领域基础知识的能力，结合其提出的“擦除唤醒”机制学习输入和标签之间的关联。

3.3.5 抗同义词替换攻击效果

为了验证模型的有效性，本章和现有的基线模型进行了精确匹配准确率（EM）的对比，其中下标 L 表示 large-BERT，ManualMAS 和 AutoMAS 是 Gan 等人[33]为数据库模式扩展的同义词词表，ManualMAS 是人工将测试集的自然语言查询中使用的同义词加入词表中，是实际应用场景中无法达到的上界，AutoMAS 是利用预训练模型自动创建的词表。在用户提出的自然语言查询更能反映实际应用场景的 spider-syn 数据集上的实验结果如表 3.5 所示，在 spider 数据集上的实验结果如表 3.6 所示。

表 3. 5 spider-syn 数据集实验结果

Table 3. 5 Experimental results on spider-syn dataset

Model	EM
GNN + ManualMAS	38.20%
IRNet + ManualMAS	39.30%
RAT-SQL$_L$	48.20%
S2SQL + Grappa	51.40%
Rosql + AutoMAS	58.70%
Rosql + ManualMAS	59.82%
ETA$_L$+CTA	60.40%
ExSQL(ours)	**60.83%**

从表 3.5 spider-syn 数据集实验结果可知，在受到了同义词替换攻击后，本章提出的 ExSQL 达到了最先进的性能，与 Rosql + AutoMAS 相比取得了 2.13%的性能提升（此处的 Rosql 是使用 Gan 等人[33]提出的利用同义词扩展数据库模式方法重新训练的 LGESQL，Gan 等人在论文中并没有提出新模型，且 LGESQL 的性能优于其在论文中实验的模型）。这说明现有模型对数据库模式显式提及的依赖，而本章模型利用解释信息来丰富数据库建模的领域知识，弥补数据库模式与自然语言查询之间的语义鸿沟极大地提高了模型抵抗同义词替换攻击的能力。本章模型性能已经超过了 Rosql 和 ManualMAS 同时使用达到的性能上界，说明弥补自然语言查询和数据库模式间的语义鸿沟才是 Text2SQL 效果的关键。

表 3. 6 spider 数据集实验结果

Table 3. 6 Experimental results on spider dataset

Model	EM
GNN	48.50%
Global-GNN	52.70%
IRnet	61.90%

RAT-SQL	62.70%
ETA	64.50%
BRIDGE	65.50%
LGESQL	67.60%
ExSQL(ours)	67.99%
S2SQL + RoBERTa	**71.40%**

从表 3.6 spider 数据集实验结果中可以看出，本章提出的 ExSQL 与 LGESQL 相比仅取得了 0.39%的性能提升，这是因为现有模型利用模式链接器在表名和列名被自然语言查询显式提及时，可以准确地判断 SQL 查询对象，即表名和列名，因此 ExSQL 获得的提升较小。S2SQL[25]虽然利用比 BERT 效果更好的预训练模型 RoBERTa[61]在 spider 数据集上达到了 71.40%的性能，但在受到了同义词替换攻击后性能大大下降。本章主要解决的问题就是在自然语言查询使用同义词提及表名和列名时，如何帮助模型明确用户的查询意图。因此，本章提出的 ExSQL 在 spider-syn 数据集上达到最先进的性能已经可以证明其效果。

3.3.6 消融实验

为了验证模型中各个模块的效果，本章在 spider-syn 和 spider 两个基准数据集上进行了一系列的消融实验，实验结果如表 3.7 和表 3.8 所示。本章对模型中使用的各个模块进行了如下命名:

S：在自然语言查询和表名与列名之间，利用字面相似性进行模式链接。

SIM：利用自然语言查询和表名与列名的解释信息计算语义相似度得分，作为对应结点间的边的权重。

MER：在通过表名和列名获得的嵌入表达中，融合其解释信息的语义。

ExSQL-S：在 spider 数据集上，为了更好地模拟实际的应用场景，不使用字面相似性进行模式链接。

表 3. 7 spider-syn 数据集消融实验

Table 3. 7 Ablation experiments on spider-syn dataset

Model	EM
ExSQL w/o SIM	59.19%
ExSQL w/o MER	59.62%
ExSQL w/o AutoMAS	59.57%
ExSQL	**60.83%**

在表 3.7 中展示了 spider-syn 数据集消融实验结果，取消 SIM 模块导致了 1.64%的性能下降，说明利用本章为数据库模式扩展的解释信息可以帮助模型弥补自然语言查询

和数据库模式间的语义鸿沟，进而判断哪些数据库模式与当前问题更加相关。取消 MER 模块后模型的性能下降了 1.21%，说明在数据库模式的嵌入表达中融合解释信息的语义然后利用注意力隐式捕捉自然语言查询和数据库模式间的相关性对模型的帮助低于显式建模的 SIM 模块。取消 AutoMAS 模块后模型的性能下降了 1.26%，说明其可以起到类似表名和列名在自然语言查询中的显式提及的作用，也间接证明了这种显式提及确实是现有模型在 Spider-syn 和 Spider 数据集上性能差距的原因。

表 3. 8 spider 数据集消融实验

Table 3. 8 Ablation experiments on spider dataset

Model	EM
ExSQL-S w/o SIM	65.09%
ExSQL-S w/o MER	64.75%
ExSQL-S	**65.38%**

在实际的应用场景中，自然语言查询和数据库模式间难以建立显式的对应关系。因此，本章在 spider 数据集上取消了利用字面相似性的模式链接器，验证本章提出的 SIM 模块和 MER 模块在这种情况下起到的作用。在表 3.8 的实验中，取消 SIM 模块和 MER 模块，分别导致了 0.29%和 0.63%的性能下降，说明在数据库模式被自然语言查询显式提及时，无论是利用字面相似性的硬对齐方法还是利用注意力的软对齐方法都能为模型识别查询对象提供较大帮助，进而导致了本章提出的解释信息在 Spider 数据集上起到的作用较小。

3.3.7 鲁棒性验证

为了探索在 spider 数据集上受到破坏模式链接的攻击后， ExSQL 和其他模型的性能下降情况，本章进行了如图 3.3 所示的实验，图中的-S 表示取消利用字面相似性的模式链接器。

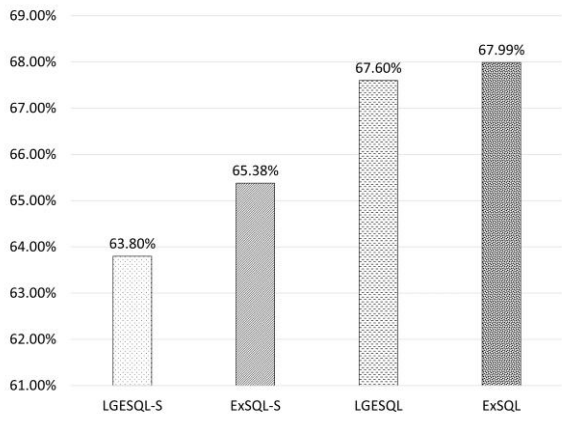

图 3. 3 鲁棒性验证

Figure 3. 3 Robustness verification

ExSQL 取得了 67.99%的性能，仅比 LGESQL 的 67.60%提高了 0.39%，然而在表 3.5 中展示的 Spider-syn 数据集实验结果中，ExSQL 与使用了 ManualMAS 词表达到理论性能上界的 LGESQL 相比性能提高了 1.01%，这说明与扩展同义词词表方法相比，本文提出的为表名和列名扩展解释信息来弥补数据库模式和自然语言查询之间语义鸿沟的方法具有更好的同义词替换鲁棒性。在取消了利用字面相似性的模式链接器后，ExSQL-S 和 LGESQL-S 分别取得了 65.38%和 63.80%的效果，说明 LGESQL 在 spider 数据集上取得的虚高的性能严重依赖模式链接器，而本章提出的 ExSQL 对模式链接的依赖较小，具有更好的 Text2SQL 实际应用鲁棒性。

3.5 总结

针对现有方法在新的基准数据集 spider-syn 上受到同义词替换攻击后性能下降的问题，本章提出了利用词典扩展数据库模式信息的 Text2SQL 方法，为表名和列名扩展解释信息来弥补数据库模式和自然语言查询之间的语义鸿沟。通过在 spider-syn 和 spider 数据集上的一系列对比实验和消融实验，验证了本章模型的有效性。

本章在领域通用词典中为表名和列名扩展的解释信息，会不可避免地引入噪声。例如词 mouse，在和动物领域有关的数据库中更可能表示“老鼠”的语义，而在和电脑，办公用品等领域有关的数据库中更可能表示“鼠标”的语义。因此，下一步工作从研究的角度考虑，可以尝试利用数据库建模的领域知识对扩展的解释信息进行过滤，从应用的角度考虑，可以通过人工为数据库模式扩展更准确的解释信息，帮助模型获得更优异的性能。

第4章 基于视图的 Text2SQL 方法

现有 Text2SQL 研究中仅涉及单张表的 SQL 翻译准确率远高于多表查询，这是因为当查询对应的数据库模式仅有一张表时，模型无需学习表和列的包含关系，以及表名会在 SQL 语句的哪些位置出现。因此，本章提出了基于视图的 Text2SQL 方法，将能够归结到视图上的查询，从生成基本表上的 SQL 语句转化为生成视图上的 SQL 骨架，来提高模型的 SQL 翻译准确率。

4.1 引言

多数据库多表 Text2SQL 研究的基准数据集 spider[11]中，无论作为标签的 SQL 语句是单表查询还是多表查询，数据库内都包含多张表。目前的研究通常采用图神经网络方法[23-27]和序列化方法[28-30] 来建模数据库模式。在图神经网络方法中，研究者利用自然语言查询与表名、列名间的字面匹配结果在图中建立边，然后，通过图注意力机制为结点加权聚合邻居结点和相关边的嵌入表达，以更好地捕捉图的拓扑结构。在序列化方法中，现有方法利用 BERT 中的自注意力机制来捕捉自然语言查询和数据库模式之间的软对齐关系。在解码的过程中，研究者[23-30]利用中间语法树 AST[32]或指针网络来生成格式不被槽约束的 SQL 语句。

虽然上述方法能提高模型生成多表查询的能力，但最终的翻译准确率仍然明显低于单表查询。以 Spider 数据集中，在不使用付费大语言模型（GPT-4 等）的情况下，性能最佳的模型 RESDSQL[65]为例。在本章中，将 SQL 查询划分为三种类型，其中"single"代表单表查询，"nature"为自然连接查询，"other"代表除自然连接查询外的其余多表查询。这些不同类型的查询在 SQL 翻译准确率方面的表现如图 4.1 所示。

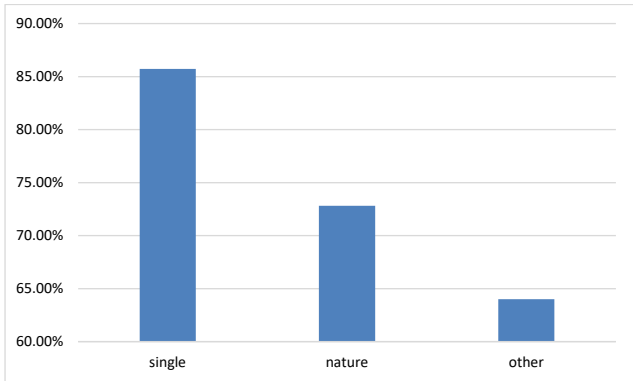

图 4.1 RESDSQL 在 spider 数据集上不同查询类型的准确率

Figure 4.1 Accuracy of different query types of RESDSQL on spider dataset

在 spider 数据集上，RESDSQL 模型在不同查询类型上的性能表现有所不同。它在

单表查询"single"的准确率最高，自然连接查询"nature"的准确率次之，其余多表查询"other"的准确率最低。这是因为相比于单表查询，模型在生成多表查询时需要进行更加复杂的表名和列名的选择。值得注意的是，自然连接查询的预测准确率在多表查询中相对较高。这是因为根据数据库设计原则，设计者通常将同一实体的不同属性分布在多张表中，通过主键和外键来标识父表和子表间的关系，从而减少存储冗余和提升存储效率。因此，在生成自然连接查询时，模型能够更好地利用数据库模式中的主键和外键信息，更加准确地确定哪些表名和列名构成了当前查询的连接条件。

在这三种不同查询类型中，单表查询"single"在性能上表现最好，但仍然与 SeaD[62] 在 WikiSQL 数据集上达到的 93%准确率相差较远。这是因为 spider 数据集的数据库模式中包含多张表，学习表和列的包含关系给模型带来了额外的挑战。一旦表名和列名的对应关系错误，就会导致模型生成的 SQL 语句不可执行。而 WikiSQL 数据集用于多数据库单表 Text2SQL 研究，每个数据库内仅包含一张表。因此，无论是采用何种方式生成 SQL 语句，模型都只需进行列名的选择，而无需进行表名的预测。由此可知，与多表查询相比，现有研究更擅长生成单表查询，当数据库内仅包含一张表时，模型的 SQL 翻译准确率最高。

因此，为了提高模型的 SQL 翻译准确率，本章提出了基于视图的 Text2SQL 方法 ViSQL（SQL Based On View），将模型从生成基本表上的 SQL 语句转化为生成视图上不包含表名和 from 子句的 SQL 骨架。鉴于事先定义的视图无法覆盖所有的查询场景，本章首先针对实际应用场景中最常见的单表查询和自然连接查询建立视图。在多数据库多表 Text2SQL 研究的基准数据集 spider 中，不同类型查询的分布如图 4.2 所示。除单表查询"single"和自然连接查询"nature"外的其他类型多表查询"other"仅占 12.7%，这也证明了单表视图和自然连接视图能够覆盖大部分查询场景。

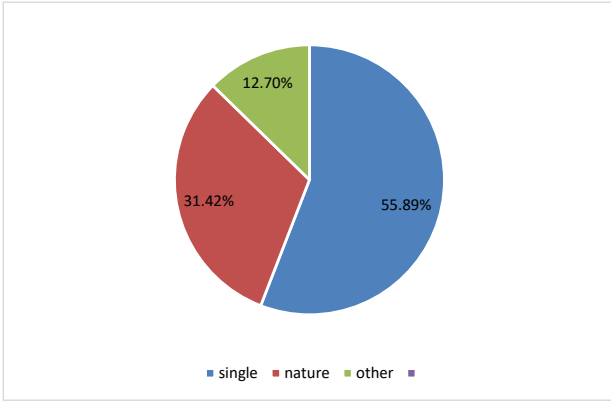

图 4.2 spider 数据集上不同类型的 SQL 查询分布

Figure 4.2 Distribution of different types of SQL queries on spider dataset

通过从生成基本表上的查询转化为生成视图上的 SQL 骨架，模型的输入序列中只需要包含对应视图的列名，，而不必涵盖整个数据库模式。这意味着模型不需要学习表

和列的包含关系，也不需要考虑表名在 SQL 语句中的具体位置。因此，待生成序列的复杂程度大大降低，如图 4.3 所示。

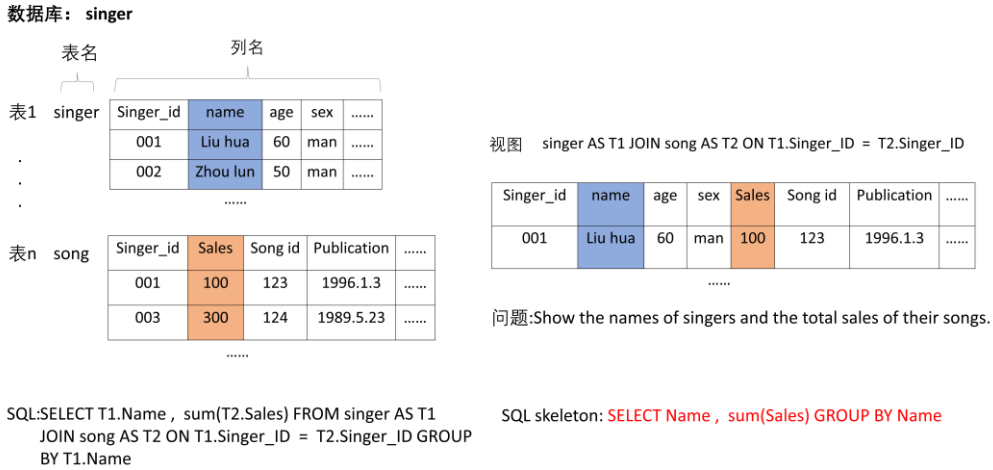

图 4.3 SQL 转化实例

Figure 4.3 An SQL conversion instance

4.2 基于视图的 Text2SQL 方法 ViSQL

ViSQL 方法的模型架构如图 4.4 所示，由 Text2SQLSkeleton 模型和 Text2SQL 模型组成。

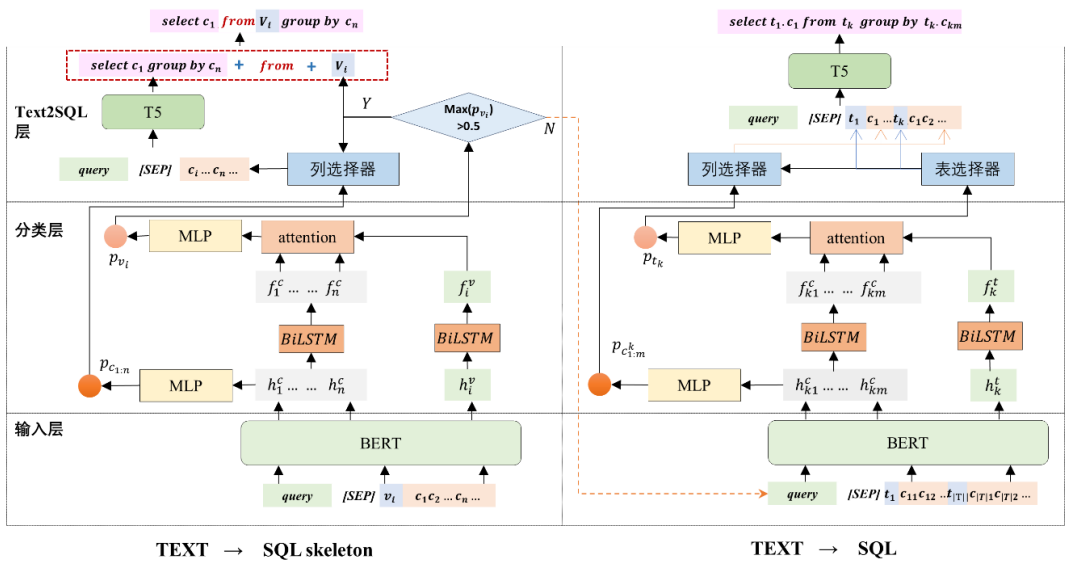

图 4.4 ViSQL 模型框架

Figure 4.4 Architecture of the ViSQL model

Text2SQLSkeleton 模型和 Text2SQL 模型都包含三部分：输入层、分类层和 Text2SQL

层。Text2SQLSkeleton 模型首先对查询进行分类，以确定其是否属于事先创建的视图上的查询。对于视图查询，Text2SQLSkeleton 模型首先生成视图上的 SQL 骨架，然后添加表名和 from 子句等细节信息，来逐步构建完整的 SQL 语句。非视图查询由 Text2SQL 模型负责直接生成 SQL 语句。

4.2.1 问题定义

ViSQL 将给定的自然语言查询$query$和数据库模式S作为输入，其中$query = \{q_1, \dots \dots q_{|Q|}\}$共$|Q|$个词，$S$中主要包含表名$T = \{t_1, \dots, t_k \dots, t_{|T|}\}$、列名$C = \{c_{11}, \dots, c_{km} \dots\}$和利用主键、外键对$K = \{(p_1, f_1), \dots \dots, (p_x, f_x), \dots \dots\}$创建的虚拟表$V = \{v_1, \dots, v_i \dots\}$，其中，$|T|$代表当前数据库中表的数量，$t_k$是第$k$张表，$c_{km}$是第$k$张表的第$m$个列，$C_{v_i} = \{c_1, \dots, c_n \dots\}$，$c_n$是第$i$个视图中的第$n$个列。

4.2.3 视图创建

本章首先考虑将实际应用场景中最常见的查询类型转化为视图上的查询，即单表查询和自然连接查询，并为其创建了基本表和虚拟表两种视图。对于基本表，本章直接将表名作为当前视图的视图名V_i，然后将经过分词和词形还原后的视图名v_i作为视图分类器的输入，同时将该表包含的列作为视图的列。对于虚拟表，本章将主键和外键对(p_x, f_x)对应的表t_1和表t_k的表名经过词形还原，去重和拼接后作为视图分类器的输入v_i，然后将V_i: $t_1\ join\ t_k\ on\ t_1.p_x = t_k.f_x$作为视图名，用于组成 from 子句，填入 Text2SQLSkeleton 模型生成的 SQL 骨架中生成 SQL 语句。随后，将表t_1和表t_k中的列作为视图的列。当p_1列和f_1列名相同时只保留一个，而当表t_1和表t_k表中包含其他同名的列时，分别在列名前拼接对应基本表的表名。

4.2.4 Text2SQLSkeleton 模型

Text2SQLSkeleton 模型首先对查询进行分类，来确定其是否为视图上的查询。在确定其为视图上的查询后，本章将该视图的列名作为 Text2SQLSkeleton 模型的输入，生成视图上的 SQL 骨架。然后，将表名和 from 子句等细节信息添加到骨架中，以生成最终的 SQL 语句。本章利用那些能够归结到视图上的查询来训练 Text2SQLSkeleton 模型。

4.2.4.1 输入层

输入层将自然语言查询和本章创建的视图作为输入，针对每个视图，为其组织如式（4-1）所示的序列：v_i和c_n分别代表经过分词和词形还原的视图名和列名。将这个序列输入预训练模型 BERT 后，从而到序列中所有子词的嵌入表达。

$$[CLS]query[SEP]v_i c_1 c_2 \dots \dots c_n \dots \dots [SEP] \tag{4-1}$$

4.2.4.2 分类层

Text2SQLSkeleton 模型的分类层，又称为视图分类器，本章将视图名v_i的所有子词的嵌入表达h_i^v和列名c_n的所有子词的嵌入表达h_n^c，分别输入到双向$LSTM_1$和双向$LSTM_2$中，如式(4-2)和(4-3)。其中，v_{ij}和c_{nj}分别代表当前视图名或者列名中第j个子词的嵌入表达，将前向和后向的最终时刻隐藏层状态拼接后经过池化层，获得视图名的嵌入表达f_i^v和列名的最终嵌入表达f_n^c，如式(4-4)和(4-5)所示。

$$h_i^v = \left[\overrightarrow{LSTM}\left(\vec{h}_{i-1}^v, v_{ij}\right); \overleftarrow{LSTM}\left(\overleftarrow{h}_{i+1}^v, v_{ij}\right)\right] \tag{4-2}$$

$$h_n^c = \left[\overrightarrow{LSTM}\left(\vec{h}_{n(j-1)}^c, c_{nj}\right); \overleftarrow{LSTM}\left(\overleftarrow{h}_{n(j+1)}^c, c_{nj}\right)\right] \tag{4-3}$$

$$f_i^v = g\left(\left[\vec{h}_N^v; \overleftarrow{h}_1^v\right]\right) \tag{4-4}$$

$$f_n^c = g\left(\left[\vec{h}_N^c; \overleftarrow{h}_1^c\right]\right) \tag{4-5}$$

获得视图名和列名的嵌入表达后，本章将f_i^v作为查询向量，利用注意力机制对当前视图内的所有列名和视图名的嵌入表达进行加权聚合，从而得到视图名的最终嵌入表达x_i^v，如式(4-6)和(4-7)所示，式(4-7)中的f_n为对应表名或列名的嵌入表达。

$$\alpha_{in} = sfotmax(\exp\left(f_i^v, f_n^c\right) / \sum \exp\left(f_i^v, f_n^c\right)) \tag{4-6}$$

$$x_i^v = \sum \alpha_{in} f_n \tag{4-7}$$

本章将视图名和列名的最终嵌入表达x_i^v和h_n^c分别输入到分类器，计算p_{v_i}和p_{c_n}。其中，p_{v_i}表示当前视图包含的信息能够回答当前自然语言查询的概率，p_{c_n}表示第n列与当前自然语言查询相关的概率，如式(4-8)和(4-9)所示。

$$p_{v_i} = MLP(x_i^v) \tag{4-8}$$

$$p_{c_n} = MLP(h_n^c) \tag{4-9}$$

由于不同数据库中视图的数量是不同的，因此本章将其转化为二分类任务来判断每个视图或列名是否与当前查询相关。在训练模型时，本章采用了交叉熵损失函数，这是分类任务中常用的损失函数。然而，与当前查询相关的视图或者列的数量往往远小于无关的数量，导致标签分布非常不平衡，负例的数量是正例的数倍。这种不平衡会导致严重的训练偏差，影响模型的性能。为了缓解这个问题，本章使用如式（4-10）的 focal loss[66]损失函数来提高模型处理不平衡数据分类的表现。

$$loss = -\alpha_t (1 - p_{y_t})^{\gamma} y_t log\ (p_{y_t}) \tag{4-10}$$

如果视图包含的信息能够回答当前查询，则本章认为当前查询是该视图上的查询。然而，当该查询为单表查询时，由于所有由该表创建的虚拟表都包含这张表的全部信息，因此可能出现判断上的歧义。为了更准确地进行判断，本章采用基于规则的方法对视图预测结果进行二次筛选。具体而言，当一个查询既可以被认为是基本表上的查询，又可以被认为是虚拟表上的查询时，将该查询判定为基本表上的查询，然后选择当前视图与自然语言查询最相关的 10 个列，作为 Text2SQL 层的输入，以生成最终的 SQL 语句。

4.2.4.3 Text2SQL 层

Raffel 提出的 T5[63]模型为 NLP 预训练模型领域提供了一个通用框架，该框架可以将任何自然语言处理任务都转化成 Text-to-Text（文本到文本）任务，因此在 Text2SQL 领域取得了极大的成功。本章利用 T5 模型来生成 SQL 骨架（skeleton）和 SQL 语句。T5 模型共有三种版本，分别是包含 12 层和 60M 参数的 small 版本，包含 12 层和 200M 参数的 base 版本和包含 12 层和 770M 参数的 large 版本，本章使用 base 版本的 T5 进行实验。

本章将如式（4-11）的序列作为 SQL 骨架生成模型的输入，其中$query$表示当前自然语言查询，c_m代表与该查询相关联的视图中的列。经过排序和筛选，本章选择与当前查询最相关的 10 个列，这样的处理方式不仅可以过滤掉不相关的模式项，同时有序的模式序列可以帮助模型捕获用于模式链接的潜在位置信息。

生成序列如式（4-12）所示，其中在‘|’前的部分是不包含列名的中间结果，‘|’后的部分才是完整的 SQL 骨架。通过这种方式，模型将 SQL 子句的生成和数据库模式的选择解耦，从而有效地降低模型的生成难度。在获得 SQL 骨架后，本章在 select 子句后添加 from 子句，以生成最终可执行的 SQL 语句。

$$[CLS]query[SEP]c_1 c_2 \dots\dots c_m \dots\dots \tag{4-11}$$

$$select\ \_\ group\ by\ \_\ |\ select\ c_1\ group\ by\ c_m \tag{4-12}$$

4.2.5 Text2SQL 模型

对于无法归结到视图上的查询，本章将自然语言查询和数据库模式作为 Text2SQL 模型的输入，以生成相应的 SQL 语句。本章利用训练集中的所有数据对 Text2SQL 模型进行训练，以提高模型的准确性和适应性。通过这种方式，模型可以更有效地处理复杂的查询请求，提高数据库的查询效率。

4.2.5.1 输入层

输入层将自然语言查询$query$和数据库中所有的表名T和列名C组织成如式(4-13)的序列，其中$query$代表自然语言查询，t_k代表数据库中的第k张表，c_{km}代表第k张表的第m个列，使用‘|’作为分隔符，用来区分不同的表。随后，将该序列输入预训练模型 BERT，获得所有被拆分后的子词的嵌入表达。

$$[CLS]query|t_1 : c_{11}, c_{12} \dots, |t_k \dots c_{km}, \dots\dots [SEP] \tag{4-13}$$

4.2.5.2 分类层

本章分别将表名和列名中所有子词的嵌入表达输入双向$LSTM$中，以获得表名和列名的初始嵌入表达h_k^t和h_{km}^c，如式(4-15)和(4-16)所示。然后，将前向和后向的最终时刻

隐藏层状态拼接，并映射回原本维度，如式(4-16)和(4-17)所示。通过这样的处理，模型可以更好地捕捉和利用表名和列名中的上下文信息，为后续的模式匹配和 SQL 生成提供更为精准的表达。

$$h_k^t = [\overrightarrow{LSTM}(\vec{h}_{k-1}^t, t_k); \overleftarrow{LSTM}(\overleftarrow{h}_{k+1}^t, t_k)] \tag{4-14}$$

$$h_{km}^c = [\overrightarrow{LSTM}(\vec{h}_{k(m-1)}^c, c_{km}); \overleftarrow{LSTM}(\overleftarrow{h}_{k(m+1)}^c, c_{km})] \tag{4-15}$$

$$f_k^t = g([\vec{h}_N^t; \overleftarrow{h}_1^t]) \tag{4-16}$$

$$f_{km}^c = g([\vec{h}_N^c; \overleftarrow{h}_1^c]) \tag{4-17}$$

用户提出的自然语言查询可能没有提及表名。因此，本章将表名的嵌入表达f_k^t作为查询向量，利用多头注意力机制对属于当前表的所有列名的最终嵌入表达f_{km}^c和当前表名的嵌入表达f_k^t进行加权聚合，获得表名的最终嵌入表达x_k^t。随后，将表名和列名的最终嵌入表达分别输入到分类器中，来判断当前表名和列名是否会在 SQL 语句中出现，如式(4-18)-(4-20)所示。

$$p_{c_m^k} = MLP(f_{km}^c) \tag{4-18}$$

$$x_k^t = MultiheadAttention(f_k^t; f_{km}^c) \tag{4-19}$$

$$p_{t_k} = MLP(f_k^{t\prime}) \tag{4-20}$$

获得表名和列名与当前查询相关的概率后，本章选择概率最高的 4 个表，并从每张表取概率最高的 5 个列，作为 Text2SQL 层的输入。

4.2.5.3 Text2SQL 层

本章将如式（4-21）的序列作为 SQL 生成模型的输入，其中包括当前自然语言查询 query、与当前查询相关的基本表t_k以及属于表t_k且与当前查询相关的列名。表名和列名同样按照模式分类器输出的概率从高到低进行排序。输出如式（4-22）的序列，在‘|’后的部分即为可执行的 SQL 语句。

$$[CLS]query[SEP]t_1 c_1 c_2 \ldots\ldots | t_k \ldots\ldots \tag{4-21}$$

$$select\ \_\ from\ \_\ group\ by\ \_ \mid select\ t_1.c_1\ from\ t_k\ group\ by\ t_k.c_{km} \tag{4-22}$$

4.3 实验设置

4.3.1 实验环境

本章程序均使用 python 编写，模型基于 PyTorch 框架开发，在 Ubuntu 18.04.4 环境下利用 GPU 进行本文实验，实验中使用的开源库版本如表 4.1 所示。

表 4.1 实验环境信息

Table 4.1 Experimental environment information

事项	内容
操作系统	Ubuntu 18.04.4 LTS
编程语言	Python3.8.5
集成开发环境	Pycharm
第三方开源库	Pytorch1.11.0，Numpy1.23.5，torchvision0.12.0
GPU	NVIIDA Ge-Force RTX 3090Ti GPU

4.3.2 实验数据和评价指标

为了验证 ViSQL 方法的有效性，本章将 spider-syn 和 spider 数据集中的数据划分为三类：单表查询"single"，涉及两张表的自然连接查询"nature"，和其他多表查询"other"。

由于 spider 测试集不公开，所以本章无法在测试集的数据库中事先创建视图，只能在原验证集上测试模型的性能，并将原训练集的 17 个数据库及其对应的数据作为验证集，用来调整模型的参数。新的训练集、验证集和测试集中不同类型数据的分布情况如表 4.2 所示。

表 4.2 数据集中各数据类型的分布情况

Table 4.2 Distribution of different types of data in the dataset

数据集	Single	Nature	Other	All
训练集	3624	1500	1337	6461
验证集	291	151	97	539
测试集	575	264	195	1034

在创建视图时，本章可能采用了与数据集提供的黄金标注 SQL 语句中不同的连接条件写法。这种差异可能导致 spider 数据集提供的评估程序无法给出正确的评估结果。因此，本章选择以执行准确率 (EA) 作为评估模型性能的指标，如果模型生成的 SQL 查询与黄金标注 SQL 查询的执行结果集相同，则认为模型生成的 SQL 语句是正确的。

4.3.3 实验设置

本章分开训练分类层模型和 Text2SQL 层模型，但都采用同样的训练策略，学习率在前 10%的训练轮数线性增长（linear warm-up）到最大，然后逐渐余弦衰减（linear warm-up）。Text2SQL 层的 T5 模型初始学习率为 1e-4，batch size 为 16，其余参数使用默认值。

分类层模型选择 AdamW[54]作为优化器，使用 dropout 策略来防止模型过度拟合，

在模型训练的过程中对 BERT large 进行微调，其他超参数设置如表 4.2 所示，其中 α 和 γ 是损失函数 focal loss 中的参数。

表 4.3 超参数设置

Table 4. 3 The hyperparameters settings

超参数描述	超参数值
Epoch	128
学习率	1e-5
注意力头数	8
Dropout rate	0.2
α	0.75
γ	2
Beam size	8
batch size	16

4.4 实验分析

4.4.4 基线模型

现有研究均使用模型在测试集上所有数据的准确率作为评价指标，而不会分开统计不同类型测试数据的准确率。本章选择目前不使用付费大语言模型（GPT-4 等）的模型中，性能最好的 RESDSQL 作为本章的基线模型，并重新统计了其在 spider-syn 和 spider 数据集上不同类型 SQL 语句的执行准确率 EA。

RESDSQL[65]：采用 pipeline 的方法，首先利用分类器筛选出和当前自然语言查询最相关的数据库模式，然后将其和自然语言查询组织成输入序列，使用 SQL 语句作为标签，微调大语言模型 T5 来生成 SQL 语句。

4.4.5 视图查询准确率

与 Li[65]等人在 spider 数据集上训练 RESDSQL 并在 spider 和 spider-syn 数据集上进行测试来验证模型性能的方法不同，本章在能更能模拟真实应用场景的 spider-syn 数据集上训练 ViSQL。为了公平比较，本章重新训练了 RESDSQL，然后在 spider-syn 和 spider 数据集上比较了两个模型的执行准确率 EA。

在下述实验结果中，本章模型 ViSQL 和基线模型 RESDSQL 均使用 base 级别的大语言模型 T5 来生成 SQL 语句，其中 single 和 nature 代表能够转化为视图查询的单表查询和涉及两张表的自然连接查询的执行准确率 EA。ViSQL_v 是在视图分类时，使用了投票策略后获得的执行准确率，表 4.4 和表 4.5 分别是模型在 spider-syn 和 spider 数据集

上的实验结果。

表 4. 4 spider-syn 数据集上视图查询的执行准确率

Table 4. 4 Execution accuracy of view queries on spider-syn dataset

Model	Single	Nature	All
RESDSQL	82.43%	68.56%	78.07%
ViSQL	83.30%	**71.21%**	79.50%
ViSQL$_v$	**85.04%**	70.08%	**80.33%**

从表 4.4 的实验结果可知，与 RESDSQL 相比，本章提出的 ViSQL 在 single 类型的数据上，准确率提高了 0.87%，在 nature 类型的数据上准确率提高了 2.56%，总体的准确率上提高了 1.43%。这一提升主要得益于 ViSQL 的设计思想。RESDSQL 将与当前自然语言查询相关的表名，列名和当前查询共同作为模型的输入，模型既需要学习表和列之间的对应关系，又需要判断表名会在 SQL 语句中的哪些位置出现。而 ViSQL 仅将当前视图的列名和自然语言查询作为模型的输入，输出不包含表名和 from 子句的 SQL 骨架，从而大大降低了待生成序列的复杂程度。另外，nature 类型数据的 EA 提升幅度比 single 更大，主要是视图分类器的贡献。但是需要注意的是，当视图分类器误将查询归结到错误的视图时，错误传播问题会导致 ViSQL 无法生成正确的 SQL 语句。

因此本章使用视图分类器和模式分类器，通过投票策略将置信度高的查询分类到视图上，其余查询使用 Text2SQL 模型来为其生成 SQL 语句。最终获得的视图查询准确率结果如表 4.4 中的 ViSQL$_v$ 所示，单表查询 single 类型的准确率提升了 2.61%，这是因为使用投票策略后，大部分 single 数据被正确分类到对应的视图上。即使对于分类错误的数据，也基本没有被分类到错误的视图上，仍然可以利用 Text2SQL 模型来为其生成 SQL 语句。然而，nature 类型的数据提升仅为 1.52%，这是因为在视图分类的过程中，对应到正确的视图上的测试数据较少。在总体的准确率上提高了 2.26%，这说明仅将确信度高的查询转化为视图查询，能够获得更高的 SQL 翻译准确率。本章将在下一节 4.4.6 对分类器的性能，以及其对 SQL 翻译准确率的影响进行详细分析。

表 4. 5 spider 数据集上视图查询的执行准确率

Table 4. 5 Execution accuracy of view queries on spider dataset

Model	Single	Nature	All
RESDSQL	85.74%	75.00%	82.36%
ViSQL	86.43%	**77.65%**	83.67%
ViSQL$_v$	**87.83%**	76.52%	**84.27%**

表 4.5 是模型在 spider 数据集上的视图查询准确率，与在 spider-syn 数据集上的表现相比，本章模型 ViSQL 和基线模型 RESDSQL 的准确率均有所提升。这是因为在 spider

数据集中，自然语言查询显式提及数据库模式中的表名和列名，模型能够更容易地判断哪些表名和列名会在 SQL 语句中出现。

本章提出的 ViSQL 与 RESDSQL 相比，在 single 类型的数据上 EA 提高了 0.69%，nature 类型的数据上的 EA 提高了 1.65%，总体准确率提高了 1.31%。在使用投票策略后，在 single 类型的数据 EA 提高了 2.09%， nature 类型的数据 EA 提高了 1.52%，在总体准确率上使用投票策略后提升了 1.91%。但这只是视图查询的准确率，在测试集中还包含着无法转化为视图查询的 other 类型数据，并且从表 4.4 和 4.5 的实验结果可知，分类器对模型的准确率影响较大，因此本章将在 4.4.6 节测试模型在所有测试数据上的表现，以及探究分类器对模型性能的影响。

4.4.5 分类器的影响

为了将能够转化为视图查询的数据更好地分类到正确的视图上，本文使用 single 和 nature 类型的数据来训练视图分类器，然后在测试的过程中，将每一个查询都对应到视图上，在 spider-syn 数据集上得到的实验结果如表 4.6 所示。

表 4. 6 spider-syn 数据集上视图查询的分类准确率

Table 4. 6 Classification accuracy of view queries on spider-syn dataset

Model	Single	Nature	All
View classifier	90.96%	91.29%	91.06%

表 4.6 中展示了本文视图分类器在 spider-syn 数据集上的分类准确率，single 类型获得了 90.96%的分类准确率，nature 类型获得了 91.29%的分类准确率，整体分类准确率为 91.06%。视图分类错误存在两种情况。第一种情况是没有将视图查询对应到任何一个视图上，本章会将此类查询输入 Text2SQL 模型来为其直接生成 SQL 语句。第二种情况是查询被分类到了错误的视图上，此时模型为 SQL 骨架添加的 from 子句一定是错误的，这会产生错误传播问题，导致最终获得的 SQL 语句一定是错误的。因此本章将视图分类器的阈值设置为 0.5，希望通过 Text2SQL 模型来为 other 类型的数据生成 SQL 语句，并减少错误传播问题。视图分类准确率如表 4.7 所示，下标 c 代表被分类到正确的视图，下标 z 代表模型预测的视图标签为 0，没有被分类到任何视图上。

表 4. 7 spider-syn 数据集的分类准确率

Table 4. 7 Classification accuracy on spider-syn dataset

Model	$Single_c$	$Single_z$	$Nature_c$	$Nature_z$	Other
View classifier	**90.61%**	4.5%	**76.52%**	18.18%	28.21%
$View\ classifier_v$	85.75%	**12.70%**	46.59%	**52.65%**	**63.59%**

从表 4.7 的实验结果可知，当把视图分类器的阈值设置为 0.5 后，single 类型的视

图分类准确率几乎没有下降，而 nature 类型的视图分类准确率下降了 14.77%，other 类型的查询大部分也都被错误地分类到视图上，仅有 28.21%的数据分类正确。这是因为本文在训练视图分类器的过程中，没有加入 other 类型的数据，导致其分辨此类数据的能力较差。并且许多 other 类型的数据对应的 SQL 语句，与创建视图的连接条件使用的是相同的表，导致模型分辨困难。

为了获得更好地整体 SQL 翻译准确率，本章采用了投票机制来提高视图分类的精确率。当视图分类器判断当前查询为视图上的查询，并且模式分类器认为该视图的连接条件中涉及的表和列都与当前查询相关时，才认为当前查询是该视图上的查询。通过投票机制获得的视图分类准确率如表中 View classifier$_v$ 行所示，几乎没有 single 和 nature 类型的查询被对应的错误的视图上，但 other 类型的视图分类准确率仍然只有 63.59%，说明如何识别非视图查询类型的数据是未来的研究重点，不同视图分类器对应 SQL 翻译准确率如表 4.7 所示，其中 ViSQL$_v$ 对应的是使用了投票策略的分类器。

表 4. 8 spider-syn 数据集上模型的执行准确率

Table 4. 8 Execution accuracy of the model on spider-syn dataset

Model	Single	Nature	Other	All
RESDSQL	82.43%	68.56%	**47.69%**	72.34%
ViSQL	83.30%	**71.21%**	21.03%	68.47%
ViSQL$_v$	**85.04%**	70.08%	38.46%	**72.44%**

从表 4.8 的实验结果可知，在为视图分类器设置 0.5 的阈值后，本文提出的 ViSQL 虽然在 single 和 nature 两种类型的查询上准确率超过了 RESDSQL，但其在 other 类型和整体准确率上的表现较差。使用了投票策略的 ViSQL$_v$ 在不仅视图查询的准确率超过了 RESDSQL，在测试集上的整体准确率也略有提高。而稍微有计算机常识的使用者，可以很轻易地判断出其提出的查询是否为结果集间的集合运算等 other 类型的查询，从而通过简单人工干预的方式，在实际场景中获得更好的 SQL 翻译准确率。

表 4.9 中展示了本章提出的视图分类器在 spider 数据集上视图查询的分类准确率，表 4.10 中展示了为视图分类器设置 0.5 的阈值和使用投票策略后的分类准确率，表 4.11 是模型在测试集上的执行准确率。

表 4. 9 spider 数据集上视图查询的分类准确率

Table 4. 9 Classification accuracy of view queries on spider dataset

Model	Single	Nature	All
View classifier	93.57%	92.37%	92.97%

从表 4.9 的实验结果可知，视图分类器的 single 类型和 nature 类型的分类准确率与模型在 spider-syn 数据集上的预测结果相比均有所提升。这说明当自然语言查询使用原

词提及表名和列名时，本章提出的视图分类器也可以更容易地判断当前查询对应的视图。

表 4. 10 spider 数据集的分类准确率

Table 4. 10 Classification accuracy on spider-syn dataset

Model	$Single_c$	$Single_z$	$Nature_c$	$Nature_z$	Other
View classifier	**93.22%**	3.83%	**81.81%**	12.88%	27.69%
View $classifier_v$	91.65%	**7.65%**	63.64%	**35.61%**	**59.49%**

从表 4.10 的实验结果可知，视图分类器在 spider 数据集上不仅将更多的视图查询对应到了正确的视图上，还将更少的视图查询对应到了错误的视图上，但在 other 类型的数据上分类准确率有所下降，这再次说明了如何帮助视图分类器识别 other 类型的数据是未来研究的重点。

表 4. 11 spider 数据集上模型的执行准确率

Table 4. 11 Execution accuracy of the model on spider dataset

Model	Single	Nature	Other	All
RESDSQL	85.74%	75.00%	**63.07%**	**78.72%**
ViSQL	87.48%	76.52%	36.41%	75.05%
$ViSQL_v$	**87.83%**	**76.52%**	51.28%	78.05%

由表 4.11 的实验结果可知，本文提出的 $ViSQL_v$ 在 spider 数据集上的整体准确率与 RESDSQL 相比略有下降，这是因为当测试集中的自然语言查询全部使用数据库模式中的原词来提及表名和列名时，模型能够更容易地判断哪些表名和列名与当前查询相关，使得本文提出的先生成视图上的 SQL 骨架，再为其补充表名和 from 子句来生成 SQL 语句的方法提升较小。

此外，本文分析了视图分类器在面对哪些数据类型时分类难度更高，并举出了 nature 类型和 other 类型的各一个例子，如图 4.5 所示。

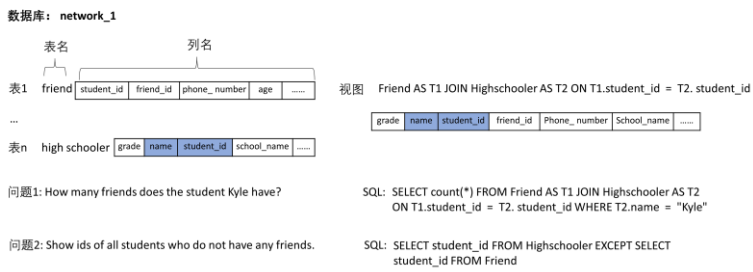

图 4.5 视图分类案例分析

Figure 4.5 View classification case analysis

图 4.5 中展示的问题 1 对应的 SQL 语句虽然是自然连接查询，但仅 high schooler 表

就包含了回答该问题所需要的全部列名，所以视图分类器可能将问题 1 错误地分类成该表上的单表查询。问题 2 对应的 SQL 语句虽然不是单表查询，但同样需要使用创建该视图的连接条件中对应的两张表，因此视图分类器可能将这类查询错误地判断为虚拟表上的视图查询。未来的研究应该将如何帮助视图分类器正确地分类这两种类型的数据作为重点。

4.4.5 消融实验

为了更直观地判断本章提出的从生成基本表上的 SQL 语句转化成生成视图上 SQL 骨架的方法能带来多大的性能提升，本章去掉了会导致错误传播问题的视图分类器，在 Spider-syn 和 Spider 两个数据集上进行了将视图分类结果置为完全正确的消融实验，从而更进一步地验证视图查询带来的性能提升。

表 4. 12 spider-syn 数据集消融实验

Table 4. 12 ablation experiment on spider dataset

Model	Single	Nature	Other	All
RESDSQL	82.43%	68.56%	47.69%	72.34%
ViSQL w/o VC	**87.65%**	**73.49%**	**47.69%**	**76.50%**

从表 4.12 的实验结果可知，将单表查询置为正确视图上的查询后，性能提高了 5.22%。将自然连接查询置为正确视图上的查询后，性能提高了 4.93%，在整体的准确率上提高了 4.16%。该实验结果充分证明了将查询转化为确定视图上的查询能有效地提高模型的 Text2SQL 准确率。而将 nature 类型的查询转化为类似单表查询的视图查询后，其准确率仍然低于 single 类型的查询，这是因为自然连接查询对应的视图均为虚拟表，与单表查询对应的基本表相比，通常会包含更多的列名，这会提高模型选择数据库模式的难度，并且 SQL 语句中包含的子句的数量也会影响模型的生成难度。

表 4. 13 spider 数据集消融实验

Table 4. 13 ablation experiment on spider dataset

Model	Single	Nature	Other	All
RESDSQL	85.74%	75.00%	63.07%	78.72%
ViSQL w/o VC	**89.74%**	**79.55%**	**63.07%**	**82.11%**

从表 4.13 中模型在 spider 数据集上的实验结果可知，ViSQL 对于 single 和 nature 类型数据的翻译准确率分别提升了 4%和 4.55%，在整体的准确率上提高了 3.39%，提升幅度小于其在 spider-syn 数据集上的表现，这和本章前一节的实验中得到的结论是一致的，当自然语言查询使用原词来提及表名和列名时，模型能够更容易地识别哪些数据库模式与当前查询相关，所以本章提出的 ViSQL 带来的提升较小。

4.4 总结

本章方法为常见的查询场景事先创建视图，将单表查询和多表查询转化为特定视图上的查询，从生成基本表上的 SQL 语句转化为生成视图上的 SQL 骨架，为能够归结到视图上的查询带来了较大的性能提升。但本章提出的分类器，识别非视图查询的能力较差，这会导致错误传播问题，影响模型在测试集上的整体准确率。因此如何识别非视图查询类型的数据是未来的研究重点。并且本章事先创建的视图无法涵盖所有查询。因此，在未来的研究中，可以考虑让数据库的设计者来创建常用视图，从而将更多的查询转化为视图上的查询。

第5章 结论与展望

5.1 论文工作总结

现有 Text2SQL 研究已经在数据集上取得了较好的效果，因此本文将研究的焦点从如何提高模型的 SQL 翻译准确率转移到了探索现有数据集与实际应用场景存在着哪些差异，以及在现有模型卓越的性能是否是数据集上的假象。在此过程中，本文发现现有方法严重依赖自然语言查询对数据库模式的显式提及，和多表查询的准确率远低于单表查询的问题。针对以上现象，本文进行了以下两个工作：

（1）针对现有研究仅利用自然语言查询和数据库模式来生成 SQL 语句，缺乏数据库领域建模知识导致模型严重依赖表名和列名在自然语言中显式提及的问题，提出了利用词典扩展数据库模式信息的 Text2SQL 方法，在使用同义词提及数据库模式的 spider-syn 数据集上取得了 2.13%的性能提升，并且当被查询数据库的表名和列名在自然语言查询中明确出现时，本文扩展的解释信息也不会对模型造成干扰，在有偏的 spider 数据集上也取得了 0.39%的性能提升。

（2）现有研究将目光聚焦在复杂 SQL 语句的生成，并引入了序列化、图神经网络等技术，以帮助模型更好地捕捉数据库模式的语义，从而提高复杂 SQL 语句的翻译准确率。但现有方法的多表查询准确率仍然明显低于单表查询，当数据库内仅包含一张表时，模型的单表查询准确率可以达到最高水平。因此，本文提出了基于视图的 Text2SQL 方法，为查询场景中常见的单表查询和自然连接查询事先创建视图，在特定的视图上先为其生成 SQL 骨架，再添加表名和 from 子句来获得 SQL 语句。在 spider-syn 数据集上单表查询和自然连接查询分别取得了 0.87%和 2.56%的性能提升，在 spider 数据集上单表查询和自然连接查询分别取得了 0.69%和 1.65%的性能提升。

5.2 研究展望

本文将目光聚焦在现有 Text2SQL 数据集和实际应用场景之间的差异，对现有研究进行了一些针对性的改进并取得了一定的进展。但本文研究仍然存在一些缺点和不足，未来工作可以从几个方向进行进一步研究。

（1）本文在领域通用词典中为表名和列名扩展的解释信息，会不可避免地引入噪声。因此，下一步研究可以考虑首先判断当前数据库中存储的信息所对应的领域，然后在领域词典中为表名和列名扩展解释信息，或者利用当前数据库建模的领域知识来对获取的解释信息进行过滤，减少噪声对模型性能的影响。

（2）本文提出的基于视图的 Text2SQL 方法，目前仅考虑了如何将实际的应用场景

中最常见的单表查询和自然连接查询转化为视图上的查询，无法处理集合运算和嵌套查询等情况。在接下来的研究中可以考虑将数据库中的所有表建立成笛卡尔积，从而将所有查询均转化成在视图上的查询。

参考文献

[1] Androutsopoulos I, Ritchie G D, Thanisch P. Natural Language Interfaces to Databases–An Introduction[J]. Natural language engineering, 1995, 1(1): 29-81.

[2] Li F, Jagadish H V. Understanding Natural Language Queries over Relational Databases[J]. Acm Sigmod Record, 2016, 45(1): 6-13.

[3] Affolter K, Stockinger K, Bernstein A. A Comparative Survey of Recent Natural Language Interfaces for Databases[J]. The VLDB Journal, 2019, 28(5): 793-819.

[4] Warren D H D, Pereira F C N. An Efficient Easily Adaptable System for Interpreting Natural Language Queries[J]. American Journal of Computational Linguistics, 1982, 8(3-4): 110-122.

[5] Popescu A M, Armanasu A, Etzioni O, et al. Modern Natural Language Interfaces to Databases: Composing Statistical Parsing with Semantic Tractability[C]//COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics. Geneva, Switzerland: Association for Computational Linguistics (ACL), 2004: 141.

[6] Hallett C. Generic querying of relational databases using natural language generation techniques[C]//Proceedings of the Fourth International Natural Language Generation Conference. Sydney, NSW, Australia: Association for Computational Linguistics (ACL). 2006: 95-102.

[7] Giordani A, Moschitti A. Generating SQL Queries Using Natural Language Syntactic Dependencies and Metadata[C]// Natural Language Processing and Information Systems. 17th International Conference on Applications of Natural Language to Information Systems, NLDB 2012. Groningen, Netherlands: Springer Verlag. 2012: 164-170.

[8] Price P J. Evaluation of Spoken Language Systems: the ATIS Domain[C]// Proceedings of the third DARPA Speech and Natural Language Workshop. 1990: 91-95.

[9] Zelle J M, Mooney R J. Learning to Parse Database Queries Using Inductive Logic Programming[C]//Proceedings of the national conference on artificial intelligence. PORTLAND, OR: AAAI, 1996: 1050-1055.

[10] Zhong V, Xiong C, Socher R. Seq2sql: Generating Structured Queries from Natural Language Using Reinforcement Learning[J]. arXiv preprint arXiv:1709.00103, 2017.

[11] Yu T, Zhang R, Yang K, et al. Spider: A Large-Scale Hu-man-Labeled Dataset For Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task[C]// Conference on Empirical Methods in Natural Language Processing (EMNLP). Brussels, Belgium: Association for Computational Linguistics (ACL), 2020: 3911-3921.

[12] Finegan-Dollak C, Kummerfeld J K, Zhang L, et al. Improving Text-to-SQL Evaluation Methodology[C]// 56th Annual Meeting of the Association-for-Computational-Linguistics (ACL). Melbourne, Australia: Association for Computational Linguistics (ACL), 2018: 351-360.

[13] Dong L, Lapata M. Coarse-to-Fine Decoding for Neural Semantic Parsing[C]. 56th Annual Meeting of the Association-for-Computational-Linguistics (ACL). Melbourne, Australia: Association for Computational Linguistics (ACL), 2018: 731-742.

[14] Krishnamurthy J, Dasigi P, Gardner M. Neural Semantic Parsing with Type Constraints for Semi-structured Tables[C]//Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. Copenhagen, Denmark: Association for Computational Linguistics (ACL), 2017: 1516-1526.

[15] Wang C, Tatwawadi K, Brockschmidt M, et al. Robust Text-to-SQL Generation with Execution-Guided Decoding[J]. arXiv preprint arXiv:1807.03100, 2018.

[16] Yaghmazadeh N, Wang Y, Dillig I, et al. SQLizer: Query Synthesis from Natural Language[J]. Proceedings of the ACM on Programming Languages, 2017, 1(OOPSLA): 1-26.

[17] Xu X, Liu C, Song D. Sqlnet: Generating Structured Queries from Natural Language without Reinforcement Learning[J]. arXiv preprint arXiv:1711.04436, 2017

[18] Yu T, Li Z, Zhang Z, et al. TypeSQL: Knowledge-Based Type-Aware Neural Text-to-SQL Generation[C]// 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL HLT 2018. New Orleans, LA, United states: Association for Computational Linguistics (ACL), 2018: 588-594.

[19] Hwang W, Yim J, Park S, et al. A Comprehensive Exploration on WikiSQL with Table-Aware Word Contextualization[J]. arXiv preprint arXiv:1902.01069, 2019.

[20] He P, Mao Y, Chakrabarti K, et al. X-SQL: reinforce schema representation with context[J]. arXiv preprint arXiv:1908.08113, 2019.

[21] Zhang X, Yin F, Ma G, et al. M-SQL: Multi-Task Representation Learning for Single-Table Text2SQL Generation[J]. IEEE Access, 2020, 8: 43156-43167.

[22] Sutskever I, Vinyals O, Le Q V. Sequence to Sequence Learning with Neural Networks[[J]. Advances in Neural Information Processing Systems, 2014, 31: 3104-3112..

[23] Bogin B, Berant J, Gardner M. Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing[C]// 57th Annual Meeting of the Association-for-Computational-Linguistics (ACL). Florence, Italy: Association for Computational Linguistics (ACL), 2019: 4560-4565.

[24] Bogin B, Gardner M, Berant J. Global Reasoning over Database Structures for Text-to-SQL Parsing[C]// Conference on Empirical Methods in Natural Language Processing / 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). Hong Kong, HONG KONG: Association for Computational Linguistics (ACL),2019: 3659-3664.

[25] Hui B, Geng R, Wang L, et al. S2SQL: Injecting Syntax to Question-Schema Interaction Graph Encoder for Text-to-SQL Parsers[C]// 60th Annual Meeting of the Association-for-Computational-Linguistics (ACL). Dublin, Ireland: Association for Computational Linguistics (ACL), 2022: 1254-1262.

[26] Wang B, Shin R, Liu X, et al. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers[C]// 58th Annual Meeting of the Association for Computational Linguistics (ACL). Virtual, Online, United states: Association for Computational Linguistics (ACL), 2020: 7567-7578.

[27] Cao R, Chen L, Chen Z, et al. LGESQL: Line Graph Enhanced Text-to-SQL Model with Mixed Local and Non-Local Relations[C]// Joint Conference of 59th Annual Meeting of the Association-for-Computational-Linguistics (ACL) / 11th International Joint Conference on Natural Language Processing (IJCNLP). ELECTR NETWORK: Association for Computational Linguistics (ACL), 2021: 2541-2555.

[28] Lei W, Wang W, Ma Z, et al. Re-examining the Role of Schema Linking in Text-to-SQL[C]// Conference on Empirical Methods in Natural Language Processing (EMNLP). ELECTR NETWORK: Association for Computational Linguistics (ACL), 2020: 6943-6954.

[29] Guo J, Zhan Z, Gao Y, et al. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation[C]// 57th Annual Meeting of the Association-for-Computational-Linguistics (ACL). Florence, ITALY: Association for Computational Linguistics (ACL) ,2019: 4524-4535.

[30] Lin X V, Socher R, Xiong C. Bridging Textual and Tabular Data for Cross-Domain Text-to-SQL Semantic Parsing[C]//Findings of the Association for Computational Linguistics: EMNLP 2020. Brussels,

BELGIUM: Association for Computational Linguistics (ACL) 2020: 4870-4888.

[31] Kenton J D M W C, Toutanova L K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[C]//Proceedings of NAACL-HLT. Minneapolis, MN, United states: Association for Computational Linguistics (ACL), 2019: 4171-4186.

[32] Yin P, Neubig G. A Syntactic Neural Model for General-Purpose Code Generation[C]// 55th Annual Meeting of the Association-for-Computational-Linguistics (ACL). Vancouver, CANADA: Association for Computational Linguistics (ACL), 2017: 440-450.

[33] Gan Y, Chen X, Huang Q, et al. Towards Robustness of Text-to-SQL Models against Synonym Substitution[C]// Joint Conference of 59th Annual Meeting of the Association-for-Computational-Linguistics (ACL). ELECTR NETWORK: Association for Computational Linguistics (ACL), 2021: 2505-2515.

[34] Bahdanau D, Cho K H, Bengio Y, et al. Neural Machine Translation By Jointly Learning To Align And Translate[C]. 3rd International Conference on Learning Representations, ICLR 2015. San Diego, CA, United states: International Conference on Learning Representations, ICLR, 2015.

[35] Gu J, Lu Z, Li H, et al. Incorporating Copying Mechanism in Sequence-to-Sequence Learning[C]//54th Annual Meeting of the Association-for-Computational-Linguistics (ACL). Berlin, GERMANY: Association for Computational Linguistics (ACL), 2016: 1631-1640.

[36] Zettlemoyer L S, Collins M. Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars[C]// 21st Conference on Uncertainty in Artificial Intelligence, UAI 2005. PORTLAND, OR: AUAI Press. 2005: 658-666.

[37] Yih W, Richardson M, Meek C, et al. The Value of Semantic Parse Labeling for Knowledge Base Question Answering[C]// 54th Annual Meeting of the Association-for-Computational-Linguistics (ACL). Berlin, GERMANY: Association for Computational Linguistics (ACL). 2016: 201-206.

[38] Herzig J, Berant J. Decoupling Structure and Lexicon for Zero-Shot Semantic Parsing[C]. 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018. Brussels, Belgium: Association for Computational Linguistics, 2018: 1619-1629.

[39] Kolitsas N, Ganea O E, Hofmann T. End-to-End Neural Entity Linking[C]//Proceedings of the 22nd Conference on Computational Natural Language Learning. Brussels, Belgium: Association for Computational Linguistics (ACL), 2018: 519-529.

[40] See A, Liu P J, Manning C D. Get To The Point: Summarization with Pointer-Generator Networks[C]//55th Annual Meeting of the Association-for-Computational-Linguistics (ACL). Vancouver, CANADA: Association for Computational Linguistics (ACL), 2017: 1073-1083.

[41] Li Y, Tarlow D, Brockschmidt M, et al. Gated Graph Sequence Neural Networks[C]// 4th International Conference on Learning Representations, ICLR 2016. San Juan, Puerto rico: International Conference on Learning Representations, ICLR, 2015.

[42] Cho K, Van Merriënboer B, Gulcehre C, et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation[C]// 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014. Doha, Qatar: Association for Computational Linguistics (ACL), 2014: 1724-1734.

[43] De Cao N, Aziz W, Titov I. Question Answering by Reasoning Across Documents with Graph Convolutional Networks[C]//2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics. Santa Fe, NM, United states: Association for Computational Linguistics, 2019: 2306-2317.

[44] Sorokin D, Gurevych I. Modeling Semantics with Gated Graph Neural Networks for Knowledge Base

Question Answering[C]//Proceedings of the 27th International Conference on Computational Linguistics. Santa Fe, NM, United states: Association for Computational Linguistics (ACL). 2018: 3306-3317.

[45] Xiao C, Dymetman M, Gardent C. Sequence-based Structured Prediction for Semantic Parsing[C]//Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. Berlin, GERMANY**:** Association for Computational Linguistics (ACL)**.** 2016: 1341-1350.

[46] Rabinovich M, Stern M, Klein D. Abstract Syntax Networks for Code Generation and Semantic Parsing[C]//Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics. Vancouver, CANADA: Association for Computational Linguistics (ACL). 2017: 1139-1149.

[47] Lin K, Bogin B, Neumann M, et al. Grammar-based Neural Text-to-SQL Generation[J]. arXiv preprint arXiv:1905.13326, 2019.

[48] Shaw P, Uszkoreit J, Vaswani A. Self-Attention with Relative Position Representations[C]// 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL HLT 2018, New Orleans, LA, United states: Association for Computational Linguistics (ACL). 2018: 464-468.

[49] Chen D, Lin Y, Li W, et al. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view[C]// 34th AAAI Conference on Artificial Intelligence / 32nd Innovative Applications of Artificial Intelligence Conference / 10th AAAI Symposium on Educational Advances in Artificial Intelligence. New York, NY: AAAI press, 2020: 3438-3445.

[50] Vaswani A, Shazeer N, Parmar N, et al. Attention is All You Need[C]//Proceedings of the 31st International Conference on Neural Information Processing Systems. Long Beach, California: MIT Press, 2017: 6000-600.

[51] Shen Y, Tan S, Sordoni A, et al. Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks[C]// International Conference on Learning Representations (ICLR). New Orleans, LA, United states: ICLR Press, 2019.

[52] Ba J L, Kiros J R, Hinton G E. Layer Normalization[J]. arXiv preprint arXiv:1607.06450, 2016.

[53] Dozat T, Manning C D. Deep Biaffine Attention for Neural Dependency Parsing [C]// 5th International Conference on Learning Representations, ICLR 2017. Toulon, France: ICLR Press, 2016.

[54] Loshchilov I, Hutter F. Decoupled Weight Decay Regularization[C]//International Conference on Learning Representations. New Orleans, LA, United states: ICLR Press, 2018.

[55] Gal Y, Ghahramani Z. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks[C]//Proceedings of the 30th International Conference on Neural Information Processing Systems. Barcelona, SPAIN : MIT Press, 2016: 1027-1035.

[56] Tang L R, Mooney R J. Automated Construction of Database Interfaces: Integrating Statistical and Relational Learning for Semantic Parsing[C]// Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora held in Conjunction with the 38th Annual Meeting of the Association-for-Computational-Linguistics. Hong Kong Univ Sci & Technol, Hong Kong, China: Association for Computational Linguistics (ACL), 2000: 133-141.

[57] Popescu A M, Etzioni O, Kautz H. Towards A Theory of Natural Language Interfaces to Databases[C]//Proceedings of the 8th international conference on Intelligent user interfaces. Miami Beach, FL, USA : Association for Computing Machinery (ACM), 2003: 327-157.

[58] Iyer S, Konstas I, Cheung A, et al. Learning a Neural Semantic Parser from User Feedback[C]// 55th Annual Meeting of the Association-for-Computational-Linguistics (ACL). Vancouver, BC, Canada: Association for Computational Linguistics (ACL), 2017: 963-973.

[59] Li F, Jagadish H V. Constructing an Interactive Natural Language Interface for Relational Databases[J]. Proceedings of the VLDB Endowment, 2014, 8(1): 73.

[60] Liu Q, Yang D, Zhang J, et al. Awakening Latent Grounding from Pretrained Language Models for Semantic Parsing[C]//Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021. Virtual, Online: Association for Computational Linguistics (ACL), 2021: 1174-1189.

[61] Liu Y, Ott M, Goyal N, et al. Roberta: A robustly optimized bert pretraining approach[J]. arXiv preprint arXiv:1907.11692, 2019.

[62] Xu K, Wang Y, Wang Y, et al. SeaD: End-to-end Text-to-SQL Generation with Schema-aware Denoising[C]//Findings of the Association for Computational Linguistics: NAACL 2022, 2022: 1845-1853.

[63] Raffel C, Shazeer N, RoBERTs A, et al. Exploring the limits of transfer learning with a unified text-to-text transformer[J]. The Journal of Machine Learning Research, 2020, 21(1): 5485-5551.

[64] Shaw P, Chang M W, Pasupat P, et al. Compositional Generalization and Natural Language Variation: Can a Semantic Parsing Approach Handle Both?[C]//Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). Virtual, Online, 2021: 922-938.

[65] Li H, Zhang J, Li C, et al. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql[C]//Proceedings of the AAAI Conference on Artificial Intelligence. Washington, DC, United states : AAAI Press, 2023, 37(11): 13067-13075.

[66] Lin T Y, Goyal P, Girshick R, et al. Focal loss for dense object detection[C]//Proceedings of the IEEE international conference on computer vision. Los Alamitos, CA: IEEE Computer Society, 2017: 2980-2988.

附录 A

本文使用的图神经网络中结点间的关系是不对称的，在附表 1 和附表 2 中只写出了单向关系，从结点 y 到结点 x 的关系可以很容易地推测出.表中的 Q、T、C 分别代表自然语言结点，表名结点和列名结点。

附录 A- 1 结点间一跳关系

结点 x	结点 y	关系类型	关系描述
Q	Q	DISTANCE+n	y 是 x 的下 n 词(n>2 时视为 2 处理)
C	C	FOREIGN-KEY	y 是当前表的主键
T	C	HAS	y 是表 x 的列
T	C	PRIMARY-KEY	y 是表 x 的主键
Q	T	T-NO-MATCH	x 和 y 之间字面匹配未成功
Q	T	T-PARTIAL-MATCH	x 是 y 的一部分，但问题中不包含 y
Q	T	T-EXACT-MATCH	x 是 y 的一部分，并且问题中包含 y
Q	C	C-NO-MATCH	x 和 y 之间字面匹配未成功
Q	C	C-PARTIAL-MATCH	x 是 y 的一部分，但问题中不包含 y
Q	C	C-EXACT-MATCH	x 是 y 的一部分，并且问题中包含 y
Q	C	VALUE-MATCH	x 是 y 在数据集中的值的一部分

附录 A- 2 结点间其他关系

结点 x	结点 y	关系类型	关系描述
C	C	SAME-TABLE	x 和 y 属于同一张表且均非主键
C	C	FOREIGN-KEY-COL-F	x 是 y 的外键
T	T	FOREIGN-KEY-TAB-F	表 y 中包含表 x 的外键
T	T	FOREIGN-KEY-TAB-B	表 x 和表 y 中互相包含对方的外键

攻读学位期间取得的研究成果

[1]. 以第一作者身份，已被《******》正式录用待发表
[2]. 以第四著作权人，发表软著***
[3]. 以第七著作权人，发表软著***
[4]. 参与四川省重点研发项目***