分类号 TP391.1　　密 级
U D C 004.9　　编 号

中南财经政法大学

硕士专业学位论文

基于BERT模型的中文Text-to-SQL系统

研究生姓名：杨　　燊
校内导师姓名、职称：马　霄　讲　师
申请者类别：非专项计划
校外导师姓名：　　无
专业学位类别：电　子　信　息
专业名称：电　子　信　息
研究方向：不区分研究方向
入学时间：二〇二〇年九月

二〇二二年六月二日

Chinese Text-to-SQL system based on BERT model

Yang Shen

2022.06.02

摘要

在当前信息技术高速发展的情形下，各行业领域都产生了海量的数据，而这些数据往往都是存储在结构化或半结构化的数据库中。对于关系型数据库中数据的获取需要使用 SQL 语句来执行查询操作，但是这些编程语句的使用增加了非技术人员分析和使用数据的难度，开发技术人员也无法将所有可能用到的查询语句封装成接口供非技术人员使用。因此为了快速便捷地从数据库中准确获取数据信息，自然语言处理领域的研究人员尝试使用深度学习模型来实现自然语言文本到 SQL 语句的转化工作，将其作为自然语言处理领域的一个分支任务来研究。

当前 Text-to-SQL 方面的研究主要都面向英文文本，在英文转化领域已经取得了较为成熟的研究成果。然而，对于中文自然语言转化 SQL 的研究较少。主要是由于中文语言本身的多义性，导致查询语句中自然语言描述与数据库中存储的数据属性字段不一致，加大了中文转化任务的难度。因此，本文将中文的 Text-to-SQL 任务作为研究对象，采用中文 NL2SQL 数据集，针对不同的查询场景，以提升 SQL 语句转化的正确率作为评估模型的标准，引入 BERT 模型来构建深度学习模型实现中文自然语言文本到 SQL 语句的转化。然后将所构建的算法模型应用到查询系统中，实现从中文文本输入到查询数据项结果输出的功能。本文的主要工作和贡献如下所示：

（1）本文将 Text-to-SQL 任务转化成预测 SELECT 子句和 WHERE 子句两个子任务来进行研究并结合 BERT 预训练模型构建神经网络模型，将两个子任务经过模型处理后的结果合并得到目标 SQL 特征表达式。具体为首先将原始自然语言查询文本经过 BERT 编码层后得到词向量序列，作为初始化输入；然后构建四个深度神经网络模型将各部分的转化任务变换为分类任务来进行，其次将分类的结果统一融合起来得到模型输出，解决了中文自然语言文本到 SQL 语言的转换问题。

（2）将构建的算法模型应用到实际系统中，将原始数据库文件进行解析得到模型输入所需的内容格式，同时对输入文本进行特征化处理，基于本文所构建的 Text-to-SQL 模型，将处理后的输入转化得到对应的 SQL 语句，最后交由系统执行并显示结果集，实现从中文自然语言查询文本到结果输出的功能。系统实现之后，在实际查询环境下对系统进行测试，测试结果表明在准确率和时效性上该系统都有着良好的效果。

关键词：中文 Text-to-SQL 任务；BERT；自然语言处理；预训练模型；语言模型

Abstract

With the rapid development of information technology, a large amount of data has been produced in various industries, and these data are often stored in structured or semi-structured databases. For the acquisition of data in relational database, SQL statements need to be used to perform query operations, but the use of these programming statements increases the difficulty for non-technical personnel to analyze and use data, and developers and technicians cannot encapsulate all possible query statements into interfaces for non-technical personnel to use. Therefore, in order to quickly and conveniently obtain accurate data information from the database, researchers in the field of natural language processing try to use the deep learning model to realize the transformation from natural language text to SQL statement, and study it as a branch task in the field of natural language processing.

At present, the research on text to SQL is mainly oriented to English text, and more mature research results have been achieved in the field of English conversion. However, there are few studies on the transformation of Chinese natural language into SQL. Mainly due to the polysemy of the Chinese language itself, the natural language description in the query statement is inconsistent with the data attribute fields stored in the database, which increases the difficulty of the Chinese conversion task. Therefore, this thesis takes the Chinese text to SQL task as the research object, adopts the NL2SQL dataset, takes improving the accuracy of SQL sentence transformation as the standard of the evaluation model for different query scenarios, and introduces the Bert model to build a deep learning model to realize the transformation from Chinese natural language text to SQL sentence. Then the algorithm model is applied to the query system to realize the function from Chinese text input to query data item result output. The main work and contributions of this thesis are as follows:

(1) In this thesis, the text to SQL task is transformed into two subtasks to predict the select clause and where clause for research. Combined with the Bert pre training model, the neural network model is constructed, and the results of the two subtasks processed by the model are combined to obtain the target SQL feature expression. Firstly, the original natural language query text is passed through the Bert coding layer to obtain the word vector sequence as the initialization input; Then four deep neural network models are constructed to transform the transformation tasks of each part into classification tasks. Secondly, the classification results are unified and integrated to obtain the model output, which solves the problem of transformation from Chinese natural language text to SQL language.

(2) After parsing the input text of the SQL model into the system, the input text of the SQL model should be processed based on the text-to-text file, and the actual input results are obtained, Realize the function from Chinese natural language query text to result output. After the implementation of the system, the system is tested in the actual query environment. The test results show that the system has good results in accuracy and timeliness.

Key Words: Chinese text to SQL task; BERT; Natural language processing; Pre training model; Language model

目　录

第一章　绪论

第一节　研究背景与应用价值

Text-to-SQL 是一种将自然语言文本转换为 SQL 语句的技术，它使得数据管理人员能够方便快捷地从海量数据库信息中获取需要检索的数据。此技术将用户的自然语言文本转化成了计算机可识别、可执行的语义表示，对于实际生产环境使用的结构化或者半结构化数据库而言，这种特定规范的语义表示通常就是 SQL 语句。Text-to-SQL 任务实际上就是介于数据库与用户之间的接口，将自然语言和计算机结构化查询语言联系起来，用户通过此接口调用即可实现从语言文本到 SQL 的转化功能。

相比于英文 Text-to-SQL 任务，面向中文的 Text-to-SQL 任务更为复杂。目前阶段大部分与 Text-to-SQL 任务相关的研究工作都是建立在英文数据集基础上的，同时把英文数据集上的模型直接应用到中文数据集也是不可行的。因为相对于英文文本而言，中文文本的词汇之间没有分隔而是连续的，并且在语义特征上也更为复杂，计算机直接理解起来存在着很大障碍，对于中文文本的预处理过程也是一项困难的工作，因此面向中文的 Text-to-SQL 任务还有着很大的研究空间。

同时在日常生活中，此项技术应用领域也十分广泛，例如以下所示:

(1) Text-to-SQL 可以应用在结构化数据库基础上的智能问答系统，既可方便数据库管理人员对海量数据的管理，大幅降低人力物力的成本；也可以提供在应用软件的客户端上，从而为用户提供更为方便快捷的查询检索服务，消除数据获取中间流程的壁垒。比如针对存储电影信息的数据库而言，如果用户需要查询豆瓣上评分超过 9 分的中文电影有哪些，在此查询文本中存在着 9 分(评分)，中文(语言类型)这两个维度信息的检索需求，经过 Text-to-SQL 系统后都能够返回准确的查询结果给用户。

(2) Text-to-SQL 技术同样能够应用于搜索引擎的优化上，使搜索得到的结果更符合用户预期的效果。它能够使得对于待查询信息的关键字提取更加精准，同时在传统的检索方式中检索结果只是简单地将查询文本进行匹配，导致检索结果仍然是包含有查询文本的信息或链接。而此项技术的引入则可以将查询文本所得到的结果直接检索返回给用户，如此可提高用户检索的直观效率。

总之 Text-to-SQL 作为自然语言处理领域新兴的一个分支任务，具有着丰富的研究内容，同时将此项技术应用到工业实际生产生活环境中来完成自然语言和数据库的交互，降低数据获取的门槛，实现基于数据库的自动问答系统，能够极大地提高生产的质量和效率。

第二节 国内外研究现状

一、数据集介绍

目前，Text-to-SQL 领域已有 WikiSQL[1]、Spider[2]等诸多公开数据集，以上数据集都是针对英文转化任务而产生的。与此同时也诞生了对于中文任务的中文数据集，例如 CSpider[3]，NL2SQL 等等。不同数据集都有各自的特点，以下为这些数据集的简单介绍。

WikiSQL 是 Salesforce 公布的一个大型数据集，其包含 24241 张表格，80645 条自然语言问句及对应的 SQL 语句。由于其自然语言问题大部分为 what 类型的简单语句，因此对于 WikiSQL 数据集的转化任务相对比较容易处理。

Spider 数据集是耶鲁大学提出的数据集。该数据集包含了 10181 条自然语言问句，总共包含 200 个独立的数据库以及 5693 条 SQL。在其中包含了复杂的 SQL 结构，例如 Group By、Order By 等操作，一些情况还涉及到多表连接，与真实的 SQL 查询环境类似，因此转换的难度也更大，目前针对其构建的模型准确率最高只有 54.7%。CSpider 则是从 Spider 翻译而来的面向中文的数据集，其数据库表结构和内容以及自然语言问句都和 Spider 是一致的。

NL2SQL 数据集是在 2019 年举办的天池 NL2SQL 中文挑战赛中提出的，也是 2020 年之前公开的 Text-to-SQL 数据集中唯一一份高质量的中文数据集。一共包含 41522 条有标注的训练集数据，4086 条无标注数据作为测试集。

在以上的数据集中，单表英文数据集情况下的 Text-to-SQL 任务的查询已经趋于成熟，最新的 SeaD[4]模型准确率已经能够达到 93%以上，而对于复杂数据集查询准确率较低。相对而言，在中文 Text-to-SQL 任务上开展的研究不多，原因主要包含以下两个因素：首先是现阶段中文数据集还尚未完善，在之前很长一段时间之内没有公开的数据集；其次是由于中文文本本身就有着复杂的结构以及多变的语义，导致对中文文本的转化任务比英文更为复杂，还需要考虑多方面的因素来构建模型解决中文 Text-to-SQL 问题。

二、研究实践现状

在 Text-to-SQL 任务被提出的初期，研究人员主要将其处理为机器翻译任务进行工作，即实现从序列到序列的生成功能，由此提出了 Seq2Seq[5]模型框架来完成这一任务。从目前国内外实践现状来看，在 Seq2Seq 框架的基础上，主要有以下改进的模型：

(1) Seq2SQL 模型

在 Seq2SQL 模型中借鉴了 Seq2Seq 的思想，Seq2SQL 将输入的语句编码后再解码成结构化的 SQL 语言输出，同时在模型中加入了强化学习，它将自然语言文本问题、SQL 关键词、对应数据库的所有元素融合起来作为输入序列，利用 Pointer Network[6]从输入序列中直接提取出关联词汇，并将其作为目标 SQL 结构的一部分。其模型结构由三部分组成，第一部分是一个聚合函数分类器，作用是将输入语句按照查询数据类型分类成为 COUNT、MAX、MIN 等统计相关的约束条件，首先通过编码器将输入文本生成特征向量，然后使用 softmax 函数处理特征向量，最后通过多层的神经网络完成分类任务；第二部分是预测查询文本选中了数据库表的哪一列，即查询字段在数据库表中的位置，同样使用多层的神经网络和 softmax 来确定选中的是哪一列；第三部分则是确定 WHERE 子句中约束条件的内容。综合以上三个部分，即为 Seq2SQL 模型的结构，可见其只是简单地将 Text-to-SQL 任务中的 SQL 语句拆分成了三部分来分步进行转换，但是其在 WikiSQL 数据集上的转化准确率从 Seq2Seq 的 35.9%提升到了 59.4%，有了一定的提升。与 Seq2SQL 模型原理类似的还有 STAMP[7]、IRNet[8]等模型。

(2) TypeSQL[9] 模型

在问题较为简单的数据集上，其 SQL 结构只包含 SELECT 和 WHERE 关键字，相对来说查询语句结构较为简单。为了解决 Seq2Seq 模型中顺序错误带来的影响，即预测的 SQL 语句中部分关键字的顺序可能与训练集中关键字顺序不一致，但其实际查询效果是一样的，TypeSQL 则很好地解决了这一问题。它是在 SQLNet[10]基础之上改进得到的，其原理是采用模板填充的方法来生成 SQL 语句，为了更好地对查询文本中出现的实体和数字进行建模，TypeSQL 模型为每个单词都赋予了具体的类型。在类型赋予过程中，首先将文本分割成多段单词序列，并在数据库表中检索可能匹配的字段，匹配成功的部分如果是数字，则根据其类型对应赋值 INTEGER、FLOAT、DATE、YEAR 这四个值，如果是实体类型，则对应赋值 PERSON、PLACE、COUNTRY 等类型值。对查询文本进行类型分类后，即可在数据库表中检索出对应的数据作为模板的填充值。该模型在自然语言查询问句及其对应的 SQL 结构较为简单的数据集上转化效果较好，并且在此模型基础之上引申出了其他相关模型，例如 SQLova[11]、X-SQL[12]等。

(3)RAT-SQL[13]模型

对于复杂的自然语言查询文本，其对应的 SQL 形式也复杂，在其中涉及到多个 SQL 关键词的组合以及嵌套的多个子句等。并且在测试集中，一些 SQL 形式在训练集中没有出现过，这就使得在此种情形下的模型除了对新的数据库要有良好的泛化能力之外，还要对新的其他复杂 SQL 形式也要有泛化能力。此前的 Text-to-SQL 模型在编码阶段只考虑到了数据库表的结构，没有将文本中所包含的上下文信息结合起来，

使得模型的表示能力局限于已有的数据集中，限制了模型的泛化表示能力。而 RAT-SQL 模型使用基于关系感知的自注意力机制很好地解决了这一问题，并极大地提升了 SQL 转化的准确率。它将数据库表结构的显式关系与查询文本和表结构之间的隐式关系作为编码的参考因素进行模型编码，具体而言是利用数据库表中具有的属性名信息，将数据库列名与 SQL 语句的字段名相关联，使其成为一个统一的输入序列作为模型的输入，然后经过神经网络处理后完成 Text-to-SQL 任务。

第三节　本文主要内容与贡献

本文主要内容为将中文的 Text-to-SQL 任务作为研究对象，采用 NL2SQL 数据集，针对不同的查询场景，以提升 SQL 语句转化的正确率作为评估模型的标准，引入 BERT 模型来构建深度神经网络模型实现中文自然语言文本到 SQL 语句的转化，同时使用 PyQt 图形化软件开发技术完成最终的查询系统，实现从中文文本输入到所需数据项结果输出的功能。本文的工作和贡献如下所示：

（1）本文将 Text-to-SQL 任务转化成预测 SELECT 子句和 WHERE 子句两个子任务来进行研究，并结合 BERT 预训练模型构建神经网络模型，通过预训练模型出众的语义表示能力来提高模型转化的准确率，同时使用两个不同的深度神经网络模型来进行转化预测，再将其融合起来得到目标 SQL 语句。

（2）在实际应用系统中引入上述构建的中文 Text-to-SQL 算法模型，将用户的输入进行预处理操作后作为模型的输入，经过模型转换后并在系统中执行得到的 SQL 语句，最终输出得到查询结果并显示，实现从中文自然语言查询文本到结果输出显示的功能。

第四节　本文结构

本文分为五个章节，每一章节的主要内容如下所示：

第一章为绪论，该部分首先是对本文的实际背景以及应用价值的阐述，然后介绍了当前阶段国内外对于自然语言文本到 SQL 语句转化的研究现状，着重介绍了目前存在的主流数据集的内容。接下来则是论文的主要研究对象和本文的内容结构。

第二章是对于本文的 Text-to-SQL 任务相关技术和概念的介绍，主要包括目前对于中文自然语言处理问题所使用的技术以及预训练模型，从原理开始展开来概述将中文自然语言查询文本转换成 SQL 语句所用到的理论和技术。

第三章则是基于 BERT 的中文 Text-to-SQL 算法模型详细设计部分。搭建模型之前先对模型进行结构规划设计，然后根据中文自然语言文本的特点制定合适的编码方案，再分析处理 NL2SQL 中文数据集。数据集处理完毕之后根据输入以及输出要求

来搭建深度学习网络模型，模型搭建好之后开始数据训练以及验证模型有效性。同时包含实验过程以及结果产出部分，介绍了实验所需硬件以及软件环境。首先使用训练数据集对模型进行训练，然后使用测试数据集对所搭建的模型与算法进行有效性的验证，并与已有的模型进行转化结果对比，同时观察分析执行结果并进行参数调整和优化网络结构来提高模型转化的效率。

第四章为中文 Text-to-SQL 查询系统的实现，将原始数据库文件进行解析得到模型输入所需的内容格式，同时对输入文本进行特征化处理，然后在算法模型实现中文文本到 SQL 语句的转化功能，最后交由系统执行并显示结果集。

第五章是总结和展望，总结了本文的工作内容以及所做出的贡献，并对系统可改进之处和未来的研究发展方向进行了展望。

第二章 相关理论基础

近些年自然语言处理技术的蓬勃发展和趋于成熟的研究成果，开创了一些创新的研究领域以及实际生产应用场景。本文所研究 Text-to-SQL 技术也是其中之一，它为面向结构化数据库的查询检索提供了一个接口，简化了数据获取的操作。Text-to-SQL 任务其主要涉及了以下两个主要的子任务，一是对输入接口的自然语言文本和已有数据库表中属性字段文本的编码过程，只有经过合适的编码之后才能作为深度学习网络模型的输入。二是实现从编码序列到目标 SQL 的过程，此过程主要使用深度神经网络来实现。经过以上两个步骤之后即可完成自然语言文本到 SQL 语句的转化任务。

第一节 中文自然语言编码

在自然语言能被机器识别理解之前，需要将自然语言进行处理并提取出相关特征，从而转化成计算机能够理解的数据序列，以上对于原始自然语言的处理过程就是自然语言编码的过程。自然语言在大多数情况下是文本的形式，在英文的自然语言处理问题中，英文文本的相邻词汇之间有固定的间隔，因此在处理时无需再进行分词处理。但是在中文文本中，相邻词汇之间未有间隔符，相比于英文文本需要增加一道分词的操作。当前阶段使用较为普遍的中文分词工具有 Jieba 分词库，哈工大 LTP2 和 Stanford CoreNLP3 等。对中文文本进行分词处理之后，还要将分词的序列映射为向量才能作为神经网络模型的输入。得到此映射向量可以采用随机初始化的方式，也可以通过 Word2Vec[14]等方式得到。

对于自然语言文本，有着以下常用的表示方法将其转换为计算机能够理解的数学语言表示：首先是基于 one-hot、tf-idf、textrank[15] 等的 Bag-of-words 模型；其次为主题模型例如 LSA[16]、pLSA[17]、LDA[18]等；再者是基于词向量的固定表征如 Word2Vec、GloVe[19]；最后是基于词向量的动态表征如 ELMo[20]、BERT 等。

one-hot 是最为简单的语义编码方式，但存在维度过大和语义界限等问题。比如在互联网广告系统里，如果用户输入的查询信息是“手机”，而有一个广告的关键词是“华为 Mate 40”。按照一般的逻辑思维，很容易看出来这两个词汇之间是有着联系的，后者在前者的范围之内，因此用户在购买手机时，广告系统可以将此款手机推送给该用户。但是从这两个词对应的 one-hot 向量来看，由于其向量的每个维度上的编码只有 0 和 1，所以会导致这两个词汇之间毫无相关性。而早期获取词向量的方式通常是从语言模型中得到的，比如 NNLM[21] 和 RNNLM[22]，但是在此类模型中词向量只是

一个副产物。之后提出了具有相同上下文语境的词可能有着相似含义的假设，由此便引申出了 Word2Vec。

Word2vec 的核心思想是通过词的上下文得到词的向量化表示，其有两种表示方法：CBOW（通过附近词预测中心词）、Skip-gram（通过中心词预测附近的词），如图 2-1 所示：

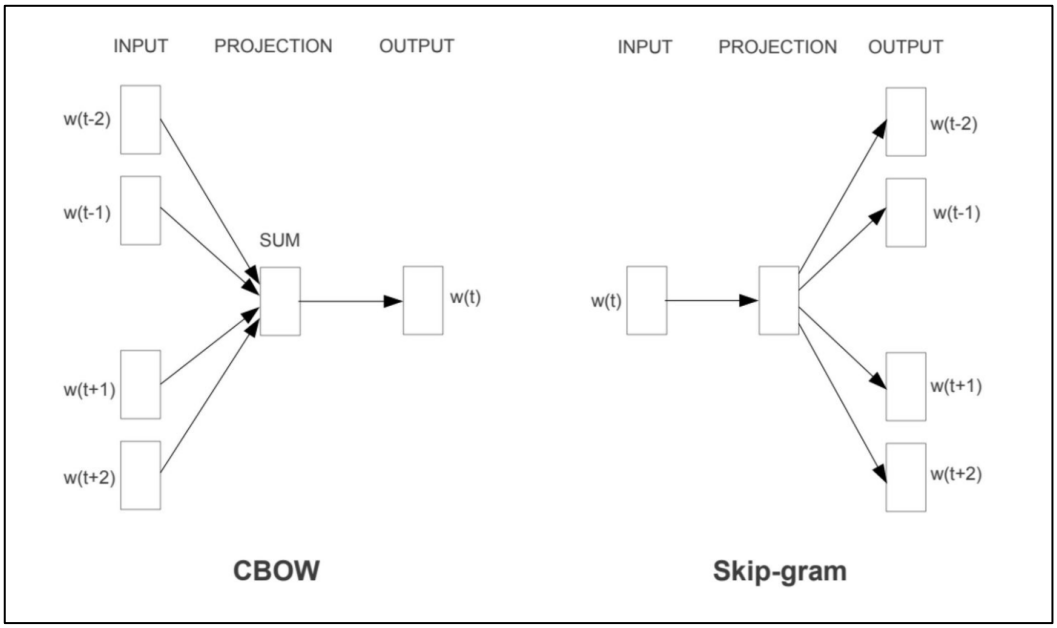

图 2-1 Word2vec 的两种方式

在上图中，CBOW 从目标词的上下文出发来进行目标词的预测，图中利用了大小为 2 的窗口，选取预测词的前后两个词作为参照来进行目标词的预测。具体而言是首先设定词向量的维度 d，并将所有的词汇转换为统一的 d 维向量，然后对上下文所有的词向量编码得到隐藏层的向量，随即根据这个隐藏层来进行词汇的预测。在 CBOW 中的做法是将隐藏层的向量进行简单的相加计算，然后使用 softmax 函数进行分类。在原理上 Skip-gram 和 CBOW 是相似的，但在其具体输入细节上有所不同，Skip-gram 的输入是目标词，它是首先将目标词转化为隐藏层向量，然后以这个向量为依据，来进行目标词上下文两个词的预测。

虽然就本质而言此类词向量是语言模型的产物，但是在此基础之上所进行的一系列优化，目的都是为了得到更优的词向量。GloVe 则结合了 LSA 和 Word2vec 的优点，通过检索全局语料库和上下文语境来构建词向量。使用 Word2vec 和 GloVe 的问题是，在不同的语言环境下，同一个词可能有着不同的含义，而在这两个模型中词的向量表示在不同语境中却是相同的，为解决这一问题，ELMo 模型被提出。它使用了多层 LSTM 去学习词汇的复杂用法，并能够学习到相同词汇在不同语义环境中的变化。

BERT[23]则是基于 Transformer[24]的双向编码表示，它是一个预训练模型，模型训练时的两个任务是预测句子中被掩盖的词以及判断输入的两个句子是不是上下句。在预训练好的 BERT 模型后面根据特定任务加上相应的神经网络结构，可以完成 NLP 的下游任务，比如文本分类、机器翻译等，在本文中也将使用 BERT 模型来对中文自然语言文本进行编码处理。

第二节 Attention 机制

Attention[24]机制是模仿人类注意力而提出的一种解决问题的办法，即从诸多信息中能够快速筛选出高价值信息的方法。在输入序列较长时，通过 LSTM 很难得到合理的向量表示，Attention 机制的出现则很好地解决了这一问题。主要思想是保留 LSTM 的中间结果，并选取新的模型再对其进行学习，并与输出进行关联，从而达到信息筛选更为精准的目的。

在对自然语言文本进行编码的时候，需要使用到 Encoder 和 Decoder，即编码器和解码器，其作用是将固定长度的输入转化为固定长度的输出。它们可以采用的模型也多种多样，例如 CNN、RNN、GRU、LSTM 等，如下图 2-2 所示为其工作原理图：

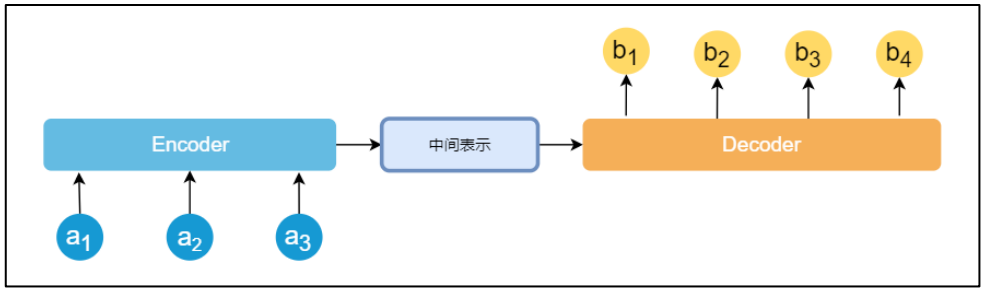

图 2-2 编码器示意图

Encoder 的作用是将输入的原始自然语言文本转化成一个中间表示，Decoder 过程则是根据语义中间表示将之前的编码输出。此种机制存在着一个问题，当输入序列长度非常长时，由此产生出来的语义中间效果较差，需要对此编码结构进行调整才能够适应此类输入。Attention 机制即是在这种情形下诞生的，在编码过程中输出不再是一个固定长度的中间语义，而是一个由不同长度向量构成的序列，解码过程根据这个序列进行进一步处理。Attention 机制运行的示意图如图 2-3 所示：

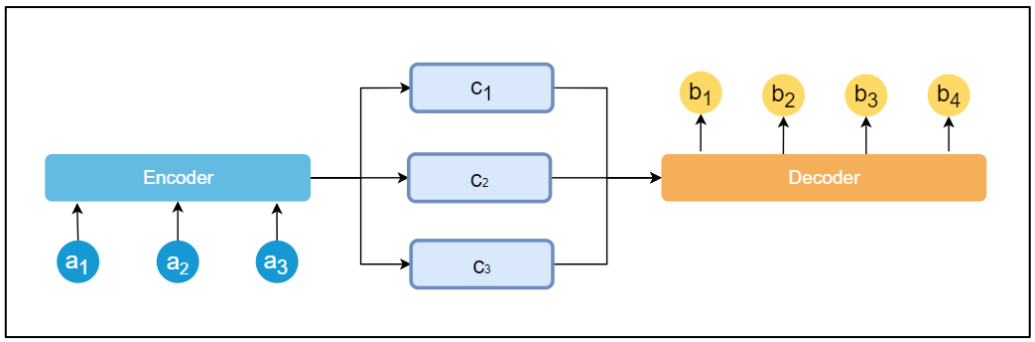

图 2-3 引入 Attention 机制的编码器

相比于传统的直接对输入序列 A 进行编码，在 Attention 机制中是根据各个元素的重要程度加权求和得到的，即公式 2-1 所示

$$C_i=\sum_{j=0}^{Tx} p_{ij}\, f(a_{ij}) \tag{2-1}$$

参数 i 表示时刻，j 表示序列中的第 j 个元素，T_x 表示序列的长度，函数 f 表示对元素 x_j 的编码，p_{ij} 为一个概率值,反映了元素 h_j 对 c_i 的重要性,可以使用公式 2-2 来表示：

$$p_{ij}=\frac{exp(e_{ij})}{\sum_{k=1}^{Tx} exp(e_{ik})} \tag{2-2}$$

其中 e_{ij} 反映了不同编码元素之间匹配度，当匹配度越高时，说明其中一个元素对另外一个元素的影响越大，从而使得最终的 p_{ij} 值也越大。

第三节 BERT 模型

在自然语言处理领域中，大多数传统算法模型极其依赖于人工特征工程。例如 log-linear、条件随机场（CRF）模型等。自然语言处理领域目前的有监督数据集非常小，以至于在其基础上建立的深度学习模型容易过拟合，从而对其他数据不能很好地进行泛化。预训练模型就是首先在大规模语料上进行无监督训练，然后学习得到通用语言表征的自然语言处理模型。由此一来后继解决类似问题或者下游任务时，就无需从头开始训练新的模型，可直接使用预训练模型来提高效率。

对于自然语言处理问题而言，目前诸多模型已经印证了预训练模型在自然语言处理任务中的效果是非常显著的。现阶段通过预训练模型来完成下游自然语言处理任务主要有两种方式：首先是基于特征的方法，例如 ELMo 模型，它主要从多个维度出发去改善传统编码方案的不足之处，该模型结合了上下文的语义环境，能够提取出词汇关于上下文敏感的特征；其次是基于微调的方法，例如 OpenAI GPT[25]模型，它主要在大规模预训练模型的基础之上，通过不断调整模型中的可训练参数来提升模型

的性能。基于微调的方法是当前使用较为普遍的方法，其优点是大部分模型参数都已经设定完备，只有小部分的可变参数需要重新再次学习，学习成本较低。

BERT（Bidirectional Encoder Representations from Transformers）是 Google 在 2018 年提出的预训练模型，该模型首次实现自然语言双向的预训练，在各个下游任务上的效果都有所提升。它使用了 Transformer 作为算法的主要框架，使其能更彻底的捕捉语句中的双向关系。同时它也是一个多任务模型，使用了 Mask Language Model(MLM) 和 Next Sentence Prediction(NSP) 两个自监督任务来实现多任务训练目标。

BERT 的输入编码向量是 3 个特征向量的单位和，如图 2-4 所示，这三个特征为符号嵌入、位置嵌入以及分割嵌入。符号嵌入将单词划分成了一组子词单元，能够保证单词的有效性和字符的灵活性之间的平衡。例如在图 2-5 的示例中，“playing”分成了“play”和“ing”；位置嵌入将单词在句子中的位置信息作为向量进行了编码；分割嵌入用于分割区别两个句子，例如判断 B 是否是 A 的下文。特征表示中有两个特殊符号[CLS]和[SEP]，其中[CLS]表示该特征用于分类模型。[SEP]表示分句符号，用于断开输入语料中的两个句子。

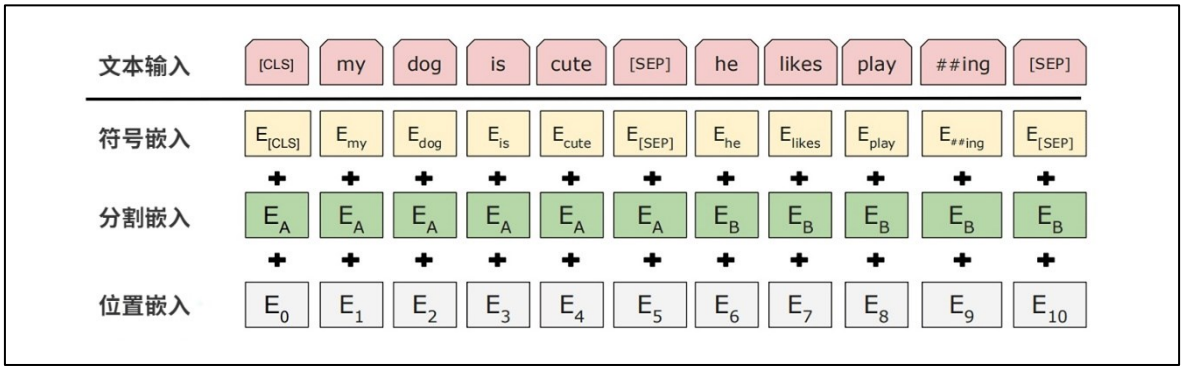

图 2-4 BERT 模型输入特征表示

BERT 多任务模型中的 NSP 主要内容为判断句子 B 是否是句子 A 的下文。如果是的话输出“IsNext”，否则输出“NotNext”，这个关系保存在图 2-5 中的[CLS]符号中。MLM 为训练时就从输入语料上 Mask 掉一些单词，然后通过上下文预测该单词。在训练模型时确定要被 Mask 掉的单词之后，80%的单词会直接替换为[Mask]，10%会替换为其它任意单词，10%会保留原始 Token。如下图 2-5 所示：

80%：my dog is hairy -> my dog is [mask]

10%：my dog is hairy -> my dog is apple

10%：my dog is hairy -> my dog is hairy

图 2-5 BERT 的 Mask 机制

第四节　本章小结

本章介绍了与 Text-to-SQL 任务的相关理论与技术。首先阐述的是针对自然语言文本的编码方案，只有将自然语言转化成计算机能够识别的数字编码之后才能让其处理结构复杂且数量庞大的自然语言。然后介绍了在编码过程中引入的 Attention 机制，可以有效地减少当输入文本长度非常长时产生的语义中间件效率低下的问题。接下来是分别介绍了基于特征的方法和基于微调的方法的预训练模型，可以有效地提高自然语言处理下游任务的效率，同时详细描述了本文所使用的 BERT 预训练模型的原理以及工作机制。

第三章 中文 Text-to-SQL 模型

本章的主要内容为针对 NL2SQL 数据集来完成中文 Text-to-SQL 任务的方法和模型。本章主要使用 BERT 预训练模型进行编码，并在预训练模型基础之上新增分类模型。与此同时，将任务分解为两个不同预测结果的任务，首先是针对 SQL 语句中 SELECT 子句的预测，即对所选取的列以及其对应的聚合函数的预测；其次是对 WHERE 子句选取的条件列以及条件关系和条件值的预测。在第一个子任务中，本文充分利用了 SQL 语句中各子句之间的相互依赖关系，对原始数据进行了特征化处理；在第二个子任务中，本文结合了自然语言查询文本和数据表中的内容，将两者联合起来进行条件值的获取，解决了当自然语言查询文本中的关键字和数据库表中存储的字段不一致时的预测问题，使得最终的 SQL 语句中条件值关键字与数据库存储值相统一，提升了转化的准确率。

第一节 问题定义

在描述问题和任务之前，首先对相关的概念进行形式化的概括。中文文本查询语句为在已有数据库表中进行查询的中文语句表述，可以是疑问句或者祈使句。在本文中对此类查询语句定义如公式 3-1 所示：

$$Q=\{s1,s2,s3,\ldots,si,\ldots,sn\} \quad (3\text{-}1)$$

其中 Q 表示该查询文本，si 表示该文本的第 i 个字符，此字符可以是汉字、数字以及标点符号等。n 表示该自然语言文本中总共包含 n 个字符，关于中文文本查询实例如下图 3-1 所示：

Q1：我不是药神的豆瓣评分是多少？

Q2：2008 年北京奥运会中国获得了多少枚金牌？

Q3：给我推荐一下景点等级为 5A 的景点

图 3-1 中文自然语言查询文本示例

以上三条文本样例即为对于特定数据库表的中文文本查询语句，本文所做工作也是围绕此类语句作为系统输入展开。

数据库表一般是具有相同属性结构的多条数据的集和，每一条数据项在数据库表中都是独一无二的。而这些数据所具有的相同的属性结构即为该表的列名称。在本文中对此类数据库表的结构化定义如公式 3-2 所示：

$$T=\{t，\{col_1,col_2,\ldots,col_i,\ldots,col_n\}\{type_1,type_2,\ldots,type_i,\ldots,type_n\}\} \quad (3\text{-}2)$$

其中 T 表示该数据库表的整体结构，也就是表本身，t 为表的名称。col_i 为此表中第 i 列的名称，n 表示表 T 一共有 n 列属性。最后的 $type_i$ 则为第 i 列数据内容的属性，具体可以为字符文本、数字、时间戳等数据类型。如图 3-2 所示为数据库表名为 student 的一个简单具体实例：

id	name	age	gender
1	July	22	1
2	Bob	23	1
3	Jane	21	0
4	Maria	24	0
5	Gan	21	0
6	John	22	1
7	Lex	22	1
8	Puff	21	0

图 3-2 数据库表实例

在该表中 id，name，age，gender 为此表的列名，列名下面的每一行则为表中的具体数据项。

在以上定义的前提下，本文的 SQL 语句生成任务可以归纳为给定已知的数据库表 T 和中文自然语言查询文本 Q，从 T 和 Q 的映射关系中构建出一种可靠有效的算法模型，从而得到理想的 SQL 语句并交由系统执行，即可实现待查询数据的获取功能。其结构化定义如公式 3-3 所示:

$$(T，Q)\rightarrow S\rightarrow A \quad (3\text{-}3)$$

此定义表示系统的输入为数据库表 T 和针对于该数据库的自然语言查询语句 Q，输出则为此查询语句的 SQL 语句表述，然后执行此 SQL 语句即可得到需要查询的数据项结果 A。简言之此系统直接实现了从查询文本到查询数据结果的功能。

第二节 NL2SQL 数据集

该数据集由追一科技在 2019 年举办的首届中文 NL2SQL 挑战赛中提出，在此数据集中包含了多个领域的数据表格，并在这些表格基础上人工标注了自然语言查询文本与 SQL 语句的匹配对。训练集中含有 41522 条有标签数据，测试集中包含 4086 条有标签数据。数据集信息如下表 3-1 所示：

表 3-1 NL2SQL 数据集信息

	训练集	验证集	测试集
数据库表数量	5013	1197	1102
样本数量	41522	4396	4086

在训练集和验证集中都包含真实 SQL 结构的特征表达式，测试集中则不包含。以训练集为例，数据集由 train.db 、train.json 和 train.tables.json 这三个文件组成。在 train.json 中存储着训练样本的信息；train.tables.json 包含了训练样本中所对应的数据库表的详细信息，包括数据库表 id，每一列的列名及其数据类型等；tables.db 则是数据表以 sqlite 格式存储的数据库文件，其中包含着数据库表的具体存储内容。在数据集 train.json 中训练样本如下图所示，图 3-3 为具体的一条训练样例，图 3-4 为此样例对应的数据库表具体内容信息，表 3-2 为此样例对应的数据库表表头信息。


```
"table_id": "a7adcaa33b0611e9b090f40f24344a08",
"question": "以城市交通，古城遗址为主题的湘西凤凰古城在什么时间段开放",
"sql": {
        "agg": [0],
        "cond_conn_op": 1,
        "sel": [8],
        "conds": [
            [6,2,"城市交通，古城遗址"],
            [1,2,"湘西凤凰古城"]
        ]
    }
```


图 3-3 训练样例

col_1	col_2	col_3	col_4	col_5	col_6	col_7	col_8	col_9
3	三亚蜈支洲岛	1281	4A	海南省	海南省三亚市林旺镇海棠湾蜈支洲岛度假中心	海滨海岛，名山名水	0898-88751258	08:00-16:00
4	三亚亚龙湾天堂森林公园	139	4A	海南省	海南省三亚市吉阳镇亚龙湾国际旅游度假区	海滨海岛，公园乐园	0898-38219999	07:30-17:30
8	嘉兴西塘古镇	1918	4A	浙江省	浙江省嘉兴市嘉善县西塘镇南苑路258号	世界遗产，古城遗址	0573-84567890	08:00-17:00
11	三亚西岛	83	4A	海南省	海南省三亚市三亚湾	海滨海岛，游船邮轮	0898-88262007	08:00-18:00
12	湘西凤凰古城	2724	4A	湖南省	湖南省湘西州凤凰县	城市交通，古城遗址	0743-3227121	旺季06:30-18:00；淡季07:30-17:30
13	成都熊猫基地	227	4A	四川省	四川省成都市成华区外北三环熊猫大道1375号	亲子合家欢，公园乐	028-83510033	08:00-18:00
15	北京欢乐谷	8390	4A	北京市	北京市朝阳区东四环小武基北路	公园乐园，主题体验	010-67201818	平日：9:00-22:00；周末：8:30-22:00
16	三亚亚龙湾	7005	4A	海南省	海南省三亚市亚龙湾国家旅游度假区	海滨海岛，演出表演	0898-88568899	07:30-18:00

图 3-4 训练样例对应数据库表

表 3-2 训练样例对应表头名称

序号	景点名称	关注人数	景区等级	省市	地址	景点主题	景区电话	开放时间
3	三亚蜈支洲岛	1281	4A	海南省	海南省三亚市林旺镇海棠湾蜈支洲岛度假中心	海滨海岛，名山名水，赏花赏叶，游船邮轮，潜水	0898-88751258	08:00-16:00
4	三亚亚龙湾热带天堂森林公园	139	4A	海南省	海南省三亚市吉阳镇亚龙湾国际旅游度假区	海滨海岛，公园乐园，游船邮轮，潜水	0898-38219999	07:30-17:30
...	...	...	...	...	...	...	...	...

在该数据集中，模型需要解决的问题是根据输入的表格名称以及用户中文自然语言查询语句来自动生成可执行的SQL表达式。在一条真实SQL语句中存在着SELECT、WHERE 以及聚合函数等相互独立的结构，对应着训练数据集中不同的模块结构。如上图所示 SQL 结构中包含着所选择的列(sel)、对于此列的聚合函数操作(agg)、选择的条件约束(conds)以及不同条件约束之间的关系(cond_conn_op）。其中数据集中 SQL 结构转化映射表如下图 3-5 所示：

```
op_sql_dict = {0:">", 1:"<", 2:"==", 3:"!="}
agg_sql_dict = {0:"", 1:"AVG", 2:"MAX", 3:"MIN", 4:"COUNT", 5:"SUM"}
conn_sql_dict = {0:"", 1:"and", 2:"or"}
```

图 3-5 SQL 表达式字典

由此可知 agg 字段对应的为转换图中 agg_sql_dict 字典，映射关系为{0:"", 1:"AVG", 2:"MAX", 3:"MIN", 4:"COUNT", 5:"SUM"}。而 conds 结构中对应的为 SQL 映射表中的 op_sql_dict 字典，其映射关系为{0:">", 1:"<", 2:"==", 3:"!="}，其中为一个三元组形式的列表，分别表示所选取的条件列、条件关系以及满足条件的值。例如[2,2,10]表示第三列的值等于 10，即条件关系为 sel_3==10；cond_conn_op 对应着 conn_sql_dict 字典，映射关系为{0:"", 1:"and", 2:"or"}，其表示的内容为不同查询条件之间的关系，例如当 SQL 表达式中 WHERE 子句部分存在多个条件约束时，条件之间的关系是且(and)、或(or)还是不存在(none)。其整体结构图如下图 3-6 所示。

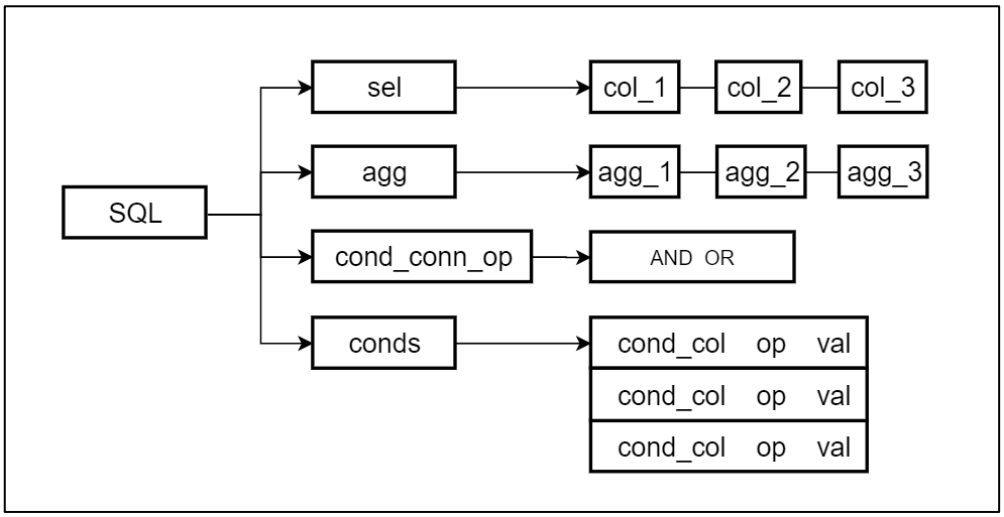

图 3-6 SQL 表达式结构图

接下来对本文数据集的相关字段进行解释，以图 3-3 所示的训练样本为例，结合 SQL 表达式字典来说明逐步转化的过程。首先所选取的列 sel 值为“8”，以及样例中的 agg 值为“0”，所以组合在一起为 sel_9（在此数据集设定中，所选取列下标从 0 开始），即查询的是数据库表第九列的值。其次 conds 值为[6,2,"城市交通，古城遗址"], [1,2,"湘西凤凰古城"]，根据字典映射值转换结果为 sel_7==“城市交通，古城遗址”和 sel_1==“湘西凤凰古城”，即对应原表中第七列值为“城市交通，古城遗址”和第一列为“湘西凤凰古城”的数据行。同时 cond_conn_op 的值为 1，对应 SQL 语句中的 AND 连接符。最后加上输入的 table_id 将上述训练样例通过表达式字典翻译过来并连接就得到 SQL 语句：SELECT sel_9 FROM Table_ a7adcaa33b0611e9b090f40f24344a08 WHERE sel_7==‘城市交通，古城遗址’AND sel_1==‘湘西凤凰古城’，通过此 SQL 语句表达的查询信息为查询主题为“城市交通，古城遗址”以及名称为“湘西凤凰古城”的景点的营业时间段。这与训练样本中给定的自然语言查询文本“以城市交通，古城遗址为主题的湘西凤凰古城在什么时间段开放”是一致的。本文所研究的任务就是如同上述过程，通过设计模型算法实现自然语言文本到可执行 SQL 语句的转换。综上所述，根据 SQL 结构映射字典可以概括出所有的 SQL 表达式格式如图 3-7 所示：

```
SELECT [$agg  $sel]*
FROM $table_name
WHERE $conds.sel $conds.op $conds.val
($conn_op $conds.sel $conds.op $conds.val)*
```

图 3-7 SQL 形式化表达式

其中[]*表示至少出现一次，()*表示出现 0 次或多次；参数 agg 为数据集中选择的列对应的聚合函数，取值为数字 0 至 5，分别对应{“NONE”,“AVG”,“MAX”,“MIN”,“COUNT”,“SUM”}；sel 为聚合函数中的内容列，即需要查询的列；table_name 为将查询的表名；conds.sel 为 WHERE 关键字后面查询条件中的所选列名，conds.op 为查询条件的关系，其取值范围为{“>”,“<”,“==”,“!=”}，conds.val 则为该查询条件列所要对应的值，也就是需要比较的值，取值范围广泛；最后的 conn_op 为条件连接符，取值为{“AND”,“OR”}，表示前后两个查询条件之间的连接关系。

原始输入数据为一条包含查询信息的中文自然语言文本和对应的表格 id，因此首先需要将自然语言文本进行特征化处理，采用同义词统一替代以及消除无关符号等方式把原始文本进行变换。同时由于转化 SQL 语句任务主要就是预测出数据库表中的哪些列被查询到，而列所对应的数据库表属性就是其表头名称，在模型中采用列名来表示这一属性，因此根据数据库表 id 将表头的名称添加到查询语句末尾，以整体形成模型的输入。拼接后的序列结构如下图 3-8 所示：

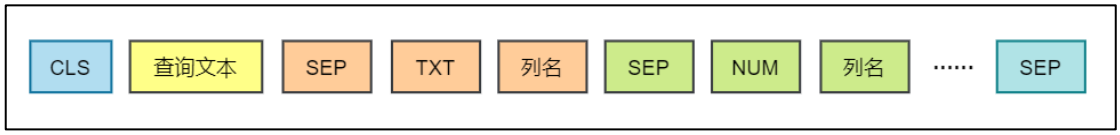

图 3-8 输入文本和表头的拼接序列

在以上结构图中，“CLS”为 BERT 模型在序列前添加的标识符，用于标识该符号之后的语句序列为一个待编码的句子，此语句经过 BERT 编码后得到一个表征向量用于后续的分类工作。“SEP”为分开两个句子的标识符，避免多个语句之间融合而产生消极影响。“TXT”标签表名了在该标签之后的字段类型为一个文本的序列，“NUM”标签则表明了其后的字段属性为一个数值类型，这两者共同概括了所有数据集中表格的数据项的内容。查询文本序列为经过特征处理之后的自然语言输入，其具体处理方式为首先去除文本中的标点符号（如，。；（）[]《》、“”？等），其次进行同义词替换，例如高于、多于等词汇统一替换为大于，低于、少于等词汇替换为小于。

第三节 Text-to-SQL 模型构建

在本文所研究的 Text-to-SQL 任务中，所选取数据集的中文自然语言查询文本和其对应的 SQL 语句都是序列的存在方式，其中 SQL 语句的序列形式如上图 3-7 所示。因此中文自然语言文本转化成 SQL 语句的任务，可以近似地看作是一个序列转化为序列（Seq2Seq）的任务。

图 3-9 描述的为本文 Text-to-SQL 模型的架构图。首先针对数据集的特点为给定一个自然语言文本和对应的表格名称，为了更方便地作为模型的输入，本文设计的模

型第一层为数据转换层，在本层中首先将文本和表格内容整合在一起，通过融合编码的方式将两个输入序列合并成为一个输入序列，有利于后续任务的处理。其次经过 BERT 编码层对上述得到的输入序列进行编码，在此过程主要使用 BERT-wwm-Chinese 预训练模型。主要操作为先把 conds 中除了条件值 val 以外的属性值加入模型中。进行完编码之后，在标签映射层中将 SQL 语句拆分成 SELECT 子句和 WHERE 子句两个部分，SELECT 部分对应的需要预测部分为选择的列 sel 和聚合函数 agg 的值，WHERE 部分为条件连接符 cond_op 和条件类型 cond 的值。在此之后即可基于训练集中的 SQL 的形式化表示序列和上述处理后的输入序列构建深度学习网络模型，神经网络由不同大小的全连接层构成，其大小与构成 SQL 表达式字典的映射值数量相对应。然后是条件填充层，将候选的条件值 val 进行预测填充，即将原始中文文本中对应的词语编码与 val 值搭建一层神经网络来进行预测。如此一来模型输出就具有了上述字典映射表格式的 SQL 特征表达式，再将此特征表达式转换为可执行的 SQL 语句。

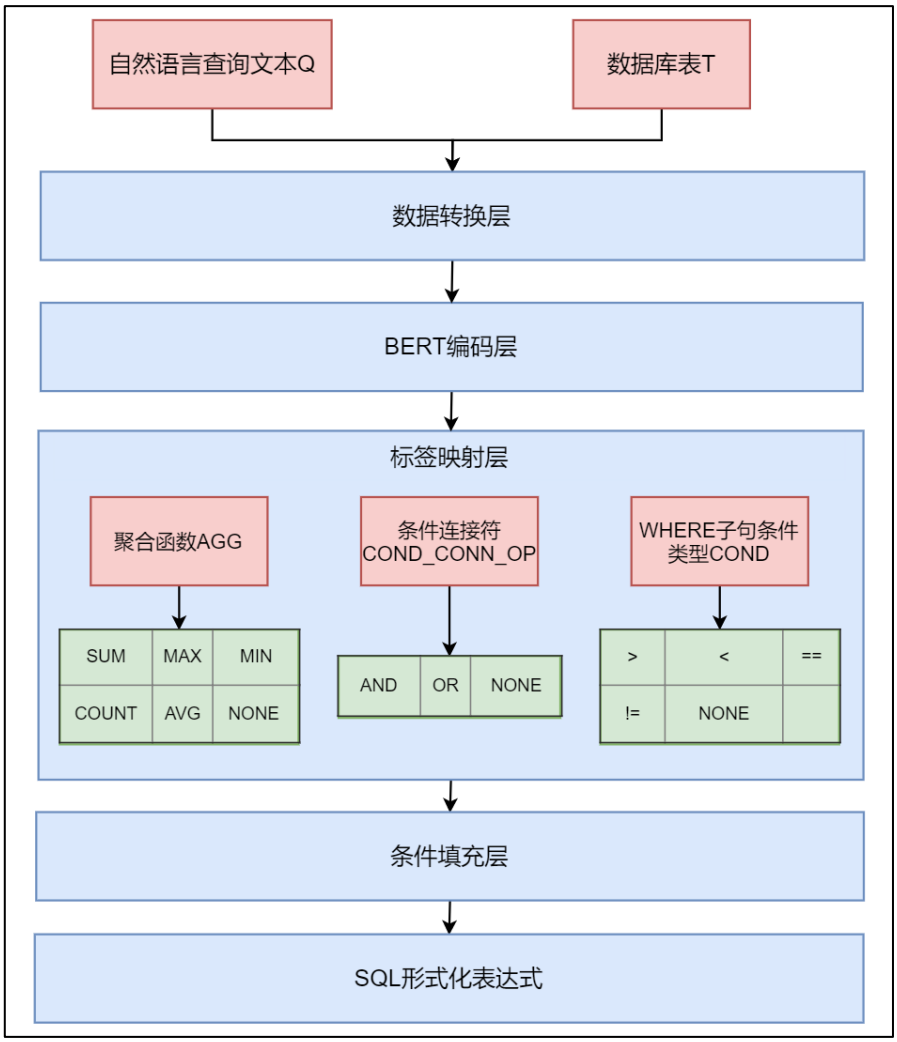

图 3-9　Text-to-SQL 模型框架图

根据以上模型架构图构建好神经网络模型后，即可开始将数据序列送入模型进行训练并保存最优的权重参数以供后续系统使用。

一、数据转换层

本文所使用的数据集中自然语言查询语句和对应的数据库表列名都是以文本的形式存在的，因此首先根据数据库表中每一列的属性类型分别加上“TXT”或“NUM”标识符，同时将列名添加到相应的标识符后面形成一个序列片段，最后将所有序列片段融合得到待编码的输入序列。对于图 3-1 所示的训练样例的输入做特征处理如下图 3-10 所示：

CLS	以城市交通，古城遗址为主题的湘西凤凰古城在什么时间段开放	SEP
NUM	景点代码	SEP
TEXT	景点名称	SEP
...	...	...
TEXT	开放时间	SEP
...	...	...

图 3-10　特征变换后的序列

此外在以上原始训练样本的 SQL 形式化格式中，聚合函数 agg 和所选取的列 sel 的长度是相等的，都为 1。但是对于不同的查询语句来说，agg 和 sel 的长度虽然相等但并不都为或者一固定长度，例如需要选取的列为第二列、第三列和第四列时，sel 的长度就变为了 3，因此在模型中采取融合补充的方法，使得每一条训练样本的 SQL 表达式结构中每一项属性值都为一固定长度，让原始数据这两个维度的输入降低一维，转换成为一个向量来表示，更有利于后续算法模型的处理。具体方法为首先将训练样本中原始 agg 序列和 sel 序列统一合并为 agg_new 序列，新的 agg_new 列表表示选中的列以及其使用的聚合函数，agg_new 列表的下标值为选取的列号，对应的值为聚合函数类型，其取值集和为{0，1，2，3，4，5}，与 SQL 结构映射表中的取值保持一致。另外在 agg_new 向量中新增属性值-1，表示位于该下标索引的列不被选中。

如此一来对于同一表格就可以用固定长度的 agg_new 列表来表示其查询的列以及使用的聚合函数了，对于上述训练样本其原始 agg 和 sel 的值分别为 0 和 8，表示查询列为第九列，聚合函数为空，即只需要查询第九列的值。经过数据预处理之后得到新的融合列表为[-1,-1,-1,-1,-1,-1,-1,-1,0],可见列表前面八项的值都为-1，说明数据库表中前八列都不是查询对象。而第九列的值为 0，说明第九项是需要查询的列，其所使用聚合函数为 SQL 结构表达式映射表中的空值，即不采取任何函数操作获取第九列景点开放时间的值。

同样对于数据集中 conds 属性值也存在多个条件值的约束，采用与上述数据融合相似的原理，将 conds 拆分为 WHERE 子句中需要判断条件的列序列以及这些列对应的值列表，处理方式为首先新增 options 向量，其长度等于选取表格的列数，用于表示 WHERE 子句中所选取的列以及其判断条件类型；其次在 options 向量取值中新增属性值-1，值为-1 的列表示该列不在 WHERE 子句中；最后新增 values 向量，其长度等于选取表格的列数，用于表示 WHERE 子句中该列所对应的判断条件值，不被选取的列对应元素为空值，选取的列为实际值。

例如对于上述样本的原始 conds 值为[[6,2,"城市交通，古城遗址"],[1,2,"湘西凤凰古城"]]，经过预处理之后得到的向量 options 为[-1，2，-1，-1，-1，2，-1，-1，-1]，可见在 options 中第二个元素和第六个元素的值为 2，其他位置的元素都为-1。values 向量则为[‘’, ‘湘西凤凰古城’,‘’,‘’,‘’，‘城市交通，古城遗址’，‘’，‘’，‘’]，说明第二列和第六列需要经过判断比较的值为“湘西凤凰古城”和“城市交通，古城遗址”。综合 options 向量和 values 向量，加上条件判断类型映射表中 2 表示“=”，可推断出表示的内容为 WHERE 子句中判断条件为第二列景点名称为“湘西凤凰古城”且第六列景点主题为“城市交通，古城遗址”。综合以上变换，得到处理之后的对应数据格式如图 3-11 所示：

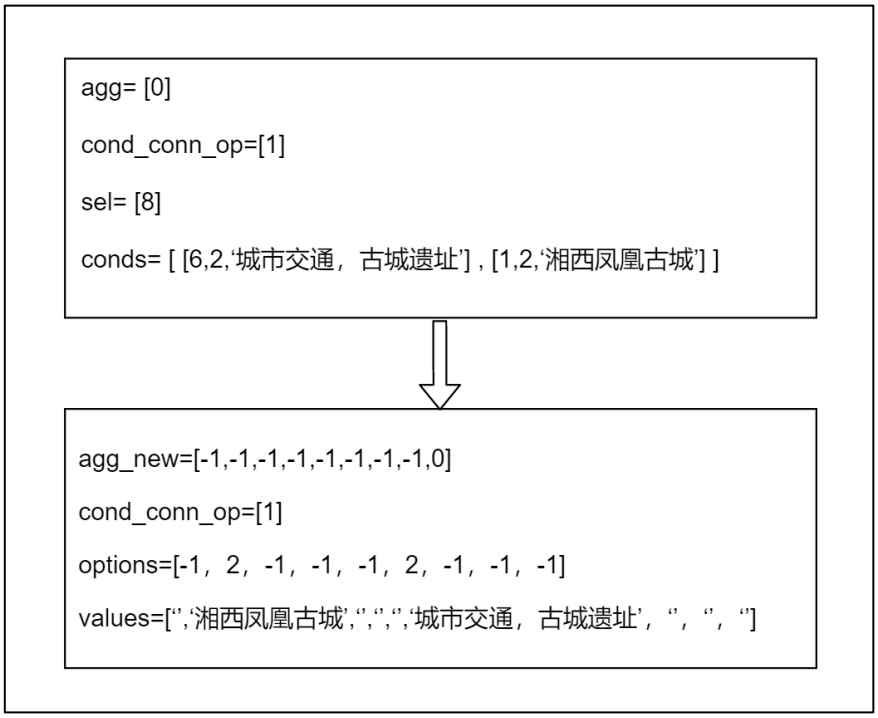

图 3-11　标签转换实例

同时变换后的 SQL 特征表达式映射表如下表 3-3，表 3-4 所示：

表 3-3 agg_new 映射表

NONE	-1
‘’	0
MAX	1
MIN	2
COUNT	3
SUM	4
AVG	5

表 3-4 options 映射表

NONE	-1
>	0
<	1
=	2
!=	3

由此一来即可将原始转化任务变为一个多类别分类的任务来进行处理，分别为对 SELECT 子句的预测任务和对 WHERE 子句的预测任务，其结构如图 3-12 所示：

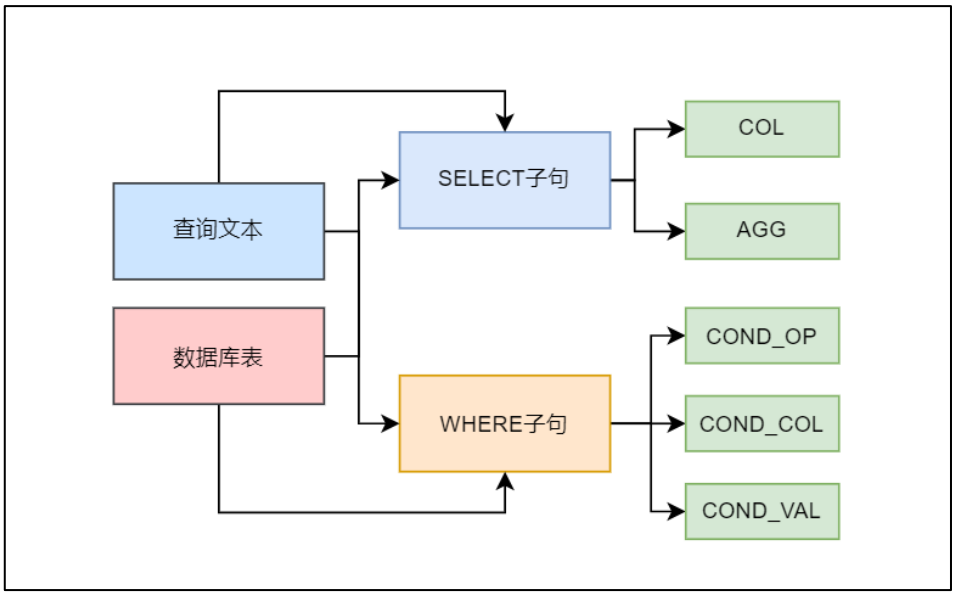

图 3-12　模型子任务结构

二、BERT 编码层

经过上述对输入数据的处理之后，即可将其经过 BERT 预训练模型转换成一个新的输出向量，使其能够更好地作为后续流程的输入。具体做法是取自然语言查询文本以及数据库列名属性组合起来的序列作为 BERT 模型的输入，输出即为经过编码之后的向量表示，其结构表达式如公式 3-4 所示：

$$\{E_{CLS}，H_Q，E_h\} = BERT(Seq) \tag{3-4}$$

在上述公式中，Seq为如图 3-6 所示的在数据转换时得到的自然语言查询语句和数据库列名拼接得到的序列，BERT为经过 BERT 预训练模型的编码，E_h 为列名及其对应的数据类型的输出向量集和，H_Q 为原始自然语言查询文本的输出向量集和，E_{CLS} 为特征标志位 CLS 的输出，用于表示输入的整体特征表达。该层的结构示意图如图 3-13 所示。

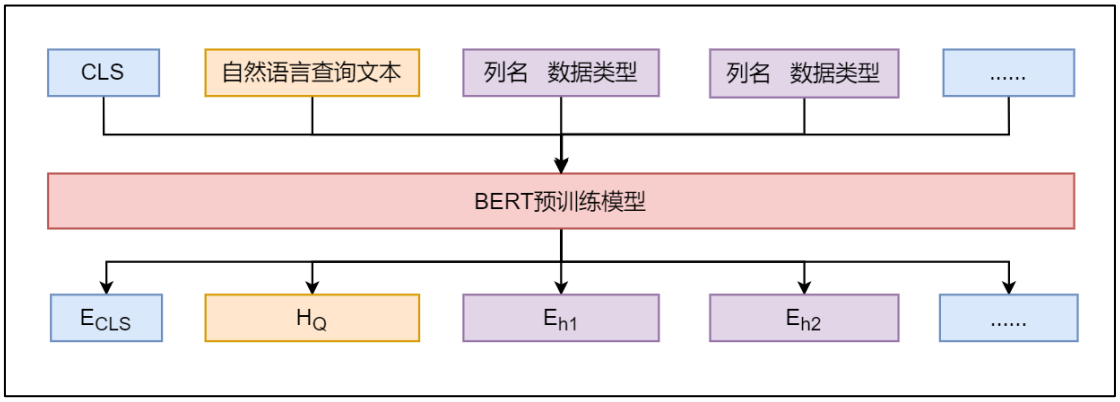

图 3-13 特征向量的输出

三、标签映射层

在前文的数据预处理部分已经将原始数据的列标签 *sel* 和聚合函数标签 *agg* 融合在一起形成了新的向量 *agg_new*，对于 *agg_new* 的取值集和为{-1，0，1，2，3，4，5}。因此可将 SELECT 子句预测的过程转化成一个七分类问题，即将输入序列分类为这七个操作类型中某一个具体的值，用于预测出哪些列被选中以及其对应的聚合函数。类似地对于 WHERE 子句的预测，*options* 取值集合为{-1，0，1，2，3}，即为一个五分类问题，分别对应对于条件列的五种操作类型；*cond_conn_op* 取值集和为{0，1，2}，是一个三分类问题，分别对应多个条件列之间的连接关系。本文采用注意力机制来提取在特定列名下的自然语言查询文本中重要的信息，减少其他文本词汇对其的影响，从而获得对于这一文本而言的特征向量 E_{QH}，计算向量 E_{QH} 如公式 3-5 所示：

$$E_{Q_H} = Attention(E_H, H_Q, H_Q) \tag{3-5}$$

Attention注意力机制中的 query 为E_H，key 和 value 均为 H_Q，H_Q 是然语言查询文本经过 BERT 预训练模型编码之后的输出集和，E_H为数据库表中列名的输出集合。

本文将各个子句中关键字的填充问题转化为多分类问题，不同的分类模型统一使用全连接层将序列较长的向量进行降维处理，其主要不同之处为分类模型的待分类种数存在差别。结合分类模型的特点，将分类列表 L 的分类概率分布P_L的计算方法定义为如公式 3-6 所示：

$$P_L = softmax\big(W * Activation(V)\big) \tag{3-6}$$

概率分布列表的长度与分类模型的分类数一致，W 为参数矩阵，其参数值是可以根据模型的训练进程而改变的。Activation 表示激活函数，表示该节点在给定的输入集合下，计算输出值的方式。softmax 函数用于将序列中的每一个数值都转化为一个概率值，它能将一个 n 维向量 V 变换到另一个 n 维实向量 V 中，并能够保证每一个元素的范围都在 0 到 1 之间，同时所有元素之和为 1。softmax 函数的通用公式如公式 3-7 所示：

$$y_i = \frac{e^{x_i}}{\sum_{i=1}^{N} e^{x_i}},\ i = 1,2,\dots,N \tag{3-7}$$

其中 N 表示向量的维度，也代表着分类模型中类别的多少，x_i表示未经过 softmax 函数处理时向量序列第 i 个位置的值，y_i表示经过 softmax 函数计算后向量序列第 i 个值或分类第 i 类的概率值，其中y_i值的总和为 1。如此可得到以上三个分类任务的每一个值的概率分布如公式 3-8、公式 3-9 和公式 3-10 所示：

$$P_{agg_{new}}{}^{i} = softmax(W_a * E_{Headers}) \tag{3-8}$$

$$P_{cond_{op}}{}^{i} = softmax(W_b * E_{CLS}) \tag{3-9}$$

$$P_{options}{}^{i} = softmax(W_c * E_{CLS}) \tag{3-10}$$

式中W_a、W_b、W_c为模型中可训练的矩阵参数，随着模型的迭代训练以及模型的不断优化，它们的值也会随之改变，最终会确定最优的值并保存在模型结构中。$E_{Headers}$为查询数据库表的表头组成的特征向量，E_{CLS}为输入的序列整体的特征向量。最后根据 $P_{aggnew}{}^{i}$ 的值来确定是表格中的哪一列被查询到以及使用的聚合函数，$P_{options}{}^{i}$ 用于选择判断条件中哪一列作为条件以及条件类型，$P_{condop}{}^{i}$ 则用于预测不同条件之间的连接符。标签映射层的神经网络结构如下图 3-14 所示：

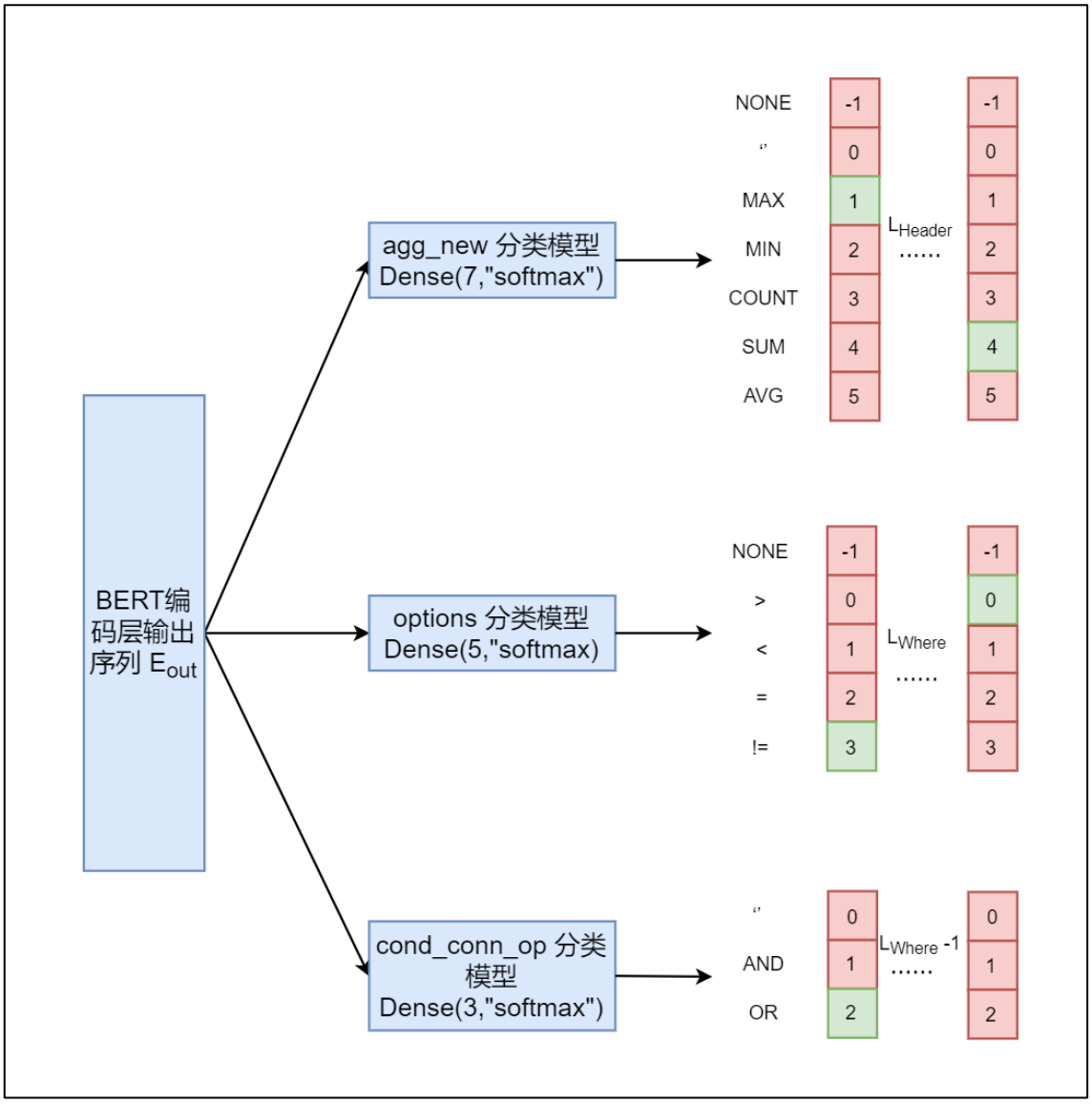

图 3-14 标签映射层结构

在上述标签映射层的结构图中构建了三个深度神经网络模型，每一个分类模型的输入均为之前的 BERT 编码层的输出序列 E_{out}，即图 3-13 所示的向量序列。经过神经网络模型后能够准确映射到如图 3-11 所示的处理后的数据集标签序列，模型训练的同时权重参数也随时更新优化。对于 *agg_new* 向量的分类模型为七分类模型，主要将序列 E_{out} 转化为长度为 L_{Header} 的 *agg_new* 向量序列，L_{Header} 为待查询数据库表表头的长度；对于 *options* 向量和 *cond_conn_op* 向量的分类模型分别为五分类模型和三分类模型，L_{Where} 为 WHERE 子句中查询条件的数量。

四、条件填充层

接下来的任务是将最后 WHERE 子句中的条件值填充到之前预测的每一个所选条件列的后面。在本文所使用的数据集中，数据库表中的字段数据类型包含文本（TEXT）和数值（NUM）类型，在此层中对于这两种类型条件值的处理也有所不同。目前在处理条件值的填充问题时，大多数模型采用的方式为直接从自然语言查询文本中提取出字段来作为填充值。但是这种方式在直接从查询文本中获取的条件值与数据表中存储的条件值表述不一致的情况下，往往会导致条件值不匹配的情况，例如表 3-5 所示：

表 3-5 自然语言查询表述与数据库列值不一致的样例

自然语言查询文本	数据库表中存储内容
什么公司在 **2018 年 7 月 17 号**生产了鲜牛奶	2018/07/17
专项经费预算超过 **1000 万**的项目有哪些	1000.00
商标是 **lenovo** 并且产品是便携式计算机的型号一共有多少种？	联想
13 年的乒乓球女子团体世界杯比赛中有多少支队伍参加了比赛	2013

在本文所使用的数据集中也存在着大量上述的情形，为了解决表述不一致的问题，本文使用以下候选列分类的方式来处理。

对于数据类型为文本的列，由于在 SQL 语句中对于文本的条件比较符只能是“=”，所以只有 WHERE 子句中的条件值与数据表中的存储值一致时才能在查询时正确地匹配。本文通过枚举 WHERE 子句中条件列中所有的值来构建候选列，选取图 3-3 中所示的 SQL 语句，当预测对象列为“景点名称”时，其比较操作符只能为“=”，所构建的候选列为“景点名称”这一列所有的文本值的组合，其候选样例如下图 3-15 所示：

TXT 景点名称=三亚蜈支洲岛 SEP
TXT 景点名称=嘉兴西塘古镇 SEP
TXT 景点名称=湘西凤凰古城 SEP
TXT 景点名称=成都熊猫基地 SEP
.....

图 3-15 文本类型生成候选条件值

将所有可能的条件值序列拼接在之前的输入序列之后，在模型中只需要判断其中的每一个候选序列是否是 WHERE 子句中的一个条件。类似地使用输入序列的整体特征表达向量E_{CLS}，对其构建一个二分类模型，用来将构建的每一条候选列分类为是

否是对应 SQL 语句的一部分。在此情形下文本类型的条件值分类概率分布P_{Txt}如公式 3-11 所示：

$$P_{Txt} = softmax(W_d * E_{CLS}) \quad (3\text{-}11)$$

相比于文本类型的数据列，数据类型为数值的列有着更多的比较操作符取值，其比较操作符可能的取值范围为{“>”，“<”，“=”，“!=”，“>=”，“<=”}。与文本类型的直接枚举数据库存储的列值来构建候选列的方式不同，对于数值类型的条件值首先将查询文本中存在的数值抽取出来，并使用正则表达式将其归一化为数据库表中的表示方式。在此过程中数值类型的抽取和归一化过程都是采用类似的正则表达式完成的，对数值类型的数据归一化后的格式样例如表 3-6 所示。

表 3-6　数值类型数据的归一化示例

自然语言查询文本	标准化数据格式
专项经费预算超过 **1000 万**的项目有哪些	1000.00
13 年的乒乓球女子团体世界杯比赛中有多少支队伍参加了比赛	2013
市盈率多少的股票交易价格高于十元或者涨幅大于百分之八	10，0.08

对于查询语句“市盈率多少的股票交易价格高于十元或者涨幅大于百分之八”而言，选择交易价格为条件列，则对应的可选择的数据值有 10 和 0.08，形成的序列如图 3-16 所示：

NUM　交易价格大于 10　SEP
NUM　交易价格大于 0.08　SEP

图 3-16 数值类型生成候选条件值

上述候选条件值中，“高于”字段是根据模型在上一步预测的值转化得到的。将所有可能的数值条件值序列拼接在之前的输入序列之后，同样地在模型中只需要判断其中的每一个候选序列是否是 WHERE 子句中的一个条件。使用输入序列的整体特征表达向量E_{CLS}，构建二分类模型，用于将所构建的候选条件列分类为是否为 SQL 语句的一部分，分类概率分布P_{Num} 的计算方法如公式 3-12 所示：

$$P_{Num} = softmax(W_e * E_{CLS}) \quad (3\text{-}12)$$

经过以上分析，由此可以得到条件填充层的网络结构如图 3-17 所示，经过二分类模型后得到结果 0 和 1，0 表示所构建的候选列不是 SQL 语句的一部分，1 表示为 SQL 语句的组成部分。根据条件填充层输出的结果即可将指定的条件列以及其条件值填充到 WHERE 子句部分来生成完整的 SQL 表达式。

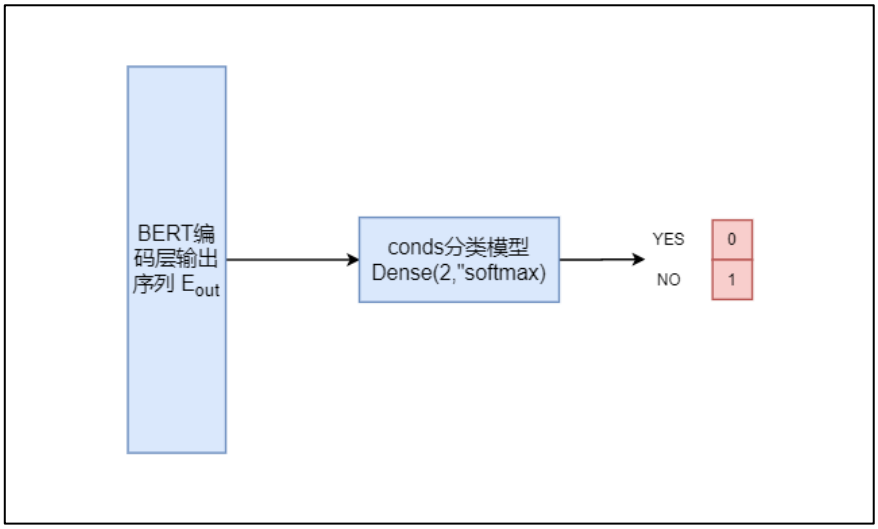

图 3-17 条件填充层结构

上述内容为模型从输入到输出的全部建模流程，综合以上内容，得到如表 3-7 所示的整体 Text-to-SQL 算法描述：

表 3-7 中文 Text-to-SQL 模型描述

输入：中文自然语言查询文本 Q，数据库表 T **输出：目标 SQL 特征表达式 S**
第一步：SELECT 子句预测 1.根据输入的 Q 和 T 构建如图 3-7 所示的模型输入序列 *Seq* 2.将 *Seq* 输入模型，经过 BERT 模型后得到特征向量编码$\boldsymbol{E}_{\boldsymbol{Headers}}$和$\boldsymbol{E}_{\boldsymbol{CLS}}$ 3.把上述得到的特征向量输入到分类模型中，计算选取的列 *sel* 和聚合函数 *agg* 的各类别概率分布 4.根据各类别概率分布排序，确定具体的列 *sel-s* 和聚合函数 *agg* 类别
第二步：WHERE 子句预测 5.根据特征向量得到条件列的概率分布，确定条件列 *sel-w* 6.按照条件列的所属类别为文本类型和数值类型，执行不同流程构成候选列 7.如果为文本类型，直接将数据库表 T 中 *sel-w* 下的所有取值构建如图 3-4 所示的候选列 8.如果为数值类型，首先将查询文本中的数值标准化成图 3-6 所示的数据类型，再构建成如图 3-15 的候选列 9.将上述候选列拼接到输入序列之后，得到具体的条件类型以及条件值
第三步：子句合并 10.将 SELECT 子句和 WHERE 子句合并，得到最终的模型输出，即 SQL 特征表达式

接下来即可将训练数据集输入到模型进行训练，并使用测试集进行转化准确率测试并分析模型结果。

五、模型目标函数

目标函数又称之为损失函数，在深度学习模型不断优化的过程中，其是模型参数更新的核心依据，设计一个合适的目标函数对于一个模型优化来说至关重要。损失函数通过奖励正确的预测，惩罚错误的预测的方式，使得模型整体误差降到最低。损失函数用于测量模型输出结果和真实标签之间的误差，利用误差反向传播的算法计算各个模型层参数的更新范围，在此期间同步更新模型中可变的参数值，直到误差小于预设值或达到目标训练的周期。

本文的 Text-to-SQL 模型涉及的所有子任务都为分类模型，因此在整个训练过程中，模型的损失函数值即为所有的分类模型损失函数之和，在本文中以损失函数总和的不断最小化为依据，进行模型的参数更新。对于本文构建的模型而言，选取交叉熵损失函数作为模型目标函数，如公式 3-13 所示：

$$loss\big(P(x)，\ y\big) = -\left\{\sum_{i=1}^{N}\Big(y_i log_2 P(x_i) + (1-y_i) log_2\big(1-P(x_i)\big)\Big)\right\} \tag{3-13}$$

其中 N 表示在模型分类过程中，所要分类的类别总数，y_i表示第 i 类的标签值，取值为 0 或 1，$P(x_i)$ 表示第 i 类预测结果的概率值。公式中表达的为分类模型的损失函数，在本文所构建的模型中，其由两个子任务构成，因此整体目标函数是预测 SELECT 和 WHERE 子句两个子任务损失函数值之和。

第四节　实验结果及分析

本文所设计模型的搭建以及训练过程所使用主要硬件以及软件配置如图 3-18 所示:

主机 GPU 型号：NVIDIA GeForce RTX 2080Ti
主机内存：48GB
Python 版本：Python 3.7.6
BERT 版本：BERT-wwm，Chinese
模型框架：TensorFlow-gpu 1.14，Keras 2.2.4

图 3-18 训练所使用的硬件及软件配置

训练过程中设置每一个批次的数据量为 32，总迭代次数为 20 次，同时在训练阶段采用了一系列能够增强数据表示的操作来提升模型性能，例如将对于同一数据库表格的查询文本打乱顺序，使其均匀地分布在数据集后再输入到模型中。最终耗时 22 小时完成了数据集的训练。

在传统的深度学习模型中，主要选取模型预测的准确率(Accuracy)来作为评估模型好坏的标准，其中准确率的定义如公式 3-15 所示：

$$Score=\begin{cases} 1, & predict = real \\ 0, & predict! = real \end{cases} \tag{3-14}$$

$$Accuracy=\frac{1}{N}\sum_{i=1}^{N} Score_i \tag{3-15}$$

其中 *predict* 为模型输出的预测值，*real* 为真实正确的值，如果两者相等，则计数加一，最终的准确率为预测正确的数量占所有待预测输入数量的比重。在本文研究的中文 Text-to-SQL 任务中，对于 SQL 转化任务准确率这一标准有两种评价标准，分别为特征一致准确率（Feature Accuracy）和执行结果准确率（Execute Accuracy）。特征一致准确率为预测的具有本文数据集中形式化表示的 SQL 语句与真实正确的形式化 SQL 语句一致，同时列名的顺序不影响这一准确率的评估。执行结果准确率为将形式化表示的 SQL 语句解析成可执行的 SQL 语句后，预测的 SQL 语句和真实的 SQL 语句最终的执行结果一致。将上述一般的准确率计算方式推及到这两种准确率的计算方式如公式 3-17 和公式 3-19 所示：

$$Score_{FA}=\begin{cases} 1, & predict_{FA} = real \\ 0, & predict_{FA}! = real \end{cases} \tag{3-16}$$

$$Accuracy_{FA}=\frac{1}{N}\sum_{i=1}^{N} Score_{FA}^{i} \tag{3-17}$$

$$Score_{EA}=\begin{cases} 1, & predict_{EA} = real \\ 0, & predict_{EA}! = real \end{cases} \tag{3-18}$$

$$Accuracy_{EA}=\frac{1}{N}\sum_{i=1}^{N} Score_{EA}^{i} \tag{3-19}$$

由其定义可知$Accuracy_{FA}$所表现的准确率更为严格，以及结合本文所选用的 NL2SQL 数据集的特点，选用$Accuracy_{FA}$来作为评估模型的标准。同时为了验证本文提出模型在真实 SQL 查询环境中的有效性，选取前文研究实践现状中部分现有的模型进行实验对比，通过 SQL 语句各部分转化的$Accuracy_{FA}$来分析模型的性能。

测试集总共包含 4086 条自然语言查询文本及其对应的 1102 个数据库表，在此测试集上进行模型测试，以上各个模型的 SQL 形式化表达式各部分结构的准确率如表 3-8 所示。其中 acc_{sel} 为转化结果中 SQL 结构中 SELECT 子句所选取的列与数据标签中的列对比的准确率，acc_{agg} 为 SQL 结构中聚合函数转化的准确率，$acc_{cond\_conn\_op}$ 为

条件连接符的转化准确率，acc_{conds} 为 WHERE 子句中所选取的条件列的准确率，而 acc_{total} 则为其整体 SQL 形式化表达式的转化正确率。

表 3-8 模型准确率

	acc_{sel}	acc_{agg}	$acc_{cond\_conn\_op}$	acc_{conds}	acc_{total}
Seq2Seq	0.235	0.295	0.314	0.186	0.174
Seq2SQL	0.652	0.671	0.552	0.526	0.511
SQLNet	0.739	0.726	0.651	0.643	0.637
TypeSQL	0.783	0.754	0.735	0.712	0.703
RAT-SQL	0.816	0.811	0.804	0.793	0.785
本文	0.875	0.866	0.862	0.858	0.852

通过分析以上实验结果可知，本文所提出的中文 Text-to-SQL 模型对于 SQL 语句各个部分转化的准确率都有着良好的效果，同时也能够从上述表中对比得出不同模型的优劣所在。Seq2Seq 模型在此数据集上的准确率最低，主要是因为此模型只是将 Text-to-SQL 任务简单地看作是一个基础的序列到序列的生成任务，没有充分地考虑 SQL 语句中各部分子句之间的结构关系，导致在语句转化时只关注了内容上的对应关系，因此也反映出了在研究 Text-to-SQL 任务时关注 SQL 语句结构的重要性。Seq2SQL 模型在 Seq2Seq 模型的基础上，将 SQL 语句的结构特点加入其中并选取输入序列中的部分有关的文本信息作为 SQL 语句的组成部分得到的，改进了 Seq2Seq 模型忽略 SQL 结构关系的不足，从准确度结果表中也可以看出在各个部分都有了较大的提升。SQLNet 模型将转化任务当作模板填充问题来解决，也在一定程度上提升了转化的准确率。TypeSQL 则是在 SQLNet 基础之上，增加对查询文本的关键信息字段实体标注操作，并将其作为辅助信息来完成转化任务，所以在转化时能够更精准地将文本字段和 SQL 语句中的字段匹配，从结果表中 acc_{sel} 和 acc_{agg} 这两个准确度的提高即能反映出这一优化之处。RatSQL 将数据库的结构以及列名信息加入到模型之中，充分利用了数据库中字段名信息来进行转化，极大地提高了条件值转换的准确率。

在本文提出的模型中，究其本质而言也是一个序列到序列的生成模型，但是在其基础之上加入了 BERT 预训练模型，解决了语义编码上的问题；同时将单个任务分解成两个子任务来进行，每个子任务中又分别包含着多个分类模型，实验过程中通过不断优化分类模型即可提升整体模型的效率，减少模型复杂程度的同时也减少了不必要的中间变量；最后借鉴了 RatSQL 的思想，在确定条件值的时候分别针对数值和文本类型构建了不同的分类模型，直接从数据库列名中匹配最为合适的字段作为条件值，提升了模型在条件值转化上的准确率，最终模型在测试集上的 SQL 表达式转化的整体正确率达到了 0.852。

此外为了验证本文构建的条件值分类模型对于自然语言查询文本和数据库表存储字段的中文表述不一致问题上的改善效果，选取了测试集中两者表述不一致的样例共计 352 条进行实验，对于以上模型的转化结果如表 3-9 所示。

表 3-9 表述不一致模型准确率

	Seq2Seq	Seq2SQL	SQLNet	TypeSQL	RAT-SQL	本文
acc_{total}	0.014	0.019	0.122	0.125	0.173	0.838

通过分析上述实验结果可以得出在自然语言查询文本和数据库表存储字段的中文表述不一致时，本文提出的 Text-to-SQL 模型较其他模型的准确率有着极大的提升。例如在自然语言查询文本中的关键字段为“首都”，而在数据库表中存储的内容为“北京”，“首都”这一字段没有直接出现在数据库表中，其他的几种模型在在 SQL 语句转化时得到的结果仍为“首都”，因此在执行 SQL 语句时无法在数据库中找到与之匹配的内容，导致查询结果有误。本文的模型则利用了数据库表中存储的字段信息，通过一一列举条件值候选列，使得“首都”与“北京”能够正确地匹配，从而提升了在条件值转化上的准确率，以至于整体转化准确率得到了极大的提高。

图 3-19 表示的是本文模型在训练过程中准确率变化的过程，其横坐标是训练过程中迭代的次数，纵坐标为对应迭代次数下的准确率。可以看出模型最初始状态下的准确率不高，但随着迭代次数的增加，准确率不断提高，在第五次迭代时准确率开始保持平稳增长。同时还可以分析出对于 SQL 语句中条件值的转化初始准确率最低，表明了对于 WHERE 子句的预测的难度大于 SELECT 子句部分，分析其原因是由于 WHERE 子句部分不仅需要通过模型预测所选取的条件列，还要通过条件值填充的方式来完成 WHERE 子句的整体结构，故而初始准确率较低。

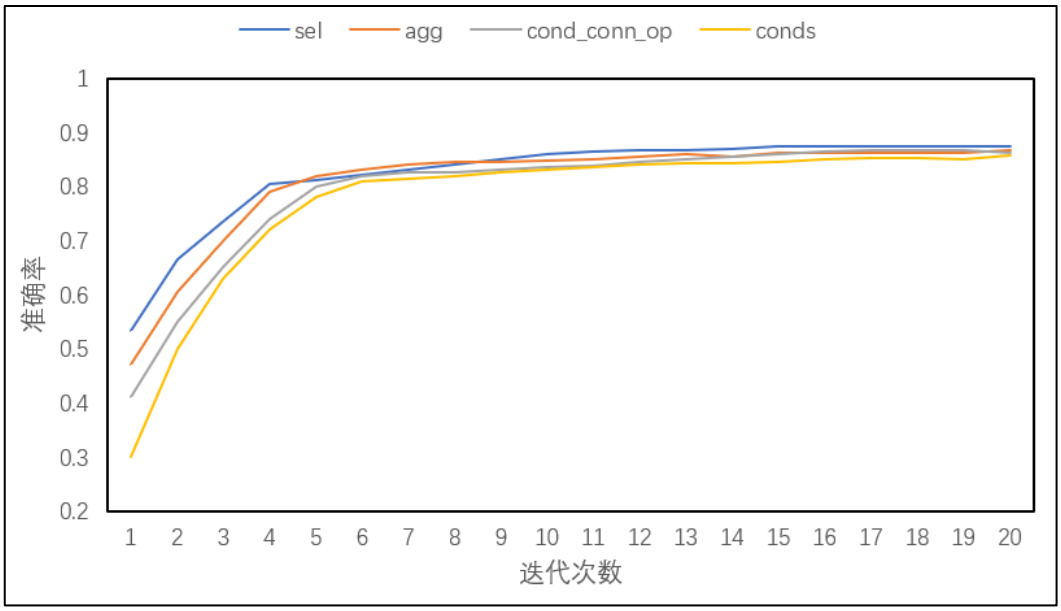

图 3-19 模型准确率变化过程

同时对于本文的模型而言，所有的测试样例中每一条查询文本的转化时耗如图 3-20 所示：

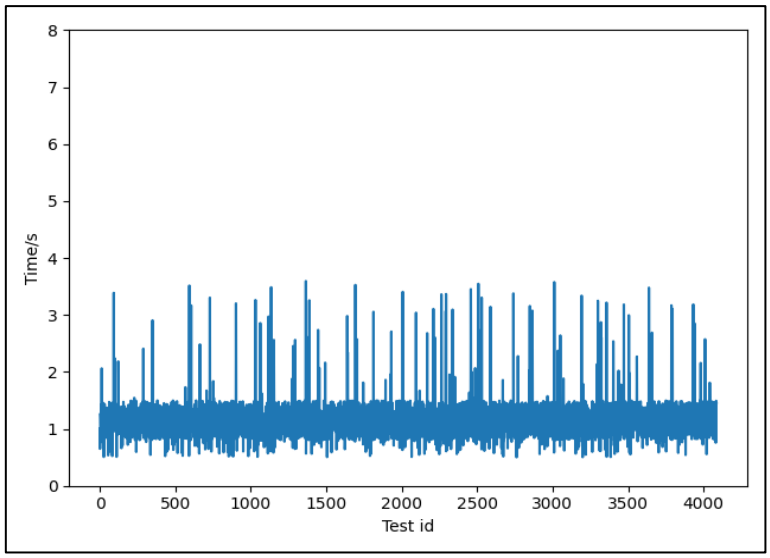

图 3-20 测试集的时耗图

由此可见对于大部分查询样例文本来说，均能在 1.3 秒之内完成转换功能，小部分文本需要耗费 2 到 4 秒时间。分析以上转化准确率和时效性，准确率和时效性都已到达后续系统实现可接受的水准，但在实验过程中仍然还有一些值得改进和优化的地方。

首先从对预处理之后对输入序列的编码分析，在 BERT 的预训练阶段，训练数据时 Mask 掉了部分词汇，而在微调阶段无法观察到 Mask 标记，这两个阶段的不一致会产生一定的性能损失；此外在 BERT 的预训练阶段，句子中有多个单词被 Mask 掉，而在某些情况下这些单词之间是有关系的，如此也会对模型精度造成负面影响。因此可考虑尝试使用诸如 XLNet 等规避了以上问题的模型进行编码来代替 BERT 模型是否会提升模型转化的准确率。

其次考虑在训练阶段时，不单单只是去关注自然语言查询文本本身和 SQL 特征表达式标签之间的关系，可以将训练数据中所有与其同属一个数据库表的查询文本联合起来，经一定处理后再进行编码，这样就做到了全局字段的注意力关注，能够增强输入数据的表示能力。

最后是优化器部分，可以选择不同的优化器来作为模型的参考，如 Adagrad、RMSprop、Adabound 等，通过对比实验来选取最优的作为模型的优化器。

第五节　本章小结

在本章主要对整体模型架构进行了介绍，模型是在 BERT-wwm，Chinese 预训练模型上实现的。首先针对所使用数据集特点进行了预处理操作，使得模型的输入序列更具有语义表示能力。其次将 Text-to-SQL 任务转化成了分类问题，同时将 SQL 语句分为 SELECT 子句和 WHERE 子句两个部分来进行预测，即在 SELECT 子句中将自然语言文本分类为不同的查询列类以及聚合函数类，在 WHERE 子句中将文本分类为条件列、条件类型以及条件值来进行转化，最后将两个子句拼接起来得到 SQL 表达式。同时对训练好的模型在测试数据集上进行了测试验证工作，根据数据集特点采取了$\mathrm{Accuracy}_{\mathrm{FA}}$来作为评估模型的标准，并将本文模型与已有的 Text-to-SQL 模型进行实验对比，分析模型转化结果的同时也提出了可改进以及需要优化的地方。

第四章　中文 Text-to-SQL 系统实现

在前文中进行了自然语言文本到 SQL 语句的模型转化任务，本章的主要工作是将构建的模型应用到实际系统中，从需求分析到具体的设计过程来进行 Text-to-SQL 整体系统的实现，该系统将基于特定数据库的中文自然语言查询文本转化生成对应的 SQL 语句并执行，同时显示结果集在系统界面。

第一节　需求分析

本文所需实现的功能为从中文自然语言查询文本到查询结果的 Text-to-SQL 系统。将以上搭建好的模型运用到系统中，同时返回结果即为系统的输出。系统的核心业务为将前文中训练好的模型与系统界面进行交互，即从系统显示界面获取用户的查询文本信息，将其作为模型输入得到模型输出的 SQL 特征表达式。

系统的功能流程首先是对原始数据库文件的解析操作，即将原始数据库文件解析为模型所需的 json 格式；其次是输入预处理操作，使得系统输入和模型输入保持一致，才能够进行后续的模型转化工作；接下来是整个系统的核心部分，也就是将自然语言查询文本转化为 SQL 形式化表达式的模型交互；最后是将 SQL 表达式转换成可执行的 SQL 语句并交由程序执行，同时返回结果集到显示界面。经分析后系统整体架构设计如图 4-1 所示：

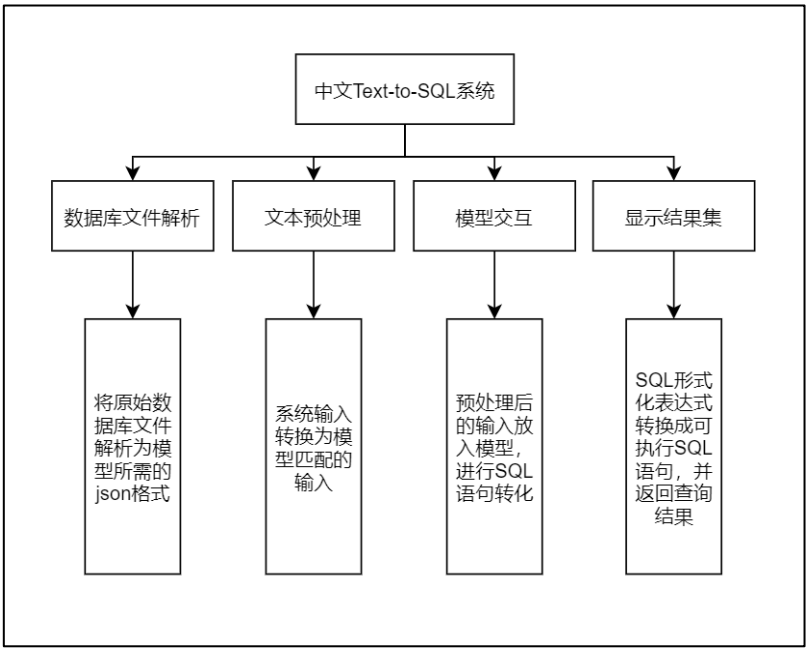

图 4-1 系统结构

对于用户交互界面而言，系统需要保证尽可能地将原始查询文本信息、查询结果以及当前数据库中所有数据库表显示出来，以供用户了解到查询的具体信息，同时系统具有历史记录的功能，即用户可以查看之前的查询数据结果。

第二节　系统设计与实现

本文所要设计的系统为一个基于数据库的对话问答系统，其对应的任务是根据用户输入的自然语言查询文本来返回结果集给用户。在前文搭建的模型中，转换两个查询子句时需要将对应数据库表的表头作为输入系列的一部分，而存储在数据库系统中的数据库表中的内容是外界程序无法直接获取到的。因此在本系统中，首先将这些数据库表的结构和内容解析成 json 文件格式，以供后续模型构建候选列时使用，具体解析之后的文件格式如图 4-2 所示：


```
"name": "Table_a7b048383b0611e99038f40f24344a08",
"title": "图书详情",
"header": ["索书号","书名","编著者","出版社","出版时间","册数"],
"types": ["text","text","text","text","real","real"]
```


图 4-2 解析之后的数据库表结构

上述结构中 name 表示在原数据库中该数据表的 id，即 SQL 语句中 FROM 字段后需要填充的内容，title 为数据表的内容标题，header 为每一列的列名构成的一个列表，即数据库表的表头集和；types 为每一列的具体属性类型，text 表示是文本类型，real 为数值类型。如此一来，在模型中进行 SELECT 和 WHERE 子句预测时就可以参照图 3-10 所示的特征变换，将 header 和 types 列表融合起来，进行输入序列的变换得到新的序列，然后将新的序列作为模型输入序列的一部分。

在数据文件全部加载就绪之后，同时将之前通过模型训练得到的最优权重文件 best_weights.h5 加载到系统中，即可开始实现 Text-to-SQL 系统的输入操作。系统整体图形化界面采用 QTCreator 进行设计，其显示界面设计如图 4-3 所示。输入阶段采用文本输入的方式，使用文本框来作为自然语言查询文本的获取容器，即将需要查询的自然语言语句通过键盘输入的方式输入到系统中。查询操作通过按钮设计完成，即文本输入完成之后点击该按钮开始进行转换查询操作，结果显示栏以及历史记录栏均采用文本域来进行显示。

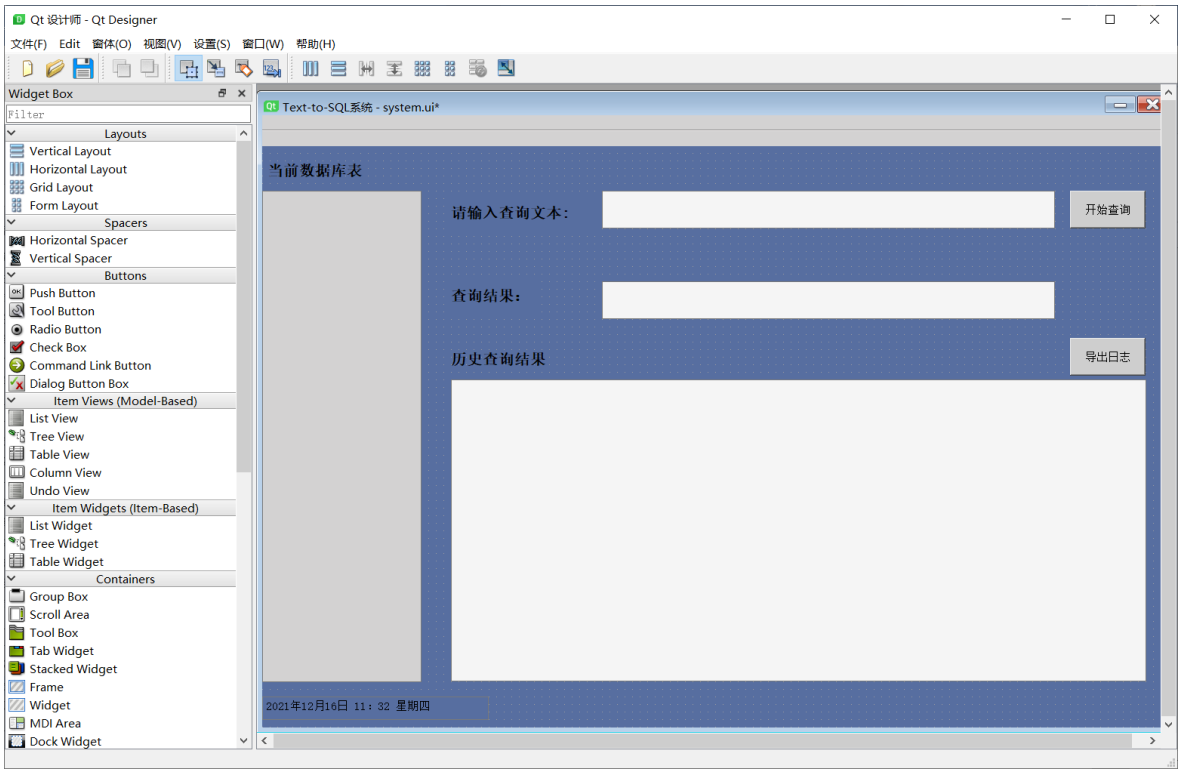

图 4-3 系统显示界面设计

系统接受输入文本之后，根据文本中所要查询的列从图 4-2 所示的数据库表解析文件中匹配对应的数据库表，获取其数据库表的名称，以供于后续拼接 SQL 语句时将此数据库表名称作为 FROM 关键字之后的内容，只有在正确的数据库表中执行操作才能得到正确的结果集。具体的匹配数据库表的方式是根据列名的重合度来确定对应数据库表，以上处理主要是保证最终 SQL 语句结构的完整以及避免执行结果出错而进行的。例如对于一条查询语句为“哪个科目的教材是人民教育出版社出版的”的文本，其输入样式如图 4-4 所示：

图 4-4 系统输入样例

在系统设计时为模型编写了接口来接受以上的输入文本，并按照模型需要将文本转化为特定的数据序列，具体做法为采用在模型训练时使用的数据预处理方法一致，将问题和表格具体内容融合在一起并编码作为模型的输入。在文本框中输入完自然语

言查询文本后，即可点击界面中的“开始查询”按钮进行查询，待查询文本输入完成以及模型序列转换结束之后则进入模型进行下一阶段的转换。

在 SQL 语句转化过程中同样是首先根据文本内容选择出待查询的列以及其对应的聚合函数，然后再将 WHERE 子句部分的查询条件进行转化，最后将两者拼接并按照 SQL 结构映射表进行替代得到最终的 SQL 语句。其整体流程如图 4-5 所示：

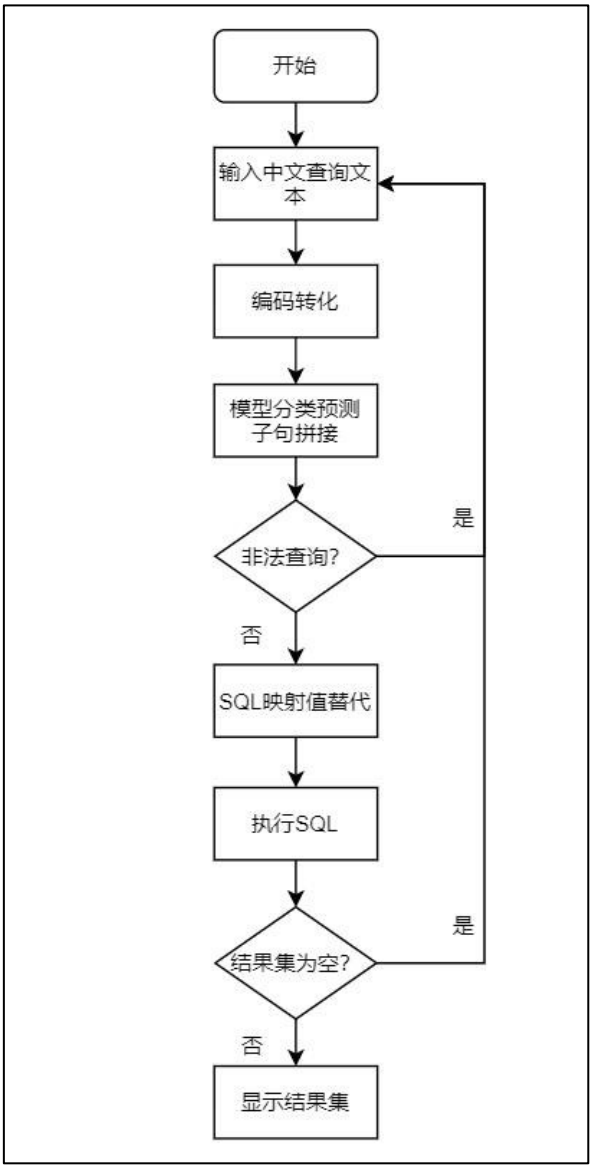

图 4-5 系统流程图

从系统流程图来看，系统首先将输入的文本预处理后得到的编码序列通过接口送入到模型中，其次模型分别预测出 SELECT 子句和 WHERE 子句所选取的列、聚合函数以及 WHERE 连接符和条件值，最后将两个子句部分结合即可得到具有 SQL 语义的形式化表达式，再根据模型构建过程中的 SQL 结构映射表字典进行映射值替代，将其转换成可执行的 SQL 语句。如果存在非法的查询语句，即查询文本中所涉及的

关键字在已有的数据库中不存在，系统在匹配对应数据库表时会出现无法匹配的情况，从而在系统界面给出非法的查询提示，其具体样例如系统测试部分图 4-9 所示。

得到最终的 SQL 语句后，使用 python 内置的 sqlite3 库进行数据库连接，执行此 SQL 语句返回结果集并显示出来，此结果就是输入的自然语言查询文本的查询结果输出。若数据库中对于查询文本的检索结果为空时，即查询文本中的关键字在数据库中有对应的表列名与之匹配，但表中无此具体的数据项，系统也会在界面上提示出查询结果为空的提示，其具体样例如系统测试部分图 4-10 所示。同时历史查询的记录也会被保存在历史查询结果的显示区域内，可根据需要进行查询日志的导出功能，方便后续的查阅检索工作。例如对于查询问题为“总有一条路通向故乡和李花是由哪个出版社出版的”的文本，其检索结果为“长江文艺出版社”，其查询文本的输入和对应的结果输出界面显示如图 4-6 所示。

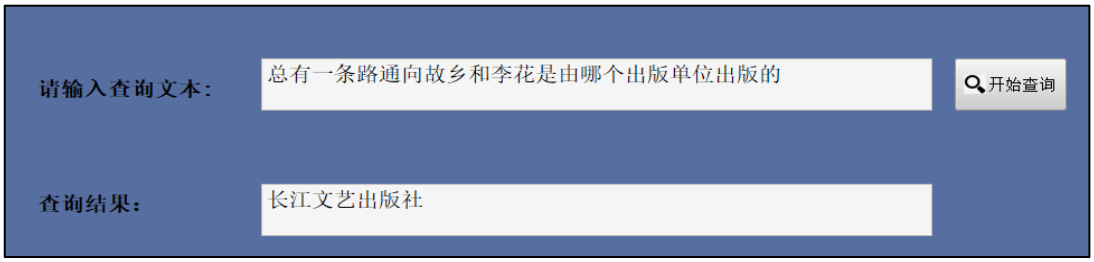

图 4-6 查询实例

将上述设计流程整合起来，得到最终的系统界面如图 4-7 所示：

图 4-7 系统界面

在上述系统界面中，最左侧一栏为当前数据库中所有存在的表，后续的查询操作都是基于这些数据库表的内容进行。界面中间的输入框为对所查询数据的文本化描述，其下面的显示标签为查询结果的显示区域。最下面一栏的文本显示区域为历史查询记录的显示，包括了查询时间、查询文本、对应的 SQL 语句以及其执行结果的显示面板。同时导出日志按钮即可将所有查询历史导出到文件系统保存，方便用户后续查看时无需再执行重复的查询操作。

第三节　系统测试

进行完系统的设计实现之后，再根据实际生产环境对系统进行测试，以保证其可用性同时优化改进系统可能存在的问题。

制定测试用例时，首先将输入的查询文本设置为数据库中能够查询到实际存在的样例，通过输入查询文本后系统即可检索输出查询结果。如图 4-8 所示为此项测试项目的一个测试实例，其查询文本为“收益估值大于 3 的有哪些公司”，对于此文本能够从数据库查询到存在的数据记录，查询结果为“中远海控 广深铁路 中远海能”并在系统界面显示出来。

图 4-8 查询结果存在的测试样例

以上输入都是针对数据库中存在的数据并且查询文本都是合法的，所以接下来的测试用例为存在不合法的查询或者在已有的数据库中不存在此项数据的查询。如图 4-

9 所示测试用例为一条不合法的查询实例，其查询文本为“明天天气怎么样”，通过模型转化时会识别出该问句的所有关键字在当前数据库中都没有匹配的数据表，因此系统会出现“非法的查询语句”的提示。图 4-10 则是一条数据库表不存在具体数据项的查询实例，其查询文本为“扬名立万的豆瓣评分是多少”，虽然根据其查询文本的关键字段可以匹配到电影详情的数据表，但是在此数据表中却没有扬名立万这一部电影的具体信息，因此系统会给出“数据库中无此项数据”的提示。

图 4-9 非法的查询文本实例

图 4-10 无待查询数据的查询文本实例

对于以上项目所有测试，其测试结果如表 4-1 所示：

表 4-1 系统测试结果

	平均耗费时间（s）	准确率
正常查询文本	1.8	0.85
非法查询文本	1.5	0.91

上述测试结果表明，正常查询文本和非法查询文本转化的准确率分别为 0.85 和 0.91，可见系统在转化准确率上达到了一定的水准。此外在前文的模型构建时对于模型的结果分析中已有时间性能图分析，即对于测试集中的自然语言文本，大多数查询文本都能够在 1.3 秒之内返回 SQL 语句。同时将模型应用到此系统中时，系统没有其他需要处理的计算过程，因此系统的响应速度也是有保证的。虽然对于个别复杂样例耗费时间较长，但对于所有正常查询文本而言其平均耗费时间为 1.8 秒，非法查询文本则为 1.5 秒，由此可见系统能够在可接受的时间范围内对中文查询文本进行转化处理并显示结果集。

第四节 本章小结

在此章节进行了中文 Text-to-SQL 系统的实现，将前文构建的模型应用到此系统中，首先从原始数据库文件进行解析得到模型输入所需的内容格式，同时对输入文本进行特征化处理，以及将输入序列进行融合处理操作；然后在算法模型实现融合序列到 SQL 语句的转化功能，最后交由系统执行 SQL 并在系统界面显示结果集，完成中文查询文本到最终查询结果的任务。测试结果表明，此系统在准确度和时效性上都有着可靠的保障。

第五章　总结及展望

Text-to-SQL 任务作为自然语言处理领域新兴的一个分支研究，目前正处于发展到趋于成熟的过程，但相较于英文的转化任务而言，对于中文的此项研究还处于落后的阶段。本文主要提出了一种基于 BERT 模型的中文 Text-to-SQL 算法的实现，同时将训练好的模型应用到实际查询系统中，该系统能够实现自然语言文本和数据库的交互，实现了基于数据库的从中文文本输入到所需数据项结果输出显示的功能。

本文的主要工作和贡献如下所示：

（1）本文将 Text-to-SQL 任务转化成两个不同的分类问题来解决，即根据原始的输入文本对 SELECT 子句和 WHERE 子句中各部分结构进行分类预测，并将分类的结果填充到对应的 SQL 形式化表达式的具体位置，最终得到转化的可执行的 SQL 语句，解决了中文自然语言文本到 SQL 的转换问题。

（2）将构建的算法模型应用到实际系统中，系统首先将输入的中文自然语言查询文本经过模型转化后得到对应数据库可执行的结构化查询语句，然后执行此查询语句得到输入文本所对应的结果集并显示，实现了从中文自然语言查询文本到目标结果的功能。

本文提供了一种基于 BERT 模型的中文 Text-to-SQL 系统及其实现过程，取得了一定的阶段性成果，但是针对于此系统而言依然存在着一些不足之处需要在后续工作中改进，主要包括以下内容：

（1）模型所使用数据集为在单表之内查询的数据集，所涉及的查询场景较为单一，SQL 语句结构相对于实际生产环境而言较为简单，若要投入到工业级使用以及使得模型更好地泛化到其他数据库表，还需在此基础之上搭建更为复杂的深度神经网络模型以及使用具有多表查询的数据集。

（2）其次在模型的训练阶段，本文所使用的模型只是去关注自然语言查询文本本身和 SQL 特征表达式标签之间的关系，没有同步考虑到同一数据库表中其他查询文本的内容对其编码结果的影响，后续可以将训练数据中所有与其同属一个数据库表的查询文本联合起来，经一定处理后再进行编码，这样就做到了全局字段的注意力关注，能够增强输入数据的表示能力。

Text-to-SQL 技术具有广泛的应用前景，并且可以直接作为应用接口部署到实际应用系统中，如此一来不仅降低了普通用户数据获取的门槛，使其能够更为便捷地从繁多的结构化数据库表中检索出需要的数据内容，而且大大降低了数据管理人员的工作强度，节省了大量对于数据库管理所投入的人力和物力，从而进一步提高了生产效率，同时也是工业自动化发展进程中的一个实际性的产物。从广泛应用场景来看，此

项技术可以应用于需要进行检索信息的软件或系统中，例如搜索引擎的检索；从具体的应用场景来看，则可以用于实现基于数据库的人机问答系统，通过直接将所需要查询的问题文本输入系统，系统即可在一定时间范围内给出相对应的检索结果，减少用户的中间操作流程。此外尽管本文所使用的中文数据集与真实生产环境中的 SQL 查询相比仍然较为简单，但是对于后续更复杂的多表嵌套查询工作，都能够在此基础之上展开研究，使其能够更好地应用于实际工业环境，提高生产的效率。

参考文献

[1] Zhong V, Xiong C, Socher R. Seq2sql: Generating structured queries from natural language using reinforcement learning[J]. arXiv:1709.00103, 2017.

[2] Yu T, Zhang R, Yang K, et al. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task[C].EMNLP.2018.

[3] Min Q, Shi Y, Zhang Y. A pilot study for Chinese SQL semantic parsing[J]. arXiv:1909.13293, 2019.

[4] Xuan K, Wang Y, Wang Y, et al. SeaD: End-to-end Text-to-SQL Generation with Schema-aware Denoising[J]. arXiv:2105.07911, 2021.

[5] Ranzato M A, Chopra S, Auli M, et al. Sequence level training with recurrent neural networks[J]. arXiv:1511.06732, 2015.

[6] Vinyals O, Fortunato M, Jaitly N. Pointer networks[J]. Advances in neural information processing systems, 2015, 28.

[7] Liu Q, Zeng Y, Mokhosi R, et al. STAMP: short-term attention/memory priority model for session-based recommendation[C].Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2018: 1831-1839.

[8] Guo J, Zhan Z, Gao Y, et al. Towards complex text-to-sql in cross-domain database with intermediate representation[J]. arXiv:1905.08205, 2019.

[9] Yu T, Li Z, Zhang Z, et al. Typesql: Knowledge-based type-aware neural text-to-sql generation[J]. arXiv:1804.09769, 2018.

[10] Xu X, Liu C, Song D. Sqlnet: Generating structured queries from natural language without reinforcement learning[J]. arXiv:1711.04436, 2017.

[11] Li Q, Li L, Li Q, et al. A comprehensive exploration on spider with fuzzy decision text-to-sql model[J]. IEEE Transactions on Industrial Informatics, 2019, 16(4): 2542-2550.

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

[13] Wang B, Shin R, Liu X, et al. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers[J]. arXiv:1911.04942, 2019.

[14] Mikolov T, Chen K, Corrado G, et al. Efficient estimation of word representations in vector space[J]. arXiv:1301.3781, 2013.

[15] Mihalcea R, Tarau P. Textrank: Bringing order into text[C].Proceedings of the 2004 conference on empirical methods in natural language processing. 2004: 404-411.

[16] Dumais S T. Latent semantic analysis[J]. Annual Review of Information Science and Technology (ARIST), 2004, 38: 189-230.
[17] Hofmann T. Probabilistic latent semantic indexing[C].Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. 1999: 50-57.
[18] Blei D M, Ng A Y, Jordan M I. Latent dirichlet allocation[J]. Journal of machine Learning research, 2003, 3(Jan): 993-1022.
[19] Pennington J, Socher R, Manning C D. Glove: Global vectors for word representation[C].Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014: 1532-1543.
[20] Peters M E, Neumann M, Logan IV R L, et al. Knowledge enhanced contextual word representations[J]. arXiv:1909.04164, 2019.
[21] Xu W, Rudnicky A. Can artificial neural networks learn language models?[J]. 2000.
[22] Mikolov T, Karafiát M, Burget L, et al. Recurrent neural network based language model[C].Interspeech. 2010, 2(3): 1045-1048.
[23] Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv:1810.04805, 2018.
[24] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.
[25] Radford A, Narasimhan K, Salimans T, et al. Improving language understanding by generative pre-training[J]. 2018.
[26] Yin P, Neubig G. TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation[J]. arXiv:1810.02720, 2018.
[27] Li H, Wang B, Cui L, et al. LGL-GNN: Learning Global and Local Information for Graph Neural Networks[C].Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition(SSPR). Springer, Cham, 2021: 129-138.
[28] García J, Díaz O, Cabot J. An adapter-based approach to co-evolve generated SQL in model-to-text transformations[C].International Conference on Advanced Information Systems Engineering. Springer, Cham, 2014: 518-532.
[29] Finegan-Dollak C, Kummerfeld J K, Zhang L, et al. Improving text-to-sql evaluation methodology[J]. arXiv:1806.09029, 2018.
[30] Guo A, Zhao X, Ma W. ER-SQL: Learning enhanced representation for Text-to-SQL using table contents[J]. Neurocomputing, 2021, 465: 359-370.

[31] Anisha T S, Rafeeque P C, Reena M. Text to SQL query conversion using deep learning: a comparative analysis (August 15, 2019)[C].Proceedings of the International Conference on Systems, Energy and Environment (ICSEE). 2019.
[32] Sun N, Yang X, Liu Y. Tableqa: a large-scale chinese text-to-sql dataset for table-aware sql generation[J]. arXiv:2006.06434, 2020.
[33] Guo J, Si Z, Wang Y, et al. Chase: A Large-Scale and Pragmatic Chinese Dataset for Cross-Database Context-Dependent Text-to-SQL[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). 2021: 2316-2331.
[34] Choi D H, Shin M C, Kim E G, et al. Ryansql: Recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases[J]. Computational Linguistics, 2021, 47(2): 309-332.
[35] Mikolov T, Zweig G. Context dependent recurrent neural network language model[C].2012 IEEE Spoken Language Technology Workshop (SLT). IEEE, 2012: 234-239.
[36] Ramachandran P, Liu P J, Le Q V. Unsupervised pretraining for sequence to sequence learning[J]. arXiv:1611.02683, 2016.
[37] Deng X, Awadallah A H, Meek C, et al. Structure-grounded pretraining for text-to-sql[J]. arXiv:2010.12773, 2020.
[38] Lin K, Bogin B, Neumann M, et al. Grammar-based neural text-to-sql generation[J]. arXiv:1905.13326, 2019.
[39] Yu T, Zhang R, Er H Y, et al. CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases[J]. arXiv:1909.05378, 2019.
[40] Wang L, Zhang A, Wu K, et al. DuSQL: A Large-Scale and Pragmatic Chinese Text-to-SQL Dataset[C].Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2020: 6923-6935.
[41] Scholak T, Li R, Bahdanau D, et al. DuoRAT: Towards Simpler Text-to-SQL Models[J]. arXiv:2010.11119, 2020.
[42] Cai Y, Wan X. IGSQL: Database schema interaction graph based neural model for context-dependent text-to-SQL generation[J]. arXiv:2011.05744, 2020.
[43] Cai R, Yuan J, Xu B, et al. SADGA: Structure-Aware Dual Graph Aggregation Network for Text-to-SQL[J]. Advances in Neural Information Processing Systems, 2021, 34.
[44] Zhao L, Cao H, Zhao Y. GP: Context-free Grammar Pre-training for Text-to-SQL Parsers[J]. arXiv:2101.09901, 2021.
[45] 曹金超, 黄滔, 陈刚, 等. 自然语言生成多表 SQL 查询语句技术研究[J]. 计算机科学与探索, 2019, 14(7): 1133-1141.

[46] 杨鹤标, 陈力. 自然语言向 SQL 代码的转化方法[J]. 计算机工程, 2011, 37(23): 72-74.
[47] 潘璇, 徐思涵, 蔡祥睿, 等. 基于深度学习的数据库自然语言接口综述[J]. 计算机研究与发展, 2021, 58(9): 1925.
[48] 曹金超. 一种基于深度学习的中文自然语言查询生成 SQL 语句技术研究[D]. 浙江大学.
[49] 仇金娟. 基于序列到序列模型的多轮对话中 SQL 语句生成方法研究[D].北京理工大学,2018.
[50] 吕海熊. SQL 语言到自然语言自动翻译的研究[D].北京理工大学,2016.
[51] 宋健龙. 面向数据库安全的 SQL 语句解析与翻译[D].北京理工大学,2015.
[52] 李子烨. 自然语言到 SQL 语言翻译任务的研究[D].北京邮电大学,2020.
[53] 孟小峰, 王珊. 中文数据库自然语言查询系统 Nchiql 设计与实现[J]. 计算机研究与发展, 2001, 38(9): 1080-1086.
[54] 周奇安, 李舟军. 基于 BERT 的任务导向对话系统自然语言理解的改进模型与调优方法[J]. 中文信息学报, 2020, 34(5): 82-90.
[55] 牛海波, 赵丹群, 郭倩影. 基于 BERT 和引文上下文的文献表征与检索方法研究[J]. 情报理论与实践, 2020, 43(9): 125-131.
[56] 万文军, 窦全胜, 崔盼盼, 等. 结合关系分类与修正的 SQL 语法结构构建[J]. 计算机科学, 47(11A): 562-569.