Spring 2021

Learning to Map Natural Language to Executable Programs Over Databases

Tao Yu
Yale University Graduate School of Arts and Sciences, ty2326@columbia.edu

Abstract

Learning to Map Natural Language to Executable Programs Over Databases

Tao Yu

2021

Natural language is a fundamental form of information and communication and is becoming the next frontier in computer interfaces. As the amount of data available online has increased exponentially, so has the need for Natural Language Interfaces (NLIs, which is not used for natural language inference in this thesis) to connect the data and the user by easily using natural language, significantly promoting the possibility and efficiency of information access for many users besides data experts. All consumer-facing software will one day have a dialogue interface, and this is the next vital leap in the evolution of search engines. Such intelligent dialogue systems should understand the meaning of language grounded in various contexts and generate effective language responses in different forms for information requests and human-computer communication.

Developing these intelligent systems is challenging due to (1) limited benchmarks to drive advancements, (2) alignment mismatches between natural language and formal programs, (3) lack of trustworthiness and interpretability, (4) context dependencies in both human conversational interactions and the target programs, and (5) joint language understanding between dialog questions and NLI environments (e.g. databases and knowledge graphs). This dissertation presents several datasets, neural algorithms, and language models to address these challenges for developing deep learning technologies for conversational natural language interfaces (more specifically, NLIs to Databases or NLIDB).

First, to drive advancements towards neural-based conversational NLIs, we design and propose several complex and cross-domain NLI benchmarks, along with introducing several datasets. These datasets enable training large, deep learning models. The evaluation is done on unseen databases. (e.g., about course arrangement). Systems must generalize

well to not only new SQL queries but also to unseen database schemas to perform well on these tasks. Furthermore, in real-world applications, users often access information in a multi-turn interaction with the system by asking a sequence of related questions. The users may explicitly refer to or omit previously mentioned entities and constraints and may introduce refinements, additions, or substitutions to what has already been said. Therefore, some of them require systems to model dialog dynamics and generate natural language explanations for user verification. The full dialogue interaction with the system's responses is also important as this supports clarifying ambiguous questions, verifying returned results, and notifying users of unanswerable or unrelated questions. A robust dialogue-based NLI system that can engage with users by forming its responses has thus become an increasingly necessary component for the query process.

Moreover, this thesis presents the development of scalable algorithms designed to parse complex and sequential questions to formal programs (e.g., mapping questions to SQL queries that can execute against databases). We propose a novel neural model that utilizes type information from knowledge graphs to better understand rare entities and numbers in natural language questions. We also introduce a neural model based on syntax tree neural networks, which was the first methodology proposed for generating complex programs from language.

Finally, language modeling creates contextualized vector representations of words by training a model to predict the next word given context words, which are the basis of deep learning for NLP. Recently, pre-trained language models such as BERT and RoBERTa achieve tremendous success in many natural language processing tasks such as text understanding and reading comprehension. However, most language models are pre-trained only on free-text such as Wikipedia articles and Books. Given that language in semantic parsing is usually related to some formal representations such as logic forms and SQL queries and has to be grounded in structural environments (e.g., databases), we propose better language models for NLIs by enforcing such compositional interpolation in them. To show they

could better jointly understand dialog questions and NLI environments (e.g. databases and knowledge graphs), we show that these language models achieve new state-of-the-art results for seven representative tasks on semantic parsing, dialogue state tracking, and question answering. Also, our proposed pre-training method is much more effective than other prior work.

Learning to Map Natural Language to Executable Programs Over Databases

A Dissertation
Presented to the Faculty of the Graduate School
of
Yale University
in Candidacy for the Degree of
Doctor of Philosophy

by
Tao Yu

Dissertation Director: Dragomir R. Radev

June, 2021

Acknowledgments

This thesis would not be possible without the mentorship and support of a number of people. First of all, I would like to profusely thank my advisor, Dragomir Radev for his constant support throughout my whole Ph.D. career, and for providing me the freedom to pursue my own research directions. I first met Drago while taking his NLP course at Columbia University in 2016 and became interested in NLP because of the class. After that, he directed me to Owen Rambow and Kathleen McKeown to work on NLP research. His energy and enthusiasm are contagious, which greatly motivates me in terms of work and study.

Second, I am incredibly and forever grateful to Owen Rambow and Kathleen McKeown. In the summer after the first year of my masters at Columbia, I sent an inquiry email to Owen about the possibility to work on some NLP research projects with him, and he replied after a few weeks even though I had never worked on any related research. The project involved Kathy too. The summer turned out to be one of the best research experiences I ever had. They have led me into the NLP research area and helped me a lot with many of my initial research projects and papers. I really thank you for being patient and supportive. I would never have decided to pursue my Ph.D. in NLP without them.

I would also like to thank my other thesis committee members for taking the time to provide feedback and suggestions about my research. First, I am deeply grateful to Luke Zettlemoyer for his advice and generous support for my job search. Luke is one of the best well-known researchers in semantic parsing (for sure, also in more broadly NLP), which is my research area. He is always open to talk and tries to introduce you to anyone he knows

during conferences. I learned a lot from him about showing generosity and modesty at all times. I would also like to thank Robert Frank and Marynel Vázquez for their time to read through the thesis and for being on my thesis committee. Robert is extremely enthusiastic and knowledgeable and I always learned a lot from talking with him.

My research experience was also shaped by the research internships. I would like to thank Xi Victoria Lin, Caiming Xiong, and Richard Socher for my internship at Salesforce Research; and Ahmed Hassan Awadallah, Alex Polozov, and Christopher Meek for my internship at Microsoft Research. I spent wonderful summers with you. I especially thank Victoria for her continuous support and mentorship in several projects. She is extremely passionate about our research and always works until several paper deadlines. Also, I am grateful to Ahmed for his patient and support. I learned a lot about how to present my work effectively.

I would also like to thank my mentors in other projects including Rahul Jha, Asli Celikyilmaz at Microsoft Research; Walter Lasecki at the University of Michigan; and Smaranda Muresan at Columbia University.

During the past four years of my Ph.D. and one year of my masters, I had the great pleasure of working with many amazing colleagues and coauthors: Rui Zhang, Alexander Fabbri, Michihiro Yasunaga, Jungo Kasai, Irene Li, Yi Chern Tan, Zifan Li, Ruiqi Zhong, Kai Yang, Dongxu Wang, He Yang Er, Chien-Sheng Wu, James Ma, Youxuan Jiang, Tianze Shi, Peng Shi, Bailin Wang, Suyi Li, Qingning Yao, Shanelle Roman, Zilin Zhang, Eric Xue, Bo Pang, Vincent Zhang, Xinyi Yang, Christopher Hidey, Axinia Radeva, Mohammad Sadegh Rasooli, Noura Farra, Sungrok Shim, Tao Chen, Luyao Chen, Yuwen Zhang, David Proctor, Shreya Dixit, and Jonathan Kraft. I would like to especially thank Rui and Michi who worked very closely with me during my Ph.D. years to get me off the ground and played a significant role in teaching me to write papers. Also, I'm immensely thankful to Alex for supporting each other during the Ph.D. study after graduating from Columbia together. We spent a very nice summer working on NLP research with Owen and Kathy.

Finally, I owe all my success to my parents for their endless support and encouragement including covering my education costs in the United States. I could not imagine that I could get so far without their support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The goal of natural language interfaces (NLIs) is to assist the user in completing a certain task by performing an action or retrieving relevant information (Tur and Mori, 2011). They are often built on top of a structured ontology grounded in a knowledge base, a database, or a set of API calls. A key component of NLIs is Semantic Parsing (SP), which requires both understanding the meaning of natural language sentences and mapping them to meaningful executable queries (e.g., logical forms, SQL, SPARQL, and Python code) that can be executed against the structured ontology.

This field has become increasingly important as it allows users to query databases and control computer systems naturally and flexibly via interactive exchanges in natural language. NLIs also increase the accessibility of database systems to underprivileged members of our society, who may lack the education and skills to query a database using a traditional programmatic interface.

Consider the example in Figure 1.1 where a user queries a database about tennis rankings by asking "Which European countries have some players who won the Australian Open at least 3 times?". To convert this natural language question into the corresponding SQL query, the semantic parser has to jointly understand compositional snippets in the question and tables. This includes cell values (e.g., "European" in the question and "Europe" in the

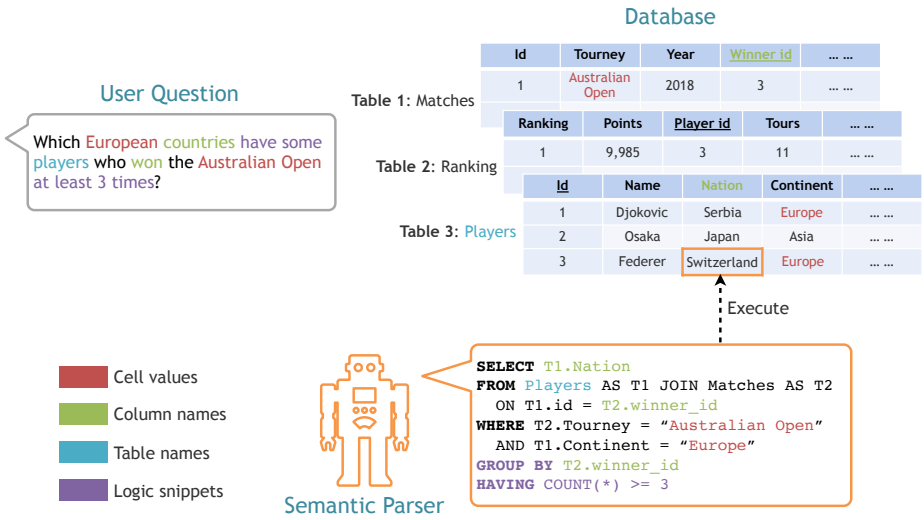

Figure 1.1: A text-to-SQL semantic parsing example. A semantic parser inputs a user question and database information and then converts the question into a corresponding SQL query.

database), column mentions (e.g., "won" in the question and "winner_id" in the database), table names (e.g., "players"), and logic compositionality (e.g., "have some ... at least 3 times" in the question and "`GROUP BY ... HAVING COUNT(*) >=3`" in the SQL output).

In real-world applications, users often access information in a multi-turn interaction with the system by asking a sequence of related questions. The users may explicitly refer to or omit previously mentioned entities and constraints and may introduce refinements, additions, or substitutions to what has already been said. This requires NLIs to process context information to synthesize correct formal programs.

Moreover, in many cases, multi-turn interaction between users and NLIs needed to clarify ambiguous questions (e.g., Q_3 and R_3 in Figure 1.2), verify returned results(e.g., R_1, R_2, and R_4), and notify users of unanswerable or unrelated questions. A robust dialogue-based NL query agent that can engage with users by forming its own responses has become an increasingly necessary component for the query process.

The key challenge in building conversational NLIs is Conversational Semantic Parsing (CSP), which is the task of converting a sequence of natural language queries to formal language, which has been extensively studied in several academic and industrial research

D_1 : Database about student dormitories containing 5 tables

Q_1 : What are the names of all the dorms? INFORM_SQL

S_1 : `SELECT dorm_name FROM dorm`

A_1 : (Result table with many entries)

R_1 : This is the list of the names of all the dorms. CONFIRM_SQL

Q_2 : Which of those dorms have a TV lounge? INFORM_SQL

S_2 : `SELECT T1.dorm_name FROM dorm AS T1 JOIN has_amenity AS T2 ON T1.dormid = T2.dormid JOIN dorm_amenity AS T3 ON T2.amenid = T3.amenid WHERE T3.amenity_name = 'TV Lounge'`

A_2 : (Result table with many entries)

R_2 : This shows the names of dorms with TV lounges. CONFIRM_SQL

Q_3 : What dorms have no study rooms as amenities? AMBIGUOUS

R_3 : Do you mean among those with TV Lounges? CLARIFY

Q_4 : Yes. AFFIRM

S_4 : `SELECT T1.dorm_name FROM dorm AS T1 JOIN has_amenity AS T2 ON T1.dormid = T2.dormid JOIN dorm_amenity AS T3 ON T2.amenid = T3.amenid WHERE T3.amenity_name = 'TV Lounge' EXCEPT SELECT T1.dorm_name FROM dorm AS T1 JOIN has_amenity AS T2 ON T1.dormid = T2.dormid JOIN dorm_amenity AS T3 ON T2.amenid = T3.amenid WHERE T3.amenity_name = 'Study Room'`

A_4 : Fawlty Towers

R_4 : Fawlty Towers is the name of the dorm that has a TV lounge but not a study room as an amenity. CONFIRM_SQL

...

Q_8 : Thanks! THANK_YOU

R_8 : You are welcome. WELCOME

Figure 1.2: A dialog from the CoSQL dataset. Gray boxes separate the user inputs (Q_i) querying the database (D_i) from the SQL queries (S_i), returned answers (A_i), and expert responses (R_i). Users send an input to the expert, who writes the corresponding SQL query (only seen by the expert) if possible and sends an answer and response description back. Dialogue acts are on the right-hand side (e.g., Q_3 is "ambiguous" and R_3 is "clarify").

settings such as dialog systems (e.g., dialog state tracking in MWoZ (Budzianowski et al., 2018)), interacting with physical agents (e.g., (Chai et al., 2018)), context-dependent semantic parsing (e.g., SParC (Yu et al., 2019b)), SQL-grounded state tracking (e.g., CoSQL (Yu et al., 2019a)), and sequential question answering (e.g., SQA (Iyyer et al., 2017)). To accom-

plish this task, a conversational NLI needs to model the relation between the unstructured language utterance and the structured ontology while representing the multi-turn dynamics of the dialog.

1.1 Background of Natural Language Interfaces

While this dissertation focuses on a more specific task of mapping natural language to executable programs over databases, most advances mentioned in the thesis can generalize to build NLIs for other structured ontologies. In this section, we provide a brief history and summary of datasets, algorithms, and language models proposed for semantic parsing.

1.1.1 Datasets

Traditional Datasets Several semantic parsing datasets with different queries have been created. The output can be in many formats, e.g., logic forms. These datasets include ATIS (Price, 1990; Dahl et al., 1994), GeoQuery (Zelle and Mooney, 1996), and JOBS (Tang and Mooney, 2001). They have been studied extensively (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Wong and Mooney, 2007; Das et al., 2010; Liang et al., 2011; Banarescu et al., 2013; Artzi and Zettlemoyer, 2013; Reddy et al., 2014; Berant and Liang, 2014; Dong and Lapata, 2016).

Recently, more semantic parsing datasets using SQL as programs have been created. Iyer et al. (2017) and Popescu et al. (2003) labeled SQL queries for ATIS and GeoQuery datasets. Other existing text-to-SQL datasets also include Restaurants (Tang and Mooney, 2001; Popescu et al., 2003), Scholar (Iyer et al., 2017), Academic (Li and Jagadish, 2014), Yelp and IMDB (Yaghmazadeh et al., 2017), and Advising (Finegan-Dollak et al., 2018). These datasets have been studied for decades in both the NLP community (Warren and Pereira, 1982; Popescu et al., 2003, 2004; Li et al., 2006; Giordani and Moschitti, 2012; Wang et al., 2017b; Iyer et al., 2017) and the Database community (Li and Jagadish, 2014;

Yaghmazadeh et al., 2017). We provide detailed statistics on these datasets in Table 2.1.

These datasets contain complex questions but are too small in terms of the number of programs for training modern data-intensive models and have only a single dataset, meaning that the same database is used for both training and testing the model. The model trained on each of them could handle some complex in-domain questions but cannot generalize to other domains.

Large-scale Datasets To enable large-scale neural modeling in building NLI systems that can generalize to new domains, Zhong et al. (2017) introduced the WikiSQL dataset, a collection of 87,673 examples of questions, queries, and database tables built from 26,521 tables. It provides train/dev/test splits such that each table is only in one split. This requires the model to generalize to not only new questions but new table schemas as well.

Although WikiSQL is large in terms of the number of programs and databases, it contains only simple SQL queries and single tables. To test a modelś real semantic parsing performance on unseen complex programs and its ability to generalize to new domains, we introduce SPIDER, the largest (about 10k) cross-domain context-independent text-to-SQL dataset, spanning 200 complex databases over 138 domains in chapter 2.

Recently, researchers have constructed several large datasets for code generation including IFTTT (Quirk et al., 2015), DJANGO (Oda et al., 2015), HEARTHSTONE (Ling et al., 2016), NL2Bash (Lin et al., 2018), and CoNaLa (Yin et al., 2018). These tasks parse natural language descriptions into a more general-purpose programming language such as Python (Allamanis et al., 2015; Ling et al., 2016; Rabinovich et al., 2017; Yin and Neubig, 2017).

Furthermore, to reduce the annotated cost, some researchers introduced datasets that are only annotated with answers as one or more table cells for studying weakly supervised semantic parsing. For example, WIKITABLEQUESTIONS (Pasupat and Liang, 2015) contains question-denotation pairs over single Wikipedia tables. The questions involve a variety of operations such as comparisons, superlatives, and aggregations, where some of them are

hard to answer by SQL queries.

Multi-turn Datasets The majority of previous work focuses on converting a single, complex question into its corresponding SQL query. However, users tend to ask a sequence of thematically related questions to learn about a particular topic or to achieve a complex goal. Only a few datasets have been constructed for mapping context-dependent questions to structured queries. Price (1990); Dahl et al. (1994) collected ATIS that includes a series of questions from users interacting with a flight database.

In chapter 3, we introduc SParC (Yu et al., 2019b), a large cross-domain semantic parsing in context dataset, consisting of 4k question sequences with 12k questions annotated with SQL queries over 200 complex databases. Similar to ATIS, SParC includes sequences of questions instead of conversational interactions. Another similar work is CONCODE (Iyer et al., 2018), which is for generating class member functions given English documentation and programmatic context.

Recently, some sequential question answering datasets have been introduced, such as QuAC (Choi et al., 2018) and CoQA (Reddy et al., 2018). They differ from SParC in that the answers are free-form text instead of SQL queries. On the other hand, Kato et al. (2004); Chai and Jin (2004); Bertomeu et al. (2006) conduct early studies of the contextual phenomena and thematic relations in database dialogue/QA systems, which we use as references when constructing SParC.

Multi-turn Datasets with Denotations Some datasets used in recovering context-dependent meaning (including SCONE (Long et al., 2016) and SequentialQA (Iyyer et al., 2017)) contain no logical form annotations but only denotation (Berant and Liang, 2014) instead. SCONE (Long et al., 2016) contains some instructions in limited domains such as chemistry experiments. The formal representations in the dataset are world states representing state changes after each instruction instead of programs or logical forms. SequentialQA (Iyyer et al., 2017) was created by asking crowd workers to decompose some complicated questions

in WikiTableQuestions (Pasupat and Liang, 2015) into sequences of inner-related simple questions. Neither of the two datasets was annotated with query labels. Thus, to make the tasks feasible, SCONE (Long et al., 2016) and SequentialQA (Iyyer et al., 2017) exclude many questions with rich semantic and contextual types. For example, (Iyyer et al., 2017) requires that the answers to the questions in SequentialQA must appear in the table, and most of them can be solved by simple SQL queries with `SELECT` and `WHERE` clauses. Such direct mapping without formal query labels becomes infeasible for complex questions. Furthermore, SequentialQA contains questions based only on a single Wikipedia table at a time.

Conversational Datasets All the corpora mentioned so far assume all user questions can be mapped into SQL queries and do not include system responses. However, in many cases, multi-turn interaction between users and NL systems is needed to clarify ambiguous questions (e.g., Q_3 and R_3 in Figure 1.2), verify returned results, and notify users of unanswerable or unrelated questions. A robust dialogue-based NL query agent that can engage with users by forming its own responses has become an increasingly necessary component for the query process. Therefore, we introduce CoSQL dataset in chapter 4. It has 30k+ turns plus 10k+ annotated SQL queries, obtained from a Wizard-of-Oz (WOZ) collection of 3k dialogues querying 200 complex databases spanning 138 domains.

Such systems also have already been studied under task-oriented dialogue settings by virtue of continuous effort of corpus creation (Seneff and Polifroni, 2000; Walker et al., 2002; Raux et al., 2005; Mrksic et al., 2015; Asri et al., 2017; Budzianowski et al., 2018) and modelling innovation (Artzi and Zettlemoyer, 2011; Henderson et al., 2013; Lee and Dernoncourt, 2016; hao Su et al., 2016; Dhingra et al., 2016; Li et al., 2016). The goal of these systems is to help users accomplish a specific task, such as flight or hotel booking or transportation planning. However, to achieve these goals, task-oriented dialogue systems rely on pre-defined slots and values for request processing (which can be represented using

simple SQL queries consisting of `SELECT` and `WHERE` clauses). Thus, these systems only operate on a small number of domains and have difficulty capturing the diverse semantics of practical user questions.

1.1.2 Algorithms

Early Systems Building natural language interfaces has been studied for decades (Warren and Pereira, 1982; Popescu et al., 2003, 2004; Li et al., 2006; Giordani and Moschitti, 2012; Wang et al., 2017b). Early studies in this field focus more on natural language interfaces to relational databases (Li and Jagadish, 2014; Pasupat and Liang, 2015; Yin et al., 2016; Zhong et al., 2017; Yaghmazadeh et al., 2017; Xu et al., 2017; Wang et al., 2017a). It requires a system that can understand natural language questions and generate corresponding SQL queries. The methods proposed in the database community (Li and Jagadish, 2014; Yaghmazadeh et al., 2017) tend to involve hand feature engineering and user interactions with the systems.

Neural Systems Inspired by neural machine translation Sutskever et al. (2014), Dong and Lapata (2016) introduce a sequence-to-sequence approach to converting text to logical forms. Most previous work focuses on specific table schemas, which means they use a single database in both train and test. Thus, they don't generalize to new databases. Zhong et al. (2017) publish the WikiSQL dataset and propose a sequence-to-sequence model with reinforcement learning to generate SQL queries. Xu et al. (2017) further improves the results on the WikiSQL task by using a SQL-sketch-based approach employing a sequence-to-set model.

In chapter 5, we present Yu et al. (2018a), which improves upon SQLNet by proposing a different training procedure and utilizing types extracted from either knowledge graph or table content to help model better understand entities and numbers in the question. Dong and Lapata (2018) propose a coarse-to-fine model which achieves the new state-of-the-art

performances on several datasets including WikiSQL. Their model first generates a sketch of the target program. Then the model fills in missing details in the sketch.

To generate more complex and syntactically correct target programs, we propose syntax tree neural networks in chapter 6 which calls each grammar component recursively to generate a SQL syntax tree guided by a SQL specific grammar. Similarly, Yin and Neubig (2017); Rabinovich et al. (2017) exploit syntax information for code generation tasks. Yin and Neubig (2017) introduce a neural model that transduces a natural language statement into an abstract syntax tree (AST). Rabinovich et al. (2017) propose abstract syntax networks that use a collection of recursive modules for decoding.

Conversational Systems In a real-world application, users often access information in a multi-turn interaction with the system by asking a sequence of related questions. As the interaction proceeds, the user often refers to the relevant mentions in the history or omits previously conveyed information assuming it is known to the system.

Therefore, in the context-dependent scenario, the contextual history is crucial to understand the follow-up questions from users, and the system often needs to reproduce partial sequences generated in previous turns. On ATIS, Miller et al. (1996) maps utterances to semantic frames which are then mapped to SQL queries; Zettlemoyer and Collins (2009) starts with context-independent Combinatory Categorial Grammar (CCG) parsing and then resolves references to generate lambda-calculus logical forms for sequences of sentences. Recently, Suhr et al. (2018) generate ATIS SQL queries from interactions by incorporating history with an interaction-level encoder and copying segments of previously generated queries.

To tackle this challenge, we proposed an editing-based model for our cross-domain multi-turn text-to-SQL task (Zhang et al., 2019d). Based on the observation that adjacent natural language questions are often linguistically dependent and their corresponding SQL queries tend to overlap, we utilized the interaction history by editing the previous predicted

query to improve the generation quality. This editing mechanism views SQL as sequences and reuses generation results at the token level in a simple manner. It is flexible to change individual tokens and robust to error propagation.

Evaluation Evaluation in semantic parsing has been a long-standing problem. Previously, simple string match and execution accuracy have been widely adopted, but they often produce false positive or false negative examples. In chapter 2, to evaluate SPIDER, we use the exact set match metric to compute the accuracy between gold and predicted SQL answers. Instead of simply employing string match, we decompose predicted queries into different SQL clauses such as `SELECT`, `WHERE`, `GROUP BY`, and `ORDER BY` and compute scores for each clause using set matching separately.

The programming language research community developed formal tools to reliably reason about query equivalence for a restricted set of query types. They lift SQL queries into other semantic representations such as K-relations Green et al. (2007), UniNomial Chu et al. (2017) and U-semiring Chu et al. (2018); then they search for an (in)equivalence proof. However, these representations cannot express sort operations and float comparisons, and hence do not support the full range of operations that Text-to-SQL models can use.

Recently, we proposed the test suite accuracy to approximate the semantic accuracy for Text-to-SQL models (Zhong et al., 2020a). Our method distills a small test suite of databases that achieves high code coverage for the gold query from a large number of randomly generated databases. At evaluation time, our approach computes the denotation accuracy of the predicted queries on the distilled test suite, hence calculating a tight upper-bound for semantic accuracy efficiently. We use our proposed method to evaluate 21 models submitted to the Spider leaderboard and manually verify that our method is always correct on 100 examples. In contrast, the current Spider metric leads to a 2.5% false negative rate on average and 8.1% in the worst case, indicating that test suite accuracy is needed.

1.1.3 Language Model Pre-Training

Static Word Embeddings Word embeddings are used to represent words in neural models. The most famous word embedding is Word2Vec (Mikolov et al., 2013), which allowed for creating dense vector representations of words by training a model to predict relevant words based on context. However, Word2Vec is a shallow network; it consists only of a single layer, and the resulting embeddings are input to a larger, task-specific neural network.

History of Language Model Pre-Training Language modeling is to train an entire network on a task where the model aims to predict the next word given context words and initializing task-specific network layers on top of the network, vastly improved performance. Training on a task before fine-tuning on a final task is called pretraining. The intuition is that different layers of the deep network capture different language phenomena, such as syntax and semantics. Furthermore, pretraining on the language modeling task teaches the model some notions of language, as predicting the next word shows some level of language understanding. When fine-tuning the model for a down-stream task, the model does not need to learn these properties from scratch.

Ramachandran et al. (2017) first applied pretraining networks for NLP. However, pretraining did not gain steam until the introduction of ULMFit (Howard and Ruder, 2018), ELMo (Peters et al., 2018), and GPT (Radford et al., 2018) models, which pretrained using the task of language modeling. Currently, the most widely used pretrained model is BERT (Bi-directional Encoder Representations from Transformers) (Devlin et al., 2019), which consists of a bi-directional encoder Transformer model with the base version containing 110 million parameters and the large version containing 340 million parameters. BERT is notably bi-direction compared to previous work; during pretraining, the model predicts words given context from before and after the word after masking a given percentage of the input. Fine-tuning BERT with task-specific neural network layers achieved state-of-the-art performance on a wide range of natural language understanding tasks.

Pre-Training for Semantic Parsing Similar to many other natural language tasks, recent work in semantic parsing has significantly benefited from advances in language model pre-training. However, existing general-purpose pre-trained language models, e.g. BERT (Devlin et al., 2019), are pre-trained on free-form text data using language model objectives. This limits their ability in modeling the structural context or the multi-turn dynamics of the dialogs. This presents an opportunity to improve pre-trained LMs to specifically address these limitations for semantic parsing tasks. Recent work has demonstrated the benefits of adapting pre-trained LMs to specific domains (Gururangan et al., 2020) or tasks (Zhang et al., 2019b) via a second phase of pre-training, e.g., summarization (Zhang et al., 2019b), knowledge inference (Sun et al., 2019b; Liu et al., 2019a), etc. This triggered an exciting line of research work under the themes of (1) cross-modal pre-training that involves text (Lu et al., 2019; Peters et al., 2019; Yin et al., 2020; Herzig et al., 2020a) and (2) pre-training architectures and objectives catering subsets of NLP tasks (Lewis et al., 2020b,a; Guu et al., 2020).

To close this gap in semantic parsing, several recent works (including TaBERT (Yin et al., 2020), TAPAS (Herzig et al., 2020a), and GRAPPA (Yu et al., 2020) presented in chapter 7) seek to learn contextual representations jointly from structured tabular data and unstructured natural language sentences, with objectives oriented towards table semantic parsing.

Pre-Training for Conversational Semantic Parsing Adapting pre-trained LMs for open-domain chit-chat models focuses on improving response generation on open-ended dialogues by adding a pre-training step on open-domain conversations data, such as Reddit data (Zhang et al., 2020a; Henderson et al., 2019). For example, open-domain dialogue language models such as DialoGPT (Zhang et al., 2020a) and ConveRT (Henderson et al., 2019) are pre-trained on the Reddit data and applied to dialog response generation and retrieval tasks. Moreover, Wu et al. (2020) introduced ToD-BERT, a pre-trained language model combining

9 high-quality human-human task-oriented dialogue datasets to conduct language model and response selection pre-training. However, they use language modeling training objectives over free-form text and therefore have limited ability to represent structural data. In chapter 8, we address the challenge of conversational semantic parsing tasks by learning pretrained representation for both the multi-turn dynamics of the dialog and the relation between the unstructured language utterance and the structured ontology.

1.2 Contributions

In this dissertation, we aim to propose datasets and methodologies in deep neural networks for improving semantic parsing. We summarize our contributions along the following axes:

1.2.1 Datasets

Prior work in semantic parsing developed rule-based, well-designed systems that are limited to a single domain or simple queries. Such designed systems only work for a specific database or handle simple user queries. To advance the state-of-the-art in this field and make semantic parsers scalable and be able to handle complex queries, I designed and proposed the first complex and cross-domain semantic parsing text-to-SQL task, along with introducing the Spider[1] dataset, the largest ($\sim$10k) cross-domain context-independent text-to-SQL dataset available in the field, spanning 200 complex databases over 138 domains. The large number of domains provide rich contextual phenomena and thematic relations between the questions, which general-purpose natural language interfaces to databases have to address. Moreover, the evaluation is done on unseen databases (e.g., about course arrangement).

In real-world applications, users often access information in a multi-turn interaction with the system by asking a sequence of related questions. To advance the state-of-the-art in

1. https://yale-lily.github.io/spider

this field, I proposed the multi-turn text-to-SQL SParC [2] task for cross-domain Semantic Parsing in Context which contains 4,298 unique multi-turn question sequences, comprised of 12k+ questions annotated with SQL queries. Moreover, the full dialogue interaction with the system's responses is also important as this supports clarifying ambiguous questions (e.g., R_3 in Figure 1.2), verifying returned results (e.g., R_1, R_2, and R_4), and notifying users of unanswerable or unrelated questions. A robust dialogue-based NLI system that can engage with users by forming its own responses has thus become an increasingly necessary component for the query process. To enable methodological advances in this field, I proposed the CoSQL [3] task (Figure 1.2) for building database-querying dialogue systems, which consists of 30k+ turns with 10k+ annotated SQL queries obtained from a Wizard-of-Oz collection of 3k dialogues. Both of these tasks are built on top of Spider. The goal of the two projects is to establish a task in service of building a conversational NLI system that possesses the ability to (1) detect questions answerable by SQL (e.g., our TriageSQL benchmark (Zhang et al., 2020b)), (2) ground user questions into executable SQL queries if possible (Zhang et al., 2019d), (3) return results to the user in a way that is easily understood and verifiable, and (4) handle unanswerable questions.

1.2.2 Algorithms

In chapter 5, we propose TypeSQL (Yu et al., 2018a), a novel neural model that utilizes type information from knowledge graphs to better understand rare entities and numbers in natural language questions, which outperformed the prior state-of-the-art by 5.5% on the WikiSQL task in a much shorter training time. We also show that accessing the content of databases can significantly improve the performance when users' queries are not well-formed. In addition, chapter 6 presents SyntaxSQL (Yu et al., 2018b) based on syntax tree neural networks, which was the first methodology proposed for solving the challenging Spider

2. https://yale-lily.github.io/sparc

3. https://yale-lily.github.io/cosql

task. It decomposes the SQL decoding process into 9 modules to handle the prediction of different SQL components such as keywords, operators, and columns. It also makes use of a SQL-specific grammar to structure the decoding process which determines which module is invoked at each recursive decoding step. This model uses a SQL-specific syntax tree-based decoder with SQL generation path history and table-aware column attention encoders and can solve nested queries on new, unseen databases. Experimental results showed that our method can handle a significantly greater number of complex SQL examples than prior work, outperforming the previous state-of-the-art model by 7.3% in exact matching accuracy.

1.2.3 Language Model Pre-Training

Language in semantic parsing is usually related to some formal representations such as logic forms and SQL queries. Consider the question in Figure 1.1 again, it consists of logic units such as cell values, column mentions, table names, and logic snippets. Small logic units make up larger ones to convey meaning, and the semantics of logic units depend on each other. Therefore, modeling logic units and their dependency relationships with the corresponding environment is fundamental for natural language understanding in semantic parsing. Recently, pre-trained language models (LMs) such as BERT and RoBERTa achieve tremendous success in many natural language processing tasks such as text understanding and reading comprehension. However, most LMs are pre-trained only on free-text such as Wikipedia articles and Books. In chpater 7 and 8, we propose effective language model pre-training approaches, GraPPa (Yu et al., 2020) and SCoRe (Yu et al., 2021), for semantic parsing that enforces compositional interpolation in the joint representation of textual and tabular data.

We demonstrate the broad applicability and effectiveness of the proposed pre-trained language models on eight popular fully supervised and weakly supervised, single-turn, and conversational semantic parsing benchmarks, they can significantly improve the performance over all these tasks. Our GraPPa and SCoRe language models achieve new state-of-the-art

results for seven representative tasks on semantic parsing, dialogue state tracking, and question answering. Finally, our proposed pre-training method is much more effective than other prior work. Compared with several days on over 100 GPUs for TaBERT, our pre-training procedure requires less than 10 hours on eight GPUs, which saves substantial time and computational energy.

1.3 Outline

The chapters in this thesis are based on the following publications:

- **Chapter 2: Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task**. In the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2018.

- **Chapter 3: SParC: Cross-Domain Semantic Parsing in Context**. In the 57th Annual Meeting of the Association for Computational Linguistics (ACL), 2019.

- **Chapter 4: CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Natural Language Interfaces to Databases**. In the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2019.

- **Chapter 5: TypeSQL: Knowledge-based Type-Aware Neural Text-to-SQL Generation**. In the 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT), 2018.

- **Chapter 6: SyntaxSQLNet: Syntax Tree Networks for Complex and Cross-Domain Text-to-SQL Task**. In the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2018.

- **Chapter 7: GraPPa: Grammar-Augmented Pre-Training for Table Semantic Parsing**. In the 9th International Conference on Learning Representations (ICLR),

2021.

- **Chapter 8: SCoRe: Pre-Training for Context Representation in Conversational Semantic Parsing**. In the 9th International Conference on Learning Representations (ICLR), 2021.

Part I

Datasets

Chapter 2

Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task

In this chapter, we present *Spider*, a large-scale, complex, and cross-domain semantic parsing and text-to-SQL dataset annotated by 11 college students. It consists of 10,181 questions and 5,693 unique complex SQL queries on 200 databases with multiple tables, covering 138 different domains. We define a new complex and cross-domain semantic parsing and text-to-SQL task where different complex SQL queries and databases appear in train and test sets. In this way, the task requires the model to generalize well to both new SQL queries and new database schemas. Spider is distinct from most of the previous semantic parsing tasks because they all use a single database and the exact same programs in the train set and the test set. We experiment with various state-of-the-art models and the best model achieves only 12.4% exact matching accuracy on a database split setting. This shows that Spider presents a strong challenge for future research. Our dataset and task are publicly available at `https://yale-lily.github.io/spider`.

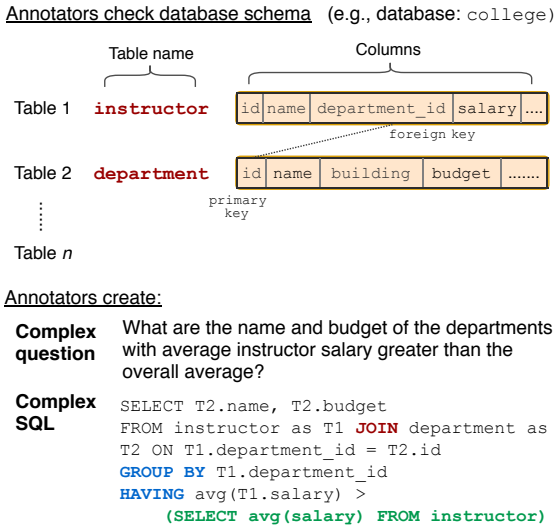

Figure 2.1: Our corpus annotates complex questions and SQLs. The example contains joining of multiple tables, a `GROUP BY` component, and a nested query.

2.1 Introduction

Recently, some state-of-the-art methods with Seq2Seq architectures are able to achieve over 80% exact matching accuracy even on some complex benchmarks such as ATIS and GeoQuery. These models seem to have already solved most problems in this field.

However, previous tasks in this field have a simple but problematic task definition because most of these results are predicted by semantic "matching" rather than semantic parsing. Existing datasets for SP have two shortcomings. First, those that have complex programs (Zelle and Mooney, 1996; Li and Jagadish, 2014; Yaghmazadeh et al., 2017; Iyer et al., 2017) are too small in terms of the number of programs for training modern data-intensive models and have only a single dataset, meaning that the same database is used for both training and testing the model. More importantly, the number of logic forms or SQL labels is small and each program has about 4-10 paraphrases of natural language problems to expand the size of the dataset. Therefore, the exact same target programs appear in both the train and test sets. The models can achieve decent performances even on very complex programs by memorizing the patterns of question and program pairs during training and decoding the programs exactly the same way as it saw in the training set during

testing. Finegan-Dollak et al. (2018) split the dataset by programs so that no two identical programs would be in both the train and test sets. They show that the models built on this question-splitting data setting fail to generalize to unseen programs. Second, existing datasets that are large in terms of the number of programs and databases such as WikiSQL (Zhong et al., 2017) contain only simple SQL queries and single tables. In order to test a model's real semantic parsing performance on unseen complex programs and its ability to generalize to new domains, an SP dataset that includes a large amount of complex programs and databases with multiple tables is a must.

To address the need for a large and high-quality dataset for a new complex and cross-domain semantic parsing task, we introduce *Spider*, which consists of 200 databases with multiple tables, 10,181 questions, and 5,693 corresponding complex SQL queries, all written by 11 college students spending a total of 1,000 man-hours. As Figure 2.1 illustrates, given a database with multiple tables including foreign keys, our corpus creates and annotates complex questions and SQL queries including different SQL clauses such as joining and nested query. In order to generate the SQL query given the input question, models need to understand both the natural language question and relationships between tables and columns in the database schema.

In addition, we also propose a new task for the text-to-SQL problem. Since Spider contains 200 databases with foreign keys, we can split the dataset with complex SQL queries in a way that no database overlaps in train and test, which overcomes the two shortcomings of prior datasets, and defines a new semantic parsing task in which the model needs to generalize not only to new programs but also to new databases. Models have to take questions and database schemas as inputs and predict unseen queries on new databases.

To assess the task difficulty, we experiment with several state-of-the-art semantic parsing models. All of them struggle with this task. The best model achieves only 12.4% exact matching accuracy in the database split setting. This suggests that there is a large room for improvement.

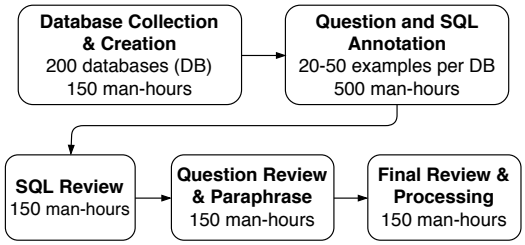

Figure 2.2: The annotation process of our Spider corpus.

2.2 Related Work

Most of the previous work using traditional datasets (including include Restaurants (Tang and Mooney, 2001; Popescu et al., 2003), Scholar (Iyer et al., 2017), Academic (Li and Jagadish, 2014), Yelp and IMDB (Yaghmazadeh et al., 2017), and Advising (Finegan-Dollak et al., 2018).) train their models without schemas as inputs because they use a single database for both training and testing. Thus, they do not need to generalize to new domains. Most importantly, these datasets have a limited number of labeled logic forms or SQL queries. In order to expand the size of these datasets and apply neural network approaches, each logic form or SQL query has about 4-10 paraphrases for the natural language input.

We also want the model to generalize not only to unseen queries but also to unseen databases. Zhong et al. (2017) published the WikiSQL dataset. In their problem definition, the databases in the test set do not appear in the train or development sets. Also, the task needs to take different table schemas as inputs. Therefore, the model has to generalize to new databases. However, in order to generate 80654 questions and SQL pairs for 24241 databases, Zhong et al. (2017) made simplified assumptions about the SQL queries and databases. Their SQL labels only cover single `SELECT` column and aggregation, and `WHERE` conditions. Moreover, all the databases only contain single tables. No `JOIN`, `GROUP BY`, and `ORDER BY`, etc. are included.

2.3 Corpus Construction

All questions and SQL queries were written and reviewed by 11 computer science students. Some of them were native English speakers. As illustrated in Figure 2.2, we develop our dataset in five steps, spending around 1,000 hours of human labor in total: §2.3.1 Database Collection and Creation, §2.3.2 Question and SQL Annotation, §2.3.3 SQL Review, §2.3.4 Question Review and Paraphrase, §2.3.5 Final Question and SQL Review.

2.3.1 Database Collection and Creation

Collecting databases with complex schemas is hard. Although relational databases are widely used in industry and academia, most of them are not publicly available. Only a few databases with multiple tables are easily accessible online.

Our 200 databases covering 138 different domains are collected from three resources. First, we collected about 70 complex databases from different college database courses, SQL tutorial websites, online csv files, and textbook examples. Second, we collected about 40 databases from the DatabaseAnswers[4] where contains over 1,000 data models across different domains. These data models contain only database schemas. We converted them into SQLite, populated them using an online database population tool[5], and then manually corrected some important fields so that the table contents looked natural. Finally, we created the remaining 90 databases based on WikiSQL. To ensure the domain diversity, we select about 500 tables in about 90 different domains to create these 90 databases. To create each database, we chose several related tables from WikiSQL dev or test splits, and then created a relational database schema with foreign keys based on the tables we selected. We had to create some intersection tables in order to link several tables together. For most other cases, we did not need to populate these databases since tables in WikiSQL are from Wikipedia,

4. `http://www.databaseanswers.org/`

5. `http://filldb.info/`

which already had real world data stored.

We manually corrected some database schemas if they had some column names that did not make sense or missed some foreign keys. For table and column names, it is common to use abbreviations in databases. For example, 'student_id' might be represented by 'stu_id'. For our task definition, we manually changed each column name back to regular words so that the system only handled semantic parsing issues.

2.3.2 Question and SQL Annotation

For each database, we ask eight computer science students proficient in SQL to create 20-50 natural questions and their SQL labels. To make our questions diverse, natural, and reflective of how humans actually use databases, we did not use any template or script to generate question and SQL queries. Our annotation procedure ensures the following three aspects.

A) SQL pattern coverage. We ensure that our corpus contains enough examples for all common SQL patterns. For each database, we ask annotators to write SQL queries that cover all the following SQL components: `SELECT` with multiple columns and aggregations, `WHERE`, `GROUP BY`, `HAVING`, `ORDER BY`, `LIMIT`, `JOIN`, `INTERSECT`, `EXCEPT`, `UNION`, `NOT IN`, `OR`, `AND`, `EXISTS`, `LIKE` as well as nested queries. This also increases the diversity and complexity of corresponding natural language since language has to reflect the comprehensive meaning of SQL query. The annotators made sure that each table in the database appears in at least one query.

B) SQL consistency. Some questions have multiple acceptable SQL queries with the same result. However, giving totally different SQL labels to similar questions can hinder the training of semantic parsing models. To avoid this issue, we designed the annotation protocol so that all annotators choose the same SQL query pattern if multiple equivalent queries are possible.

C) Question clarity. We did not create questions that are (1) vague or too ambiguous, or (2) require knowledge outside the database to answer.

First, ambiguous questions refer to the questions that do not have enough clues to infer which columns to return and which conditions to consider. For example, we would not ask "What is the most popular class at University X?" because the definition of "popular" is not clear: it could mean the rating of the class or the number of students taking the course. Instead, we choose to ask "What is the name of the class which the largest number of students are taking at University X?". Here, "popular" refers to the size of student enrollment. Thus, the "student_enrollment" column can be used in condition to answer this question. We recognize that ambiguous questions appear in real-world natural language database interfaces.

We agree that future work needs to address this issue by having multi-turn interactions between the system and users for clarification. However, our main aim here is to develop a corpus to tackle the problem of handling complex queries and generalizing across databases without multi-turn interactions required, which no existing semantic parsing datasets could do. Moreover, the low performances of current state-of-the-art models already show that our task is challenging enough, without ambiguous questions. In addition, questions are required to contain the specific information to return. Otherwise, we don't know if class id is also acceptable in the previous case. Most of the questions in the existing semantic parsing datasets are ambiguous or miss too much information. For example, the SQL label for "show me flights from Seattle to Boston next Monday." in the ATIS dataset by default selects "flight_id" column and has "year = 1993 and month_number = 2 and day_number = 8" in its where condition. This is not a serious problem if we use one single dataset because we have enough data domain specific examples to know which columns are the default. However, it would be a serious problem in cross domain tasks since the default return values differ cross domain and people.

Second, humans sometimes ask questions that require common sense knowledge outside

the given database. For instance, when people ask "Display the employee id for the employees who report to John", the correct SQL is

```
SELECT employee_id
FROM employees
WHERE manager_id = (
SELECT employee_id
FROM employees
WHERE first_name = 'John')
```

which requires the common knowledge that "X reports to Y" corresponds to an "employee-manager" relation. we do not include such questions and leave them as a future research direction.

Annotation tools We open each database on a web-based interface powered by the sqlite_web[6] tool. It allows the annotators to see the schema and content of each table, execute SQL queries, and check the returned results. This tool was extremely helpful for the annotators to write executable SQL queries that reflect the true meaning of the given questions and return correct answers.

Dataset	# Q	# SQL	# DB	# Domain	# Table /DB	ORDER BY	GROUP BY	NESTED	HAVING
ATIS	5,280	947	1	1	32	0	5	315	0
GeoQuery	877	247	1	1	6	20	46	167	9
Scholar	817	193	1	1	7	75	100	7	20
Academic	196	185	1	1	15	23	40	7	18
IMDB	131	89	1	1	16	10	6	1	0
Yelp	128	110	1	1	7	18	21	0	4
Advising	3,898	208	1	1	10	15	9	22	0
Restaurants	378	378	1	1	3	0	0	4	0
WikiSQL	80,654	77,840	26,521	-	1	0	0	0	0
Spider	10,181	5,693	200	138	5.1	1335	1491	844	388

Table 2.1: Comparisons of text-to-SQL datasets. **Spider** is the *only one* text-to-SQL dataset that contains both databases with multiple tables in different domains and complex SQL queries. It was designed to test the ability of a system to generalize to not only new SQL queries and database schemas but also new domains.

6. https://github.com/coleifer/sqlite-web

2.3.3 SQL Review

Once the database is labeled with question-query pairs, we ask a different annotator to check if the questions are clear and contain enough information to answer the query. For a question with multiple possible SQL translations, the reviewers double check whether the SQL label is correctly chosen under our protocol. Finally, the reviewers check if all the SQL labels in the current database cover all the common SQL clauses.

2.3.4 Question Review and Paraphrase

After SQL labels are reviewed, native English speakers review and correct each question. They first check if the question is grammatically correct and natural. Next, they make sure that the question reflects the meaning of its corresponding SQL label. Finally, to improve the diversity in questions, we ask annotators to add a paraphrased version to some questions.

2.3.5 Final Review

Finally, we ask the most experienced annotator to conduct the final question and SQL review. This annotator makes the final decision if multiple reviewers are not sure about some annotation issues. Also, we run a script to execute and parse all SQL labels to make sure they are correct.

2.4 Dataset Statistics and Comparison

We summarize the statistics of Spider and other text-to-SQL datasets in Table 2.1. Compared with other datasets, Spider contains databases with multiple tables and contains SQL queries including many complex SQL components. For example, Spider contains about twice more nested queries and 10 times more `ORDER BY (LIMIT)` and `GROUP BY (HAVING)` components than the total of previous text-to-SQL datasets. Spider has 200 distinct databases

covering 138 different domains such as college, club, TV show, government, etc. Most domains have one database, thus containing 20-50 questions, and a few domains such as flight information have multiple databases with more than 100 questions in total. On average, each database in Spider has 27.6 columns and 8.8 foreign keys. The average question length and SQL length are about 13 and 21 respectively. Our task uses different databases for training and testing, evaluating the cross-domain performance. Therefore, Spider is the *only one* text-to-SQL dataset that contains both databases with multiple tables in different domains and complex SQL queries It tests the ability of a system to generalize to not only new SQL queries and database schemas but also new domains.

2.5 Task Definition

On top of the proposed dataset, we define a text-to-SQL task that is more realistic than prior work. Unlike most of the previous semantic parsing or text-to-SQL tasks, models will be tested on *both different complex SQL queries and different complex databases in different domains* in our task. It aims to ensure that models can only make the correct prediction when they truly understand the semantic meaning of the questions, rather than just memorization. Also, because our databases contain different domains, our corpus tests model's ability to generalize to new databases. In this way, model performance on this task can reflect the real semantic parsing ability.

In order to make the task feasible and to focus on the more fundamental part of semantic parsing, we make the following assumptions:

- In our initial task setting, we did not evaluate model performance on generating column values in the SQL query. Predicting correct SQL structures and columns is more realistic and critical at this stage based on the low performances of various state-of-the-art models on our task at the time. In the current task setting, we also evaluate this because of the tremendous performance improvement from using language models. In a real world

situation, people need to double check what condition values are and finalize them after multiple times. It is unrealistic to predict condition values without interacting with users. In reality, most people know what values to ask but do not know the SQL logic. A more reasonable way is to ask users to use an interface searching the values, then ask more specific questions. Also, other previous work with value prediction uses one single database in both train and test which makes it vulnerable to overfitting. However, SQL queries must include values in order to execute them. For value prediction in our task, a list of gold values for each question is given. Models need to fill them into the right slots in their predicted SQL.

- As mentioned in the previous sections, we exclude some queries that require outside knowledge such as common sense inference and math calculation. For example, imagine a table with birth and death year columns. To answer the questions like "How long is X's life length?", we use `SELECT death_year - birth_year`. Even though this example is easy for humans, it requires some common knowledge of the life length definition and the use of a math operation, which is not the focus of our dataset.

- We assume all table and column names in the database are clear and self-contained. For example, some databases use database specific short-cut names for table and column names such as "stu_id", which we manually converted to "student id" in our corpus.

2.6 Evaluation Metrics

Our evaluation metrics include Component Matching, Exact Matching, and Execution Accuracy. In addition, we measure the system's accuracy as a function of the difficulty of a query. Since our task definition does not predict value string, our evaluation metrics do not take value strings into account.

We will release the official evaluation script along with our corpus so that the research community can share the same evaluation platform.

Component Matching To conduct a detailed analysis of model performance, we measure the average exact match between the prediction and ground truth on different SQL components. For each of the following components:

- `SELECT` • `WHERE` • `GROUP BY`
- `ORDER BY` • `KEYWORDS` (including all SQL keywords without column names and operators)

we decompose each component in the prediction and the ground truth as bags of several sub-components, and check whether or not these two sets of components match exactly. To evaluate each `SELECT` component, for example, consider `SELECT avg(col1), max(col2), min(col1)`, we first parse and decompose into a set `(avg, min, col1), (max, col2)`, and see if the gold and predicted sets are the same. Previous work directly compared decoded SQL with gold SQL. However, some SQL components do not have order constraints. In our evaluation, we treat each component as a set so that for example, `SELECT avg(col1), min(col1), max(col2)` and `SELECT avg(col1), max(col2), min(col1)` would be treated as the same query. To report a model's overall performance on each component, we compute F1 score on exact set matching.

Exact Matching We measure whether the predicted query as a whole is equivalent to the gold query. We first evaluate the SQL clauses as described in the last section. The predicted query is correct only if all of the components are correct. Because we conduct a set comparison in each clause, this exact matching metric can handle the "ordering issue" (Xu et al., 2017).

Execution Accuracy We exclude value prediction in Component and Exact Matching evaluations and do not provide Execution Accuracy in the current version. However, it is also important to note that Execution Accuracy can create false positive evaluation when a predicted SQL returns the same result (for example, 'NULL') as the gold SQL while they are semantically different. So we can use both to complement each other.

Finally, our evaluation also considers multiple acceptable keys if `JOIN` and `GROUP` are in the query. For example, suppose "stu_id" in one table refers to "stu_id" in another table, `GROUP BY` either is acceptable.

SQL Hardness Criteria To better understand the model performance on different queries, we divide SQL queries into 4 levels: easy, medium, hard, extra hard. We define the difficulty based on the number of SQL components, selections, and conditions, so that queries that contain more SQL keywords (`GROUP BY`, `ORDER BY`, `INTERSECT`, nested subqueries, column selections and aggregators, etc) are considered to be harder. For example, a query is considered as hard if it includes more than two `SELECT` columns, more than two `WHERE` conditions, and `GROUP BY` two columns, or contains `EXCEPT` or nested queries. A SQL with more additions on top of that is considered as extra hard. Figure 2.3 shows examples of SQL queries in 4 hardness levels.

	Test					Dev
	Easy	Medium	Hard	Extra Hard	All	All
Example Split						
Seq2Seq	22.0	7.8	5.5	1.3	9.4	10.3
Seq2Seq+Attention Dong and Lapata (2016)	32.3	15.6	10.3	2.3	15.9	16.0
Seq2Seq+Copying	29.3	13.1	8.8	3.0	14.1	15.3
SQLNet Xu et al. (2017)	34.1	19.6	11.7	3.3	18.3	18.4
TypeSQL Yu et al. (2018a)	47.5	38.4	24.1	14.4	33.0	34.4
Database Split						
Seq2Seq	11.9	1.9	1.3	0.5	3.7	1.9
Seq2Seq+Attention Dong and Lapata (2016)	14.9	2.5	2.0	1.1	4.8	1.8
Seq2Seq+Copying	15.4	3.4	2.0	1.1	5.3	4.1
SQLNet Xu et al. (2017)	26.2	12.6	6.6	1.3	12.4	10.9
TypeSQL Yu et al. (2018a)	19.6	7.6	3.8	0.8	8.2	8.0

Table 2.2: Accuracy of Exact Matching on SQL queries with different hardness levels.

2.7 Methods

In order to analyze the difficulty and demonstrate the purpose of our corpus, we experiment with several state-of-the-art semantic parsing models. As our dataset is fundamentally

Easy

What is the number of cars with more than 4 cylinders?

```
SELECT COUNT(*)
FROM cars_data
WHERE cylinders > 4
```

Meidum

For each stadium, how many concerts are there?

```
SELECT T2.name, COUNT(*)
FROM concert AS T1 JOIN stadium AS T2
ON T1.stadium_id = T2.stadium_id
GROUP BY T1.stadium_id
```

Hard

Which countries in Europe have at least 3 car manufacturers?

```
SELECT T1.country_name
FROM countries AS T1 JOIN continents
AS T2 ON T1.continent = T2.cont_id
JOIN car_makers AS T3 ON
T1.country_id = T3.country
WHERE T2.continent = 'Europe'
GROUP BY T1.country_name
HAVING COUNT(*) >= 3
```

Extra Hard

What is the average life expectancy in the countries where English is not the official language?

```
SELECT AVG(life_expectancy)
FROM country
WHERE name NOT IN
   (SELECT T1.name
    FROM country AS T1 JOIN
    country_language AS T2
    ON T1.code = T2.country_code
    WHERE T2.language = "English"
      AND T2.is_official = "T")
```

Figure 2.3: SQL query examples in 4 hardness levels.

different from the prior datasets such as Geoquery and WikiSQL, we adapted these models to our task as follows. We created a 'big' column list by concatenating columns in all tables of the database together as an input to all models. Also, for each model, we limit the column selection space for each question example to all column of the database which the question is asking instead of all column names in the whole corpus.

Method	SELECT	WHERE	GROUP BY	ORDER BY	KEYWORDS
Example Split					
Seq2Seq	23.3	4.9	15.3	9.2	17.9
Seq2Seq+Attention	31.1	9.1	28.2	20.8	21.4
Seq2Seq+Copying	28.2	8.3	25.5	21.3	19.0
SQLNet	59.8	32.9	35.9	65.5	76.1
TypeSQL	77.3	52.4	47.0	67.5	78.4
Database Split					
Seq2Seq	13.0	1.5	3.3	5.3	8.7
Seq2Seq+Attention	13.6	3.1	3.6	9.9	9.9
Seq2Seq+Copying	12.0	3.1	5.3	5.8	7.3
SQLNet	44.5	19.8	29.5	48.8	64.0
TypeSQL	36.4	16.0	17.2	47.7	66.2

Table 2.3: F1 scores of Component Matching on all SQL queries on Test set.

Seq2Seq Inspired by neural machine translation (Sutskever et al., 2014), we first apply a basic sequence-to-sequence model, **Seq2Seq**. Then, we also explore **Seq2Seq+Attention** from (Dong and Lapata, 2016) by adding an attention mechanism (Bahdanau et al., 2015). In addition, we include **Seq2Seq+Copying** by adding an attention-based copying operation similar to (Jia and Liang, 2016).

The original model does not take the schema into account because it has the same schema in both train and test. We modify the model so that it considers the table schema information by passing a vocabulary mask that limits the model to decode the words from SQL keywords, table and column names in the current database.

SQLNet introduced by (Xu et al., 2017) uses column attention and employs a sketch-based method and generates SQL as a slot-filling task. This fundamentally avoids the sequence-to-sequence structure when ordering does not matter in SQL query conditions. Because it is originally designed for WikiSQL, we extend its `SELECT` and `WHERE` modules to `ORDER BY` and `GROUP BY` components.

TypeSQL proposed by (Yu et al., 2018a) improves upon SQLNet by proposing a different training procedure and utilizing types extracted from either knowledge graph or table content to help model better understand entities and numbers in the question. In our experiment, we

use the question type info extracted from database content. Also, we extend their modules to `ORDER BY` and `GROUP BY` components as well. It is the only model that uses database content.

2.8 Experimental Results and Discussion

We summarize the performance of all models on our test set including accuracy of exact matching in Table 2.2 and F1 scores of component matching in Table 2.3.

Data Splits For the final training dataset, we also select and include 752 queries and 1659 questions that follow our annotation protocol from six existing datasets: Restaurants, GeoQuery, Scholar, Academic, IMDB, and Yelp. We report results on two different settings for all models: (1) Example split where examples are randomly split into 8659 train, 1034 dev, 2147 test. Questions for the same database can appear in both train and test. (2) Database split where 206 databases are split into 146 train, 20 dev, and 40 test. All questions for the same database are in the same split.

Overall Performance The performances of the Seq2Seq-based basic models including Seq2Seq, Seq2Seq+Attention, and Seq2Seq+Copying are very low. However, they are able to generate nested and complex queries because of their general decoding process. Thus, they can get a few hard and extra hard examples correct. But in the vast majority of cases, they predict invalid SQL queries with grammatical errors. The attention and copying mechanisms do not help much either.

In contrast, SQLNet and TypeSQL that utilize SQL structure information to guide the SQL generation process significantly outperform other Seq2Seq models. While they can produce valid queries, however, they are unable to generate nested queries or queries with keywords such as `EXCEPT` and `INTERSECT` because they limit possible SQL outputs in some fixed pre-defined SQL structures.

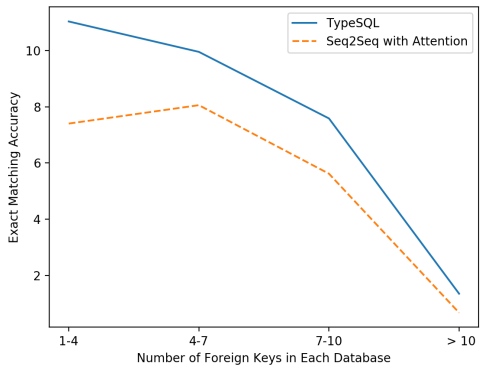

Figure 2.4: Exact matching accuracy as a function of the number of foreign keys.

As Component Matching results in Table 2.3 shows, all models struggle with `WHERE` clause prediction the most. `WHERE` clause is more likely to have multiple columns and operations, which makes its prediction the most challenging. The most number of prediction errors for each component is from column prediction.

In general, the overall performances of all models are low, indicating that our task is challenging and there is still a large room for improvement.

Example Split vs Database Split As discussed in Section 2.5, another challenge of the dataset is to generalize to new databases. To study this, in Table 2.2 and Table 2.3 we compare model performances under the two settings. For all models, the performance under database split is much lower than that under example split. Especially, TypeSQL utilizes column names as question types, and it outperforms other models with a large margin under the example split. However, its performance drops the most on the database split data set. This indicates that the model does well on complex SQL prediction but fails to generalize to new databases. In addition, we observe that all models perform much poorer on column selection under database split than example split.

Overall, the result shows that our dataset presents a challenge for the model to generalize to new databases.

Complexity of Database Schema In order to show how the complexity of the database schema affects model performance, Figure 2.4 plots the exact matching accuracy as a function of the number of foreign keys in a database. The performance decreases as the database has more foreign keys. The first reason is that the model has to choose column and table names from many candidates in a complex database schema. Second, a complex database schema presents a great challenge for the model to capture the relationship between different tables with foreign keys. SQL answers to questions on the database with more number of foreign keys are more likely to join more tables. It indicates that this task requires more effective methods to encode the relation of tables with foreign keys.

2.9 Summary

In this chapter we introduce Spider, a large, complex and cross-domain semantic parsing and text-to-SQL dataset, which directly benefits both NLP and DB communities. Based on Spider, we define a new challenging and realistic semantic parsing task. Experimental results on several state-of-the-art models on this task suggest plenty space for improvement.

2.10 Appendices

2.10.1 SQL Hardness Criteria

To better understand the model performance on different queries, we divide SQL queries into 4 levels: easy, medium, hard, extra hard. We define the difficulty as follows.

We first define:

- SQL components 1: `WHERE, GROUP BY, ORDER BY, LIMIT, JOIN, OR, LIKE, HAVING`
- SQL components 2: `EXCEPT, UNION, INTERSECT, NESTED`

- Others: number of aggregations > 1, number of select columns > 1, number of where conditions > 1, number of group by clauses > 1, number of group by clauses > 1 (no consider col1-col2 math equations etc.)

Then different hardness levels are determined as follows.

- Easy: if SQL key words have ZERO or exact ONE from [SQL components 1] and SQL do not satisfy any conditions in [Others] above. AND no word from [SQL components 2].

- Medium: SQL satisfies no more than two rules in [Others], and does not have more than one word from [SQL components 1], and no word from [SQL components 2]. OR, SQL has exact 2 words from [SQL components 1] and less than 2 rules in [Others], and no word from [SQL components 2]

- Hard: SQL satisfies more than two rules in [Others], with no more than 2 key words in [SQL components 1] and no word in [SQL components 2]. OR, SQL has $2 <$ number key words in [SQL components 1] $<= 3$ and satisfies no more than two rules in [Others] but no word in [SQL components 2]. OR, SQL has no more than 1 key word in [SQL components 1] and no rule in [Others], but exact one key word in [SQL components 2].

- Extra Hard: All others left.

Chapter 3

SParC: Cross-Domain Semantic Parsing in Context

In the previous chapter, we introduced a large-scale text-to-SQL dataset applicable for building a semantic parser that can handle complex questions over databases in random domains. In this chapter, we present SParC, a dataset for cross-domain **S**emantic **Par**sing in **C**ontext. It consists of 4,298 coherent question sequences (12k+ individual questions annotated with SQL queries), obtained from controlled user interactions with 200 complex databases over 138 domains. We provide an in-depth analysis of SParC and show that it introduces new challenges compared to existing datasets. SParC (1) demonstrates complex contextual dependencies, (2) has greater semantic diversity, and (3) requires generalization to new domains due to its cross-domain nature and the unseen databases at test time. We experiment with two state-of-the-art text-to-SQL models adapted to the context-dependent, cross-domain setup. The best model obtains an exact set match accuracy of 20.2% over all questions and less than 10% over all interaction sequences, indicating that the cross-domain setting and the contextual phenomena of the dataset present significant challenges for future research. The dataset, baselines, and leaderboard are released at `https://yale-lily.github.io/sparc`.

3.1 Introduction

Querying a relational database is often challenging and a natural language interface has long been regarded by many as the most powerful database interface (Popescu et al., 2003; Bertomeu et al., 2006; Li and Jagadish, 2014). The problem of mapping a natural language utterance into executable SQL queries (text-to-SQL) has attracted increasing attention from the semantic parsing community by virtue of a continuous effort of dataset creation (Zelle and Mooney, 1996; Iyyer et al., 2017; Zhong et al., 2017; Finegan-Dollak et al., 2018; Yu et al., 2018a) and the modeling innovation that follows it (Xu et al., 2017; Wang et al., 2018; Yu et al., 2018b; Shi et al., 2018).

While most of these work focus on precisely mapping stand-alone utterances to SQL queries, generating SQL queries in a context-dependent scenario (Miller et al., 1996; Zettlemoyer and Collins, 2009; Suhr et al., 2018) has been studied less often. The most prominent context-dependent text-to-SQL benchmark is ATIS[7], which is set in the flight-booking domain and contains only one database (Hemphill et al., 1990; Dahl et al., 1994).

In a real-world setting, users tend to ask a sequence of thematically related questions to learn about a particular topic or to achieve a complex goal. Previous studies have shown that by allowing questions to be constructed sequentially, users can explore the data in a more flexible manner, which reduces their cognitive burden (Hale, 2006; Levy, 2008; Frank, 2013; Iyyer et al., 2017) and increases their involvement when interacting with the system. The phrasing of such questions depends heavily on the interaction history (Kato et al., 2004; Chai and Jin, 2004; Bertomeu et al., 2006). The users may explicitly refer to or omit previously mentioned entities and constraints, and may introduce refinements, additions or substitutions to what has already been said (Figure 3.1). This requires a practical text-to-SQL system to effectively process context information to synthesize the correct SQL logic.

To enable modeling advances in context-dependent semantic parsing, we introduce

7. A subset of ATIS is also frequently used in context-independent semantic parsing research (Zettlemoyer and Collins, 2007; Dong and Lapata, 2016).

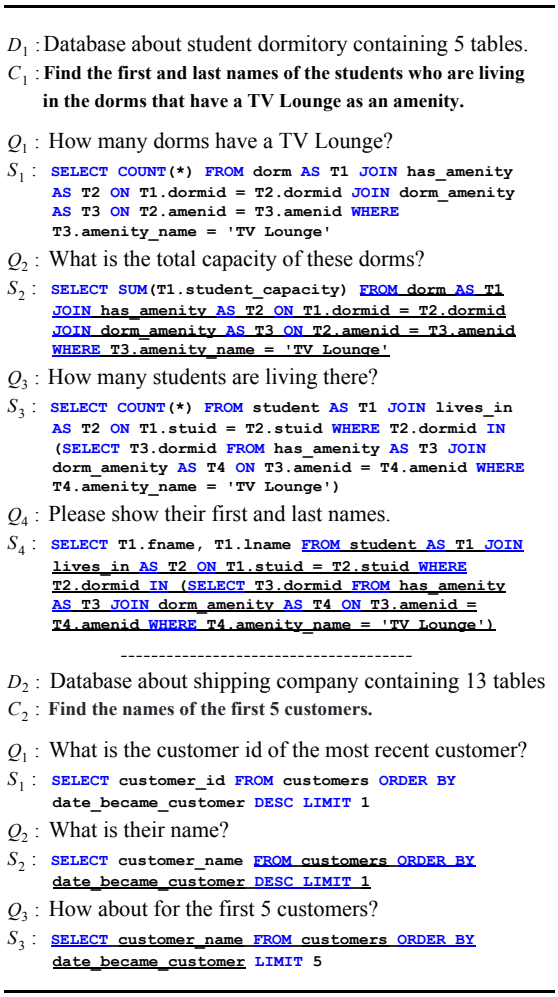

Figure 3.1: Two question sequences from the SParC dataset. Questions (Q_i) in each sequence query a database (D_m), obtaining information sufficient to complete the interaction goal (C_m). Each question is annotated with a corresponding SQL query (S_i). SQL segments from the interaction context are underlined.

SParC (cross-domain **S**emantic **Par**sing in **C**ontext), an expert-labeled dataset which contains 4,298 coherent question sequences (12k+ questions paired with SQL queries) querying 200 complex databases in 138 different domains. The dataset is built on top of Spider[8], the largest cross-domain context-independent text-to-SQL dataset available in the field (Yu et al., 2018c). The large number of domains provide rich contextual phenomena and thematic relations between the questions, which general-purpose natural language interfaces to databases have to address. In addition, it enables us to test the generalization of the trained systems to unseen databases and domains.

8. The data is available at `https://yale-lily.github.io/spider`.

We asked 15 college students with SQL experience to come up with question sequences over the Spider databases. Questions in the original Spider dataset were used as guidance to the students for constructing meaningful interactions: each sequence is based on a question in Spider and the student has to ask inter-related questions to obtain information that answers the Spider question. At the same time, the students are encouraged to come up with related questions which do not directly contribute to the Spider question so as to increase data diversity. The questions were subsequently translated to complex SQL queries by the same student. Similar to Spider, the SQL Queries in SParC cover complex syntactic structures and most common SQL keywords.

We split the dataset such that a database appears in only one of the train, development and test sets. We provide detailed data analysis to show the richness of SParC in terms of semantics, contextual phenomena and thematic relations (§ 3.4). We also experiment with two competitive baseline models to assess the difficulty of SParC (§ 3.5). The best model achieves only 20.2% exact set matching accuracy[9] on all questions, and demonstrates a decrease in exact set matching accuracy from 38.6% for questions in turn 1 to 1.1% for questions in turns 4 and higher (§ 3.6). This suggests that there is plenty of room for advancement in modeling and learning on the SParC dataset.

3.2 Related Work

Context-dependent semantic parsing with SQL labels Only a few datasets have been constructed for the purpose of mapping context-dependent questions to structured queries. (Hemphill et al., 1990; Dahl et al., 1994) collected the contextualized version of ATIS that includes series of questions from users interacting with a flight database. Adopted by several works later on (Miller et al., 1996; Zettlemoyer and Collins, 2009; Suhr et al., 2018),

9. Exact string match ignores ordering discrepancies of SQL components whose order does not matter. Exact set matching is able to consider ordering issues in SQL evaluation. See more evaluation details in section 3.6.1.

Dataset	Context	Resource	Annotation	Cross-domain
SParC	✓	database	SQL	✓
ATIS Hemphill et al. (1990); Dahl et al. (1994)	✓	database	SQL	✗
Spider Yu et al. (2018c)	✗	database	SQL	✓
WikiSQL Zhong et al. (2017)	✗	table	SQL	✓
GeoQuery Zelle and Mooney (1996)	✗	database	SQL	✗
SequentialQA Iyyer et al. (2017)	✓	table	denotation	✓
SCONE Long et al. (2016)	✓	environment	denotation	✗

Table 3.1: Comparison of SParC with existing semantic parsing datasets.

ATIS has only a single domain for flight planning which limits the possible SQL logic it contains. In contrast to ATIS, SParC consists of a large number of complex SQL queries (with most SQL syntax components) inquiring 200 databases in 138 different domains, which contributes to its diversity in query semantics and contextual dependencies. Similar to Spider, the databases in the train, development and test sets of SParC do not overlap.

Context-dependent semantic parsing with denotations Some datasets used in recovering context-dependent meaning (including SCONE (Long et al., 2016) and SequentialQA (Iyyer et al., 2017)) contain no logical form annotations but only denotation (Berant and Liang, 2014) instead. SCONE (Long et al., 2016) contains some instructions in limited domains such as chemistry experiments. The formal representations in the dataset are world states representing state changes after each instruction instead of programs or logical forms. SequentialQA (Iyyer et al., 2017) was created by asking crowd workers to decompose some complicated questions in WikiTableQuestions (Pasupat and Liang, 2015) into sequences of inner-related simple questions. As shown in Table 3.1, neither of the two datasets were annotated with query labels. Thus, to make the tasks feasible, SCONE (Long et al., 2016) and SequentialQA (Iyyer et al., 2017) exclude many questions with rich semantic and contextual types. For example, (Iyyer et al., 2017) requires that the answers to the questions in SequentialQA must appear in the table, and most of them can be solved by simple SQL queries with `SELECT` and `WHERE` clauses. Such direct mapping without formal query labels becomes unfeasible for complex questions. Furthermore, SequentialQA contains

questions based only on a single Wikipedia tables at a time. In contrast, SParC contains 200 significantly larger databases, and complex query labels with all common SQL key components. This requires a system developed for SParC to handle information needed over larger databases in different domains.

Conversational QA and dialogue system Language understanding in context is also studied for dialogue and question answering systems. The development in dialogue (Henderson et al., 2014; Mrkšić et al., 2017; Zhong et al., 2018) uses predefined ontology and slot-value pairs with limited natural language meaning representation, whereas we focus on general SQL queries that enable more powerful semantic meaning representation. Recently, some conversational question answering datasets have been introduced, such as QuAC (Choi et al., 2018) and CoQA (Reddy et al., 2018). They differ from SParC in that the answers are free-form text instead of SQL queries. On the other hand, Kato et al. (2004); Chai and Jin (2004); Bertomeu et al. (2006) conduct early studies of the contextual phenomena and thematic relations in database dialogue/QA systems, which we use as references when constructing SParC.

3.3 Data Collection

We create the SParC dataset in four stages: selecting interaction goals, creating questions, annotating SQL representations, and reviewing.

Interaction goal selection To ensure thematic relevance within each question sequence, we use questions in the original Spider dataset as the thematic guidance for constructing meaningful query interactions, i.e. the interaction goal. Each sequence is based on a question in Spider and the annotator has to ask inter-related questions to obtain the information demanded by the interaction goal (detailed in the next section). All questions in Spider were stand-alone questions written by 11 college students with SQL background after they had

Thematic relation	Description	Example	Percentage
Refinement (constraint refinement)	The current question asks for the same type of entity as a previous question with a different constraint.	Prev_Q: Which **major** has the fewest students? Cur_Q: What is the most popular one?	33.8%
Theme-entity (topic exploration)	The current question asks for other properties about the same entity as a previous question.	Prev_Q: What is *the capacity* of **Anonymous Donor Hall**? Cur_Q: List *all of the amenities* which **it** has.	48.4%
Theme-property (participant shift)	The current question asks for the same property about another entity.	Prev_Q: Tell me the *rating* of **the episode named "Double Down"**. Cur_Q: How about for **"Keepers"**?	9.7%
Answer refinement/theme (answer exploration)	The current question asks for a subset of the entities given in a previous answer or asks about a specific entity introduced in a previous answer.	Prev_Q: Please list all the different **department** *names*. Cur_Q: What is the *average salary* of all instructors in the **Statistics department**?	8.1%

Table 3.2: Thematic relations between questions in a database QA system defined by Bertomeu et al. (2006). The first three relations hold between a question and a previous question and the last relation holds between a question and a previous answer. We manually classified 102 examples in SParC into one or more of them and show the distribution. The entities (**bold**), properties (*italics*) and constraints (underlined) are highlighted in each question.

explored the database content, and the question intent conveyed is likely to naturally arise in real-life query scenarios. We selected all Spider examples classified as medium, hard, and extra hard, as it is in general hard to establish context for easy questions. In order to study more diverse information needs, we also included some easy examples (end up with using 12.9% of the easy examples in Spider). As a result, 4,437 questions were selected as the interaction goals for 200 databases.

Question creation 15 college students with SQL experience were asked to come up with sequences of inter-related questions to obtain the information demanded by the interaction goals[10]. Previous work (Bertomeu et al., 2006) has characterized different thematic relations between the utterances in a database QA system: ***refinement***, ***theme-entity***, ***theme-property***, and ***answer refinement/theme***[11], as shown in Table 3.2. We show these definitions to the

10. The students were asked to spend time exploring the database using a database visualization tool powered by Sqlite Web `https://github.com/coleifer/sqlite-web` so as to create a diverse set of thematic relations between the questions.

11. We group *answer refinement* and *answer theme*, the two thematic relations holding between a question and a previous answer as defined in (Bertomeu et al., 2006), into a single *answer refinement/theme* type.

students prior to question creation to help them come up with context-dependent questions. We also encourage the formulation of questions that are thematically related to but do not directly contribute to answering the goal question (e.g. Q_2 in the first example and Q_1 in the second example in Figure 3.1. See more examples in Appendix as well). The students do not simply decompose the complex query. Instead, they often explore the data content first and even change their querying focuses. Therefore, all interactive query information in SParC could not be acquired by a single complex SQL query.

We divide the goals evenly among the students and each interaction goal is annotated by one student[12]. We enforce each question sequence to contain at least two questions, and the interaction terminates when the student has obtained enough information to answer the goal question.

SQL annotation After creating the questions, each annotator was asked to translate their own questions to SQL queries. All SQL queries were executed on Sqlite Web to ensure correctness. To make our evaluation more robust, the same annotation protocol as Spider (Yu et al., 2018c) was adopted such that all annotators chose the same SQL query pattern when multiple equivalent queries were possible.

Data review and post-process We asked students who are native English speakers to review the annotated data. Each example was reviewed at least once. The students corrected any grammar errors and rephrased the question in a more natural way if necessary. They also checked if the questions in each sequence were related and the SQL answers matched the semantic meaning of the question. After that, another group of students ran all annotated SQL queries to make sure they were executable. Furthermore, they used the SQL parser[13] from Spider to parse all the SQL labels to make sure all queries follow the annotation

12. The most productive student annotated 13.1% of the goals and the least productive student annotated close to 2%.

13. `https://github.com/taoyds/spider/blob/master/process_sql.py`

	SParC	ATIS
Sequence #	4298	1658
Question #	12,726	11,653
Database #	200	1
Table #	1020	27
Avg. Q len	8.1	10.2
Vocab #	3794	1582
Avg. turn #	3.0	7.0

Table 3.3: Comparison of the statistics of context-dependent text-to-SQL datasets.

protocol. Finally, the most experienced annotator conducted a final review on all question-SQL pairs. 139 question sequences were discarded in this final step due to poor question quality or wrong SQL annotations

3.4 Data Statistics and Analysis

We compute the statistics of SParC and conduct a through data analysis focusing on its contextual dependencies, semantic coverage and cross-domain property. Throughout this section, we compare SParC to ATIS (Hemphill et al., 1990; Dahl et al., 1994), the most widely used context-dependent text-to-SQL dataset in the field. In comparison, SParC is significantly different as it (1) contains more complex contextual dependencies, (2) has greater semantic coverage, and (3) adopts a cross-domain task setting, which make it a new and challenging cross-domain context-dependent text-to-SQL dataset.

Data statistics Table 3.3 summarizes the statistics of SParC and ATIS. SParC contains 4,298 unique question sequences, 200 complex databases in 138 different domains, with 12k+ questions annotated with SQL queries. The number of sequences in ATIS is significantly smaller, but it contains a comparable number of individual questions since it has a higher number of turns per sequence[14]. On the other hand, SParC has overcome the domain

14. The ATIS dataset is collected under the Wizard-of-Oz setting (Bertomeu et al., 2006) (like a task-oriented sequential question answering task). Each user interaction is guided by an abstract, high-level goal such as "plan a trip from city A to city B, stop in another city on the way".

The domain by its nature requires the user to express multiple constraints in separate utterances and the user is intrinsically motivated to interact with the system until the booking is successful. In contrast, the interaction

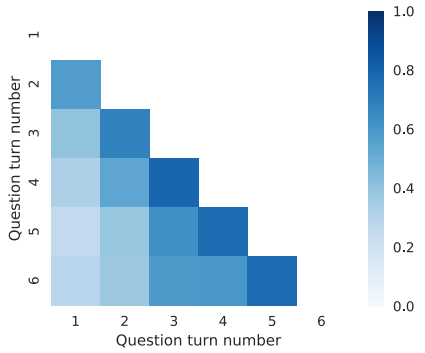

Figure 3.2: The heatmap shows the percentage of SQL token overlap between questions in different turns. Token overlap is greater between questions that are closer to each other and the degree of overlap increases as interaction proceeds.Most questions have dependencies that span 3 or fewer turns.

limitation of ATIS by covering 200 different databases and has a significantly larger natural language vocabulary.

Contextual dependencies of questions We visualize the percentage of token overlap between the SQL queries (formal semantic representation of the question) at different positions of a question sequence. The heatmap shows that more information is shared between two questions that are closer to each other. This sharing increases among questions in later turns, where users tend to narrow down their questions to very specific needs. This also indicates that resolving context references in our task is important.

Furthermore, the lighter color of the lower left 4 squares in the heatmap of Figure 3.2 shows that most questions in an interaction have contextual dependencies that span within 3 turns. Reddy et al. (2018) similarly report that the majority of context dependencies on the CoQA conversational question answering dataset are within 2 questions, beyond which coreferences from the current question are likely to be ambiguous with little inherited information. This suggests that 3 turns on average are sufficient to capture most of the contextual dependencies between questions in SParC.

goals formed by Spider questions are for open-domain and general-purpose database querying, which tend to be more specific and can often be stated in a smaller number of turns. We believe these differences contribute to the shorter average question sequence length of SParC compared to that of ATIS.

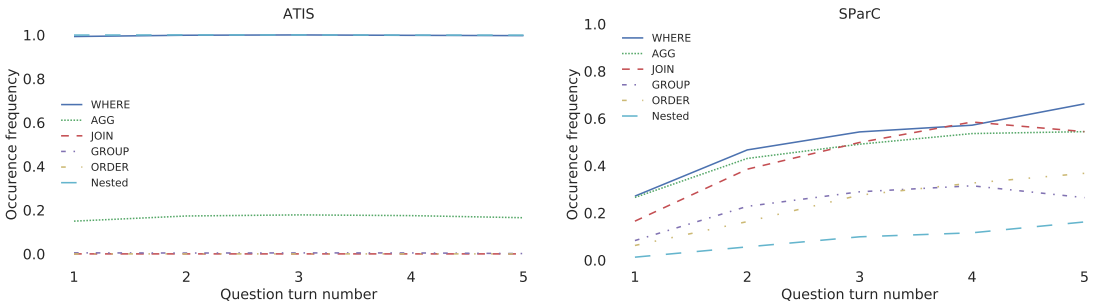

Figure 3.3: Percentage of question sequences that contain a particular SQL keyword at a given turn. The complexity of questions increases as interaction proceeds on SParC as more SQL keywords are triggered. The same trend was not observed on ATIS.

We also plot the trend of common SQL keywords occurring in different question turns for both SParC and ATIS (Figure 3.3) [15]. We show the percentage of question sequences that contain a particular SQL keyword at each turn. The upper figure in Figure 3.3 shows that the occurrences of all SQL keywords do not change much as the question turn increases in ATIS, which indicates that the SQL query logic in different turns are very similar. We examined the data and find that most interactions in ATIS involve changes in the `WHERE` condition between question turns. This is likely caused by the fact that the questions in ATIS only involve flight booking, which typically triggers the use of the ***refinement*** thematic relation. Example user utterances from ATIS are *"on american airlines"* or *"which ones arrive at 7pm"* (Suhr et al., 2018), which only involves changes to the `WHERE` condition.

In contrast, the lower figure demonstrates a clear trend that in SParC, the occurrences of nearly all SQL components increase as question turn increases. This suggests that questions in subsequent turns tend to change the logical structures more significantly, which makes our task more interesting and challenging.

Contextual linguistic phenomena We manually inspected 102 randomly chosen examples from our development set to study the thematic relations between questions. Table 3.2 shows the relation distribution.

15. Since the formatting used for the SQL queries are different in SParC and ATIS, the actual percentages of `WHERE`, `JOIN` and `Nested` for ATIS are lower (e.g. the original version of ATIS may be highly nested, but queries could be reformatted to flatten them). Other SQL keywords are directly comparable.

We find that the most frequently occurring relation is ***theme-entity***, in which the current question focuses on the same entity (set) as the previous question but requests for some other property. Consider Q_1 and Q_2 of the first example shown in Figure 3.1. Their corresponding SQL representations (S_1 and S_2) have the same `FROM` and `WHERE` clauses, which harvest the same set of entities – "dorms with a TV lounge". But their `SELECT` clauses return different properties of the target entity set (number of the dorms in S_1 versus total capacity of the dorms in S_2). Q_3 and Q_4 in this example also have the same relation. The ***refinement*** relation is also very common. For example, Q_2 and Q_3 in the second example ask about the same entity set – customers of a shipping company. But Q_3 switches the search constraint from "the most recent" in Q_2 to "the first 5".

Fewer questions refer to previous questions by changing the entity ("Double Down" versus "Keepers" in Table 3.2) but asking for the same property (***theme-property***). Even less frequently, some questions ask about the answers of previous questions (***answer refinement/theme***). As in the last example of Table 3.2, the current question asks about the "Statistics department", which is one of the answers returned in the previous turn. More examples with different thematic relations are provided in Figure 3.5 in the Appendix.

Interestingly, as the examples in Table 3.2 have shown, many thematic relations are present without explicit linguistic markers. This indicates that information tends to implicitly propagate through the interaction. Moreover, in some cases where the natural language question shares information with the previous question (e.g. Q_2 and Q_3 in the first example of Figure 3.1 form a ***theme-entity*** relation), the corresponding SQL representations (S_2 and S_3) can be very different. One scenario in which this happens is when the property/constraint specification makes reference to additional entities described by separate tables in the database schema.

Semantic coverage As shown in Table 3.3, SParC is larger in terms of number of unique SQL templates, vocabulary size and number of domains compared to ATIS. The smaller

number of unique SQL templates and vocabulary size of ATIS is likely due to the domain constraint and presence of many similar questions.

SQL components	SParC	ATIS
# WHERE	42.8%	99.7%
# AGG	39.8%	16.6%
# GROUP	20.1%	0.3%
# ORDER	17.0%	0.0%
# HAVING	4.7%	0.0%
# SET	3.5%	0.0%
# JOIN	35.5%	99.9%
# Nested	5.7%	99.9%

Table 3.4: Distribution of SQL components in SQL queries. SQL queries in SParC cover all SQL components, whereas some important SQL components like ORDER are missing from ATIS.

Table 3.4 further compare the formal semantic representation in these two datasets in terms of SQL syntax component. While almost all questions in ATIS contain joins and nested subqueries, some commonly used SQL components are either absent (ORDER BY, HAVING, SET) or occur very rarely (GROUP BY and AGG). We examined the data and find that many questions in it has complicated syntactic structures mainly because the database schema requires joined tables and nested sub-queries, and the semantic diversity among the questions is in fact smaller.

Cross domain As shown in Table 3.1, SParC contains questions over 200 databases (1,020 tables) in 138 different domains. In comparison, ATIS contains only one databases in the flight booking domain, which makes it unsuitable for developing models that generalize across domains. Interactions querying different databases are shown in Figure 3.1 (also see more examples in Figure 3.4 in the Appendix). As in Spider, we split SParC such that each database appears in only one of train, development and test sets. Splitting by database requires the models to generalize not only to new SQL queries, but also to new databases and new domains.

	Train	Dev	Test
# Q sequences	3034	422	842
# Q-SQL pairs	9025	1203	2498
# Databases	140	20	40

Table 3.5: Dataset Split Statistics

3.5 Methods

We extend two state-of-the-art semantic parsing models to the cross-domain, context-dependent setup of SParC and benchmark their performance. At each interaction turn i, given the current question $\bar{x}_i = \langle x_{i,1}, \ldots, x_{i,|\bar{x}_i|} \rangle$, the previously asked questions $\bar{I}[: i-1] = \{\bar{x}_1, \ldots, \bar{x}_{i-1}\}$ and the database schema C, the model generates the SQL query $\bar{y}_i$.

3.5.1 Seq2Seq with turn-level history encoder (CD-Seq2Seq)

This is a cross-domain Seq2Seq based text-to-SQL model extended with the turn-level history encoder proposed in (Suhr et al., 2018).

Turn-level question history encoder Following (Suhr et al., 2018), at turn i, we encode each user question $\bar{x}_t \in \bar{I}[: t-1] \cup \{\bar{x}_i\}$ using an utterance-level bi-LSTM, LSTM^E. The final hidden state of LSTM^E, $\mathbf{h}^E_{t,|\bar{x}_t|}$, is used as the input to the turn-level encoder, LSTM^I, a uni-directional LSTM, to generate the discourse state $\mathbf{h}^I_t$. The input to LSTM^E at turn t is the question word embedding concatenated with the discourse state at turn $t-1$ ($[\mathbf{x}_{t,j}, \mathbf{h}^I_{t-1}]$), which enables the flow of contextual information.

Database schema encoding For each column header in the database schema, we concatenate its corresponding table name and column name separated by a special dot token (i.e., `table_name.column_name`), and use the average word embedding[16] of tokens in this sequence as the column header embedding $\mathbf{h}^C$.

16. We use the 300-dimensional GloVe (Pennington et al., 2014) pretrained word embeddings.

Decoder The decoder is implemented with another LSTM (LSTM^D) with attention to the LSTM^E representations of the questions in η previous turns (η is a hyperparameter). At each decoding step, the decoder chooses to generate either a SQL keyword (e.g., `select`, `where`, `group by`) or a column header. To achieve this, we use separate layers to score SQL keywords and column headers, and finally use the softmax operation to generate the output probability distribution over both categories.

3.5.2 SyntaxSQLNet with history input (SyntaxSQL-con)

SyntaxSQLNet is a syntax tree based neural model for the complex and cross-domain context-independent text-to-SQL task introduced by (Yu et al., 2018b). The model consists of a table-aware column attention encoder and a SQL-specific syntax tree-based decoder. The decoder adopts a set of inter-connected neural modules to generate different SQL syntax components.

We extend this model by providing the decoder with the encoding of the previous question ($\bar{x}_{i-1}$) as additional contextual information. Both $\bar{x}_i$ and $\bar{x}_{i-1}$ are encoded using bi-LSTMs (of different parameters) with the column attention mechanism proposed by (Yu et al., 2018b). We use the same math formulation to inject the representations of $\bar{x}_i$ and $\bar{x}_{i-1}$ to each syntax module of the decoder.

More details of each baseline model can be found in the Appendix. And we opensource their implementations for reproducibility.

3.6 Experiments

3.6.1 Evaluation Metrics

Following (Yu et al., 2018c), we use the exact set match metric to compute the accuracy between gold and predicted SQL answers. Instead of simply employing string match, Yu et al.

(2018c) decompose predicted queries into different SQL clauses such as `SELECT`, `WHERE`, `GROUP BY`, and `ORDER BY` and compute scores for each clause using set matching separately[17]. We report the following two metrics: *question match*, the exact set matching score over all questions, and *interaction match*, the exact set matching score over all interactions. The exact set matching score is 1 for each question only if all predicted SQL clauses are correct, and 1 for each interaction only if there is an exact set match for every question in the interaction.

3.6.2 Results

Model	Question Match		Interaction Match	
	Dev	Test	Dev	Test
CD-Seq2Seq	17.1	18.3	**6.7**	**6.4**
SyntaxSQL-con	**18.5**	**20.2**	4.3	5.2
SyntaxSQL-inp	15.2	16.9	0.7	1.1

Table 3.6: Performance of various methods over all questions (*question match*) and all interactions (*interaction match*). Note that CD-Seq2Seq is able to achieve 37.5 and 43.6 on ATIS devlopement and test sets. Compared to 6.7 and 6.4 on our dataset, it shows that our dataset introduces some new challenges in sequential semantic parsing.

We report the overall results of CD-Seq2Seq and SyntaxSQLNet on the development and the test data in Table 3.6. The context-aware models (CD-Seq2Seq and SyntaxSQL-con) significantly outperforms the context-agnostic SyntaxSQLNet (SyntaxSQL-inp). The last two rows form a controlled ablation study, where without accessing to the previous question, the test set performance of SyntaxSQLNet decreases from 20.2% to 16.9% on *question match* and from 5.2% to 1.1% on *interaction match*, which indicates that context is a crucial aspect of the problem.

We note that SyntaxSQL-con scores higher in *question match* but lower in *interaction match* compared to CD-Seq2Seq.

17. Details of the evaluation metrics can be found at `https://github.com/taoyds/spider/tree/master/evaluation_examples`

A closer examination shows that SyntaxSQL-con predicts more questions correctly in the early turns of an interaction (Table 3.7), which results in its overall higher *question match* accuracy. A possible reason for this is that SyntaxSQL-con adopts a stronger context-agnostic text-to-SQL module (SyntaxSQLNet vs. Seq2Seq adopted by CD-Seq2Seq). The higher performance of CD-Seq2Seq on *interaction match* can be attributed to better incorporation of information flow between questions by using turn-level encoders (Suhr et al., 2018), which is possible to encode the history of all previous questions comparing to only single one previous question in SyntaxSQL-con. Overall, the lower performance of the two extended context-dependent models shows the difficulty of SParC and that there is ample room for improvement.

Turn #	CD-Seq2Seq	SyntaxSQL-con
1 (422)	31.4	38.6
2 (422)	12.1	11.6
3 (270)	7.8	3.7
$\geq$ 4 (89)	2.2	1.1

Table 3.7: Performance stratified by question turns on the development set. The performance of the two models decrease as the interaction continues.

Performance stratified by question position To gain more insight into how question position affects the performance of the two models, we report their performances on questions in different positions in Table 3.7. Questions in later turns of an interaction in general have greater dependency over previous questions and also greater risk for error propagation. The results show that both CD-Seq2Seq and SyntaxSQL-con consistently perform worse as the question turn increases, suggesting that both models struggle to deal with information flow from previous questions and accumulate errors. Moreover, SyntaxSQL-con significantly outperforms CD-Seq2Seq on questions in the first turn, but the advantage disappears in later turns (starting from the second turn), which is expected because the context encoding mechanism of SyntaxSQL-con is less powerful than the turn-level encoders adopted by CD-Seq2Seq.

Goal Difficulty	CD-Seq2Seq	SyntaxSQL-con
Easy (483)	35.1	**38.9**
Medium (441)	7.0	**7.3**
Hard (145)	**2.8**	1.4
Extra hard (134)	**0.8**	0.7

Table 3.8: Performance stratified by question difficulty on the development set. The performances of the two models decrease as questions are more difficult.

Performance stratified by SQL difficulty We group individual questions in SParC into different difficulty levels based on the complexity of their corresponding SQL representations using the criteria proposed in (Yu et al., 2018c) (The hardness definition does not account for the effect of turn position). As shown in Figure 3, the questions tended to get harder as interaction proceeds, more questions with hard and extra hard difficulties appear in late turns. Table 3.8 shows the performance of the two models across each difficulty level. As we expect, the models perform better when the user request is easy. Both models fail on most hard and extra hard questions. Considering that the size and question types of SParC are very close to Spider, the relatively lower performances of SyntaxSQL-con on medium, hard and extra hard questions in Table 3.8 comparing to its performances on Spider (17.6%, 16.3%, and 4.9% respectively) indicates that SParC introduces additional challenge by introducing context dependencies, which is absent from Spider.

Thematic relation	CD-Seq2Seq	SyntaxSQL-con
Refinement	8.4	6.5
Theme-entity	13.5	10.2
Theme-property	9.0	7.8
answer refine./them.	12.3	20.4

Table 3.9: Performance stratified by thematic relations. The models perform best on the *answer refinement/theme* relation, but do poorly on the *refinement* and *theme-property* relations.

Performance stratified by thematic relation Finally, we report the model performances across thematic relations computed over the 102 examples summarized in Table 3.2. The results (Table 3.9) show that the models, in particular SyntaxSQL-con, perform the best on

the ***answer refinement/theme*** relation.

A possible reason for this is that questions in the ***answer theme*** category can often be interpreted without reference to previous questions since the user tends to state the theme entity explicitly. Consider the example in the bottom row of Table 3.2. The user explicitly said "Statistics department" in their question, which belongs to the answer set of the previous question [18]. The overall low performance for all thematic relations (***refinement*** and ***theme-property*** in particular) indicates that the two models still struggle on properly interpreting the question history.

3.7 Summary

In this chapter, we introduced SParC, a large-scale dataset of context-dependent questions over a number of databases in different domains annotated with the corresponding SQL representation. The dataset features wide semantic coverage and a diverse set of contextual dependencies between questions. It also introduces unique challenge in mapping context-dependent questions to SQL queries in unseen domains. We experimented with two competitive context-dependent semantic parsing approaches on SParC. The model accuracy is far from satisfactory and stratifying the performance by question position shows that both models degenerate in later turns of interaction, suggesting the importance of better context modeling. The dataset, baseline implementations and leaderboard are publicly available at `https://yale-lily.github.io/sparc`.

18. As pointed out by one of the anonymous reviewers, there are less than 10 examples of *answer refinement/theme* relation in the 102 analyzed examples. We need to see more examples before concluding that this phenomenon is general.

3.8 Appendices

3.8.1 Additional Baseline Model Details

CD-Seq2Seq We use a bi-LSTM, LSTM^E, to encode the user utterance at each turn. At each step j of the utterance, LSTM^E takes as input the word embedding and the discourse state $\mathbf{h}^I_{i-1}$ updated for the previous turn $i-1$:

$$\mathbf{h}^E_{i,j} = \text{LSTM}^E([\mathbf{x}_{i,j}; \mathbf{h}^I_{i-1}], \mathbf{h}^E_{i,j-1})$$

where i is the index of the turn and j is the index of the utterance token. The final hidden state LSTM^E is used as the input of a uni-directional LSTM, LSTM^I, which is the interaction level encoder:

$$\mathbf{h}^I_i = \text{LSTM}^I(\mathbf{h}^E_{|x_i|}, \mathbf{h}^I_{i-1}).$$

For each column header, we concatenate its table name and its column name separated by a special dot token (i.e., `table_name . column_name`), and the column header embedding $\mathbf{h}^C$ is the average embeddings of the words.

The decoder is implemented as another LSTM with hidden state $\mathbf{h}^D$. We use the dot-product based attention mechanism to compute the context vector. At each decoding step k, we compute attention scores for all tokens in η previous turns (we use $\eta = 5$) and normalize them using softmax. Suppose the current turn is i, and consider the turns of $0, \ldots, \eta - 1$ distance from turn i. We use a learned position embedding $\phi^I(i-t)$ when computing the attention scores. The context vector is the weighted sum of the concatenation of the token

embedding and the position embedding:

$$
\begin{aligned}
s_k(t,j) &= [\mathbf{h}^E_{t,j}; \phi^I(i-t)]\mathbf{W}_{\text{att}}\mathbf{h}^D_k \\
\alpha_k &= \textbf{softmax}(s_k) \\
\mathbf{c}_k &= \sum_{t=i-h}^{i}\sum_{j=1}^{|x_t|}\alpha_k(t,j)[\mathbf{h}^E_{t,j}; \phi^I(i-t)]
\end{aligned}
$$

At each decoding step, the sequential decoder chooses to generate a SQL keyword (e.g., `select`, `where`, `group by`, `order by`) or a column header. To achieve this, we use separate layers to score SQL keywords and column headers, and finally use the softmax operation to generate the output probability distribution:

$$
\begin{aligned}
\mathbf{o}_k &= \tanh([\mathbf{h}^D_k; \mathbf{c}_k]\mathbf{W}_o) \\
\mathbf{m}^{\text{SQL}} &= \mathbf{o}_k\mathbf{W}_{\text{SQL}} + \mathbf{b}_{\text{SQL}} \\
\mathbf{m}^{\text{column}} &= \mathbf{o}_k\mathbf{W}_{\text{column}}\mathbf{h}^C \\
P(y_k) &= \textbf{softmax}([\mathbf{m}^{\text{SQL}}; \mathbf{m}^{\text{column}}])
\end{aligned}
$$

It's worth mentioning that we experimented with a SQL segment copying model similar to the one proposed in (Suhr et al., 2018). We implement our own segment extraction procedure by extracting `SELECT`, `FROM`, `GROUP BY`, `ORDER BY` clauses as well as different conditions in `WHERE` clauses. In this way, we can extract 3.9 segments per SQL on average. However, we found that adding segment copying does not significantly improve the performance because of error propagation. Better leveraging previously generated SQL queries remains an interesting future direction for this task.

SyntaxSQL-con As in (Yu et al., 2018b), the following is defined to compute the conditional embedding $\mathbf{H}_{1/2}$ of an embedding $\mathbf{H}_1$ given another embedding $\mathbf{H}_2$:

$$
\mathbf{H}_{1/2} = \textbf{softmax}(\mathbf{H}_1\mathbf{W}\mathbf{H}_2^\top)\mathbf{H}_1.
$$

Here $\mathbf{W}$ is a trainable parameter. In addition, a probability distribution from a given score matrix $\mathbf{U}$ is computed by

$$\mathcal{P}(\mathbf{U}) = \textbf{softmax}\left(\mathbf{V}\textbf{tanh}(\mathbf{U})\right),$$

where $\mathbf{V}$ is a trainable parameter. To incorporate the context history, we encode the question right before the current question and add it to each module as an input. For example, the COL module of SyntaxSQLNet is extended as following. $\mathbf{H}_{\text{PQ}}$ denotes the hidden states of LSTM on embeddings of the previous one question and the $\mathbf{W}_3^{\text{num}}\mathbf{H}_{\text{PQ/COL}}^{\text{num}}{}^{\top}$ and $\mathbf{W}_4^{\text{val}}\mathbf{H}_{\text{PQ/COL}}^{\text{val}}{}^{\top}$ terms add history information to prediction of the column number and column value respectively.

$$P_{\text{COL}}^{\text{num}} = \mathcal{P}\left(\mathbf{W}_1^{\text{num}}\mathbf{H}_{\text{Q/COL}}^{\text{num}}{}^{\top} + \mathbf{W}_2^{\text{num}}\mathbf{H}_{\text{HS/COL}}^{\text{num}}{}^{\top} + \mathbf{W}_3^{\text{num}}\mathbf{H}_{\text{PQ/COL}}^{\text{num}}{}^{\top}\right)$$

$$P_{\text{COL}}^{\text{val}} = \mathcal{P}\left(\mathbf{W}_1^{\text{val}}\mathbf{H}_{\text{Q/COL}}^{\text{val}}{}^{\top} + \mathbf{W}_2^{\text{val}}\mathbf{H}_{\text{HS/COL}}^{\text{val}}{}^{\top} + \mathbf{W}_3^{\text{val}}\mathbf{H}_{\text{COL}}{}^{\top} + \mathbf{W}_4^{\text{val}}\mathbf{H}_{\text{PQ/COL}}^{\text{val}}{}^{\top}\right)$$

3.8.2 Additional Data Examples

We provide additional SParC examples in Figure 3.4 and examples with different thematic relations in Figure 3.5.

D_3 : Database about wine.

C_4 : **Find the county where produces the most number of wines with score higher than 90.**

Q_1 : How many different counties are all wine appellations from?

S_1 : `SELECT COUNT(DISTINCT county) FROM appellations`

Q_2 : How many wines does each county produce?

S_2 : `SELECT T1.county, COUNT(*) FROM appellations AS T1 JOIN wine AS T2 ON T1.appellation = T2.appellation GROUP BY T1.county`

Q_3 : Only show the counts of wines that score higher than 90?

S_3 : `SELECT T1.county, COUNT(*) FROM appellations AS T1 JOIN wine AS T2 ON T1.appellation = T2.appellation WHERE T2.score > 90 GROUP BY T1.county`

Q_4 : Which county produced the greatest number of these wines?

S_4 : `SELECT T1.county FROM appellations AS T1 JOIN wine AS T2 ON T1.appellation = T2.appellation WHERE T2.score > 90 GROUP BY T1.county ORDER BY COUNT(*) DESC LIMIT 1`

D_5 : Database about districts

C_5 : **Find the names and populations of the districts whose area is greater than the average area.**

Q_1 : What is the total district area?

S_1 : `SELECT sum(area_km) FROM district`

Q_2 : Show the names and populations of all the districts.

S_2 : `SELECT name, population FROM district`

Q_3 : Excluding those whose area is smaller than or equals to the average area.

S_3 : `SELECT name, population FROM district WHERE area_km > (SELECT avg(area_km) FROM district)`

D_6 : Database about books

C_6 : **Find the title, author name, and publisher name for the top 3 best sales books.**

Q_1 : Find the titles of the top 3 highest sales books.

S_1 : `SELECT title FROM book ORDER BY sale_amount DESC LIMIT 3`

Q_2 : Who are their authors?

S_2 : `SELECT t1.name FROM author AS t1 JOIN book AS t2 ON t1.author_id = t2.author_id ORDER BY t2.sale_amount DESC LIMIT 3`

Q_3 : Also show the names of their publishers.

S_3 : `SELECT t1.name, t3.name FROM author AS t1 JOIN book AS t2 ON t1.author_id = t2.author_id JOIN press AS t3 ON t2.press_id = t3.press_id ORDER BY t2.sale_amount DESC LIMIT 3`

Figure 3.4: More examples in SParC.

D_7 : Database about school departments
C_7 : **What are the names and budgets of the departments with average instructor salary greater than the overall average?**

Q_1 : Please list all different department names.
S_1 : `SELECT DISTINCT dept_name FROM department`
Q_2 : Show me the budget of the Statistics department. ***(Theme/refinement-answer)***
S_2 : `SELECT budget FROM department WHERE dept_name = "Statistics"`
Q_3 : What is the average salary of instructors in that department? ***(Theme-entity)***
S_3 : `SELECT AVG(T1.salary)FROM instructor as T1 JOIN department as T2 ON T1.department_id = T2.id WHERE T2.dept_name = "Statistics"`
Q_4 : How about for all the instructors? ***(Refinement)***
S_4 : `SELECT AVG(salary) FROM instructor`
Q_5 : Could you please find the names of the departments with average instructor salary less than that? ***(Theme/refinement-answer)***
S_5 : `SELECT T2.dept_name FROM instructor as T1 JOIN department as T2 ON T1.department_id = T2.id GROUP BY T1.department_id HAVING AVG(T1.salary) < (SELECT AVG(salary) FROM instructor)`
Q_6 : Ok, how about those above the overall average? ***(Refinement)***
S_6 : `SELECT T2.dept_name FROM instructor as T1 JOIN department as T2 ON T1.department_id = T2.id GROUP BY T1.department_id HAVING AVG(T1.salary) > (SELECT AVG(salary) FROM instructor)`
Q_7 : Please show their budgets as well. ***(Theme-entity)***
S_7 : `SELECT T2.dept_name, T2.budget FROM instructor as T1 JOIN department as T2 ON T1.department_id = T2.id GROUP BY T1.department_id HAVING AVG(T1.salary) > (SELECT AVG(salary) FROM instructor)`

Figure 3.5: Additional example in SParC annotated with different thematic relations. Entities (purple), properties (magenta), constraints (red), and answers (orange) are colored.

Chapter 4

CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Natural Language Interfaces to Databases

In chapter 3, we presented SPARC, a context-dependent text-to-SQL dataset. However, it assumes all questions can be mapped into SQL queries without involving system responses. This chapter introduces CoSQL, a corpus for building cross-domain, general-purpose database (DB) querying dialogue systems. It consists of 30k+ turns plus 10k+ annotated SQL queries, obtained from a Wizard-of-Oz (WOZ) collection of 3k dialogues querying 200 complex DBs spanning 138 domains. Each dialogue simulates a real-world DB query scenario with a crowd worker as a user exploring the DB and a SQL expert retrieving answers with SQL, clarifying ambiguous questions, or otherwise informing of unanswerable questions. When user questions are answerable by SQL, the expert describes the SQL and execution results to the user, hence maintaining a natural interaction flow. CoSQL introduces new challenges compared to existing task-oriented dialogue datasets:

(1) the dialogue states are grounded in SQL, a domain-independent executable representation, instead of domain-specific slot-value pairs, and (2) because testing is done on unseen databases, success requires generalizing to new domains. CoSQL includes three tasks: SQL-grounded dialogue state tracking, response generation from query results, and user dialogue act prediction. We evaluate a set of strong baselines for each task and show that CoSQL presents significant challenges for future research. The dataset, baselines, and leaderboard will be released at `https://yale-lily.github.io/cosql`.

4.1 Introduction

Natural language interfaces to databases (NLIDB) have been studied extensively, with a multitude of different approaches introduced over the past few decades. To this end, considerable progress has been made in querying data via natural language (NL). However, most NL query systems expect the query to be well-formed and stated in a single sentence (Zelle and Mooney, 1996; Li and Jagadish, 2014; Yaghmazadeh et al., 2017; Iyer et al., 2017; Zhong et al., 2017; Xu et al., 2017; Shi et al., 2018; Wang et al., 2018; Yu et al., 2018b). In reality, complex questions are usually answered through interactive exchanges (Figure 1.2). Even for simple queries, people tend to explore the database by asking multiple basic, interrelated questions (Hale, 2006; Levy, 2008; Frank, 2013; Iyyer et al., 2017). This requires systems capable of sequentially processing conversational requests to access information in relational databases. To drive the progress of building a context-dependent NL query system, corpora such as ATIS (Hemphill et al., 1990; Dahl et al., 1994) and SParC (Yu et al., 2019b)[19] have been released. However, these corpora assume all user questions can be mapped into SQL queries and do not include system responses.

Furthermore, in many cases, multi-turn interaction between users and NL systems is needed to clarify ambiguous questions (e.g., Q_3 and R_3 in Figure 1.2), verify returned results,

19. SParC task is available at `https://yale-lily.github.io/sparc`

and notify users of unanswerable or unrelated questions. Therefore, a robust dialogue-based NL query agent that can engage with users by forming its own responses has become an increasingly necessary component for the query process. Such systems have already been studied under task-oriented dialogue settings by virtue of continuous effort of corpus creation (Seneff and Polifroni, 2000; Walker et al., 2002; Raux et al., 2005; Mrksic et al., 2015; Asri et al., 2017; Budzianowski et al., 2018) and modelling innovation (Artzi and Zettlemoyer, 2011; Henderson et al., 2013; Lee and Dernoncourt, 2016; hao Su et al., 2016; Dhingra et al., 2016; Li et al., 2016). The goal of these systems is to help users accomplish a specific task, such as flight or hotel booking or transportation planning. However, to achieve these goals, task-oriented dialogue systems rely on pre-defined slots and values for request processing (which can be represented using simple SQL queries consisting of `SELECT` and `WHERE` clauses). Thus, these systems only operate on a small number of domains and have difficulty capturing the diverse semantics of practical user questions.

In contrast, the goal of dialogue-based NLIDB systems is to support general-purpose exploration and querying of databases by end users. To do so, these systems must possess the ability to (1) detect questions answerable by SQL, (2) ground user questions into executable SQL queries if possible, (3) return results to the user in a way that is easily understood and verifiable, and (4) handle unanswerable questions. The difficulty of constructing dialogue-based NLIDB systems stems from these requirements. To enable modeling advances in this field, we introduce CoSQL, the first large-scale cross-domain **Co**nversational text-to-**SQL** corpus collected under the WOZ setting (Budzianowski et al., 2018). CoSQL contains 3,007 dialogues (more than 30k turns with annotated dialogue acts and 10k expert-labeled SQL queries) querying 200 complex DBs spanning across 138 different domains. For each dialogue, we follow the WOZ set-up that involves a crowd worker as a DB user and a college computer science student who is familiar with SQL as an expert (§4.3).

Like Spider[20] (Yu et al., 2018c) and SParC (Yu et al., 2019b), the cross-domain setting in

20. Spider task is available at `https://yale-lily.github.io/spider`

CoSQL enables us to test the ability of systems to generalize on querying different domains via dialogues. We split the dataset in a way that each database only appears in one of train, development, or test set. This setting requires systems to generalize to new domains without additional annotation.

More importantly, unlike most prior work in text-to-SQL systems, CoSQL demonstrates greater language diversity and more frequent user focus changes. It also includes a significant amount of questions that require user clarification and questions that cannot be mapped to SQL queries, introducing the potential to evaluate text-to-SQL dialog act prediction. These features pose new challenges for text-to-SQL systems. Moreover, CoSQL includes system responses that describe SQL queries and the returned results in a way that is easy for users with different backgrounds to understand and verify, as faithful and comprehensible presentation of query results is a crucial component of any NLIDB system.[21]

We introduce three challenge tasks on CoSQL: (1) **SQL-grounded dialogue state tracking** to map user utterances into SQL queries if possible given the interaction history (§4.5.1), (2) **natural language response generation** based on an executed SQL and its results for user verification (§4.5.2) and (3) **user dialogue act prediction** to detect and resolve ambiguous and unanswerable questions (§4.5.3). We provide detailed data analysis and qualitative examples (§4.4). For each of the three tasks, we benchmark several competitive baseline models (§4.6). The performances of these models indicate plenty of room for improvement.

4.2 Related Work

Task-oriented dialog systems Task-oriented dialog systems (Henderson et al., 2014; Wen et al., 2016; Mrkšić et al., 2017; Budzianowski et al., 2018) have attracted increasing

21. The DB community has developed query visualization (Li and Jagadish, 2014) and other techniques to provide faithful explanation of a SQL query. These explanations are complementary to the NL ones used in our work and future NLIDB systems could integrate them.

attention especially due to their commercial values. The goal is to help users accomplish a specific task such as hotel reservation, flight booking, or travel information. These systems (Bordes and Weston, 2017; Zhong et al., 2018; Wu et al., 2019) often pre-define slot templates grounded to domain-specific ontology, limiting the ability to generalize to unseen domains. In comparison, our work is to build a system for general-purpose DB exploration and querying. The domain-independent intent representation (SQL query) enables the trained system to work on unseen domains (DB schemas).

While most task-oriented dialog systems need to actively poke the user for information to fill in pre-defined slot-value pairs, the primary goal of system responses in CoSQL is to offer users a reliable way to understand and verify the returned results. If a question can be converted into a SQL query, the user is shown the execution result and the system will describe the SQL query and the result in natural language. In case sthe user questions are ambiguous or unanswerable by SQL, the system either requests the user to rephrase or informs them to ask other questions.

Data-to-Text generation Response generation in CoSQL takes a structured SQL query and its corresponding result table to generate an NL description of the system's interpretation of the user request. Compared to most dialogue-act-to-text generation tasks, the richer semantics of SQL queries makes our task more challenging – besides generating natural and coherent descriptions, faithfully preserving the logic of a SQL query in an NL response is also crucial in our task. Furthermore, this component is related to previous work on text generation from structured data (McKeown, 1985; Iyer et al., 2016; Wiseman et al., 2017).

4.3 Data Collection

We follow the Wizard-of-Oz setup which facilitates dialogues between DB users and SQL experts to create CoSQL. We recruited Amazon Mechanical Turkers (AMT) to act as DB users and trained 25 graduate- and undergraduate-level computer science students proficient

in SQL to act as DB experts. The collection interface (Lasecki et al., 2013) is designed to be easy-to-operate for the experts and intuitive for the users. Detailed explanations of the data collection process is provided below.

Reference goal selection We pre-select a reference goal for each dialogue to ensure the interaction is meaningful and to reduce redundancy within the dataset. Users are asked to explore the given DB content to come up with questions that are likely to naturally arise in real-life scenarios and reflect their query intentions as specified by the reference goals. Following (Yu et al., 2019b), we selected the complex questions classified as medium, hard, and extra hard in Spider (Yu et al., 2018c) as the reference goals.[22] In total, 3,783 questions were selected on 200 databases. After annotation and reviewing, 3,007 of them were finished and kept in the final dataset.

User setup We developed online chatting interfaces to pair the user with the expert (Figure 4.5 and 4.6 in Appendix). When a data collection session starts, the user is first shown multiple tables from a DB to which a reference goal is groundedand is required to read through them. Once they have examined the data stored in the tables, the reference goal question will be revealed on the same screen. The user is encouraged to use the goal question as a guide to ask interrelated questions, but is also allowed to ask other questions exploring the DB. We require the user to ask at least 3 questions.[23] In each turn, if the user question can be answered by a SQL query, they will be shown the result table, and the expert will write an NL response interpreting the executed SQL query based on their understanding of the user's query intent (Figure 4.7 Appendix). If the user question is ambiguous or cannot be answered with SQL, they will receive clarification questions or notice to rephrase from

22. Yu et al. (2019b) also includes 12.9% of the easy questions in Spider in order to increase dataset diversity. In this work we prioritize the complex questions that trigger more interesting interactions and do not include any easy questions.

23. The worker is paid $1.20 USD for each dialog. To encourage interaction, we offer $0.50 USD bonus for each dialogue if the user asks more than 4 interesting, interrelated questions.

the expert (detailed in expert setup).

Expert setup Within each session, the expert is shown the same DB content and the reference goal as the user (Figure 4.7 in Appendix). For each dialogue turn, the expert first checks the user question and labels it using a set of pre-defined user dialog action types (DATs, see Table 4.4). Then the expert sets the DAT of his response according to the user DAT. Both the user and the expert can have multiple DATs labels in each turn. If the user question is answerable in SQL (labeled as `INFORM_SQL`, e.g. Q_1 in Figure 1.2), the expert writes down the SQL query[24], executes it, checks the result table, and sends the result table to the user. The expert then describes the SQL query and result table in natural language and sends the response. If the user question is ambiguous, the expert needs to write an appropriate response to clarify the ambiguity (labeled as `AMBIGUOUS`, e.g. Q_3 in Figure 1.2). Some user questions require the expert to infer the answer based on their world knowledge (labeled as `INFER_SQL`, e.g. Q_3 in Figure 4.9). If the user question cannot be answered by SQL, the expert will inform them to ask well-formed questions (labeled as `NOT_RELATED`, `CANNOT_UNDERSTAND`, or `CANNOT_ANSWER`). In other cases (labeled as `GREETING`, `THANK_YOU`, etc.), the expert responds with general dialogue expressions (Q_8 in Figure 1.2).

User quality control Because of the real-time dialogue setting and the expensive annotation procedures on the expert side, conducting quality control on user is crucial for our data collection. We use LegionTools[25] (Lasecki et al., 2014) to post our tasks onto AMT and to recruit and route AMT workers for synchronous real time crowd sourcing tasks. We specify that only workers from the U.S. with 95% approval rates are allowed to accept our task. Before proceeding to the chat room, each AMT worker has to go through

24. We use the same SQL annotation protocol as Spider (Yu et al., 2018c) to ensure the same SQL pattern was chosen when multiple equivalent queries were available.

25. `https://www.cromalab.net/LegionTools/`

a tutorial and pass two short questions[26] to test their knowledge about our task. Only the user who passes the quiz proceeds to the chat room. Throughout the data collection, if a user ignores our instructions in a specific turn, we allow the experts to alert the user through chat and label the corresponding turn as DROP. If a user's actions continue to deviate from instructions, the expert can terminate the dialog before it ends. After each dialogue session terminates, we ask the expert to provide a score from 1 to 5 as an evaluation of the user's performance. Dialogues with a score below 3 are dropped and the user will be blocked from future participation.

Data review and post-process We conduct a multi-pass data reviewing process.[27] Two students conducted a first-round review. They focus on correcting any errors in the DATs of the users and the experts, checking if the SQL queries match the user's questions, and modifying or rewriting the expert's responses to contain necessary information in the SQL queries in case they miss any of them. Also, they re-evaluate all dialogues based on the diversity of user questions and reject any dialogues that only contain repeated, simple, and thematically-independent user questions (about 6% of the dialogs). After the first-round review, another two student experts reviewed the refined data to double check the correctness of the DATs, the SQL queries, and the expert responses. They also corrected any grammar errors, and rephrased the user's questions and the expert's responses in a more natural way if necessary. Finally, we ran and parsed all annotated SQL queries to make sure they were executable, following the same annotation protocol as the Spider dataset.

26. One is on how to ask interrelated questions and the other is on how to read multiple tables with reference keys.

27. The review interface is shown in Figure 4.8 (Appendix).

4.4 Data Statistics and Analysis

We report the statistics of CoSQL and compare it to other task-oriented dialog and context-dependent text-to-SQL datasets. We also conduct detailed analyses on its contextual, cross-domain nature, and question diversity.

	DSTC2	WOZ 2.0	KVRET	MultiWOZ	**CoSQL**
# dialogs	1,612	600	2,425	8,438	2,164
Total # turns	23,354	4,472	12,732	115,424	22,422
Total # tokens	199,431	50,264	102.077	1,520,970	22.8197
Avg. # turns/dialog	14.49	7.45	5.25	13.68	10.36
Avg. # tokens/turn	8.54	11.24	8.02	13.18	11.34
Total # unique tokens	986	2,142	2,842	24,071	7,502
# databases	1	1	1	7	140
# Slots #	8	4	13	25	3,696
# Values #	212	99	1,363	4,510	>1,000,000

Table 4.1: Comparison of CoSQL to some commonly used task-oriented dialogue datasets. The numbers are computed for the training part of data in consistency with previous work Budzianowski et al. (2018).

	CoSQL	SParC	ATIS
# Q sequence	3,007	4298	1658
# user questions	15,598*	12,726	11,653
# databases	200	200	1
# tables	1020	1020	27
Avg. Q len	11.2	8.1	10.2
Vocab	9,585	3794	1582
Avg. # Q turns	5.2	3.0	7.0
Unanswerable Q	✓	✗	✗
User intent	✓	✗	✗
System response	✓	✗	✗

Table 4.2: Comparison of CoSQL with other context-dependent text-to-SQL datasets. The number are computed over the entire datasets. *For CoSQL we count the total # user utterances.

Data statistics Table 4.1å and 4.2 summarize the statistics of CoSQL. CoSQL contains 3k+ dialogues in total (2,164 in training), which is comparable to or bigger than most commonly used task-oriented dialogue datasets. Figure 4.1 shows the distribution of dialogue length in the corpus, approximately 80% of dialogues involve 8 or more turns, with a total of

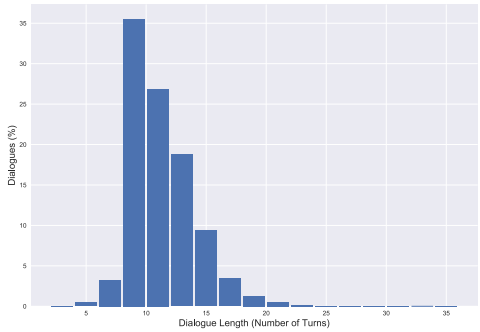

Figure 4.1: Distributions of dialogue lengths.

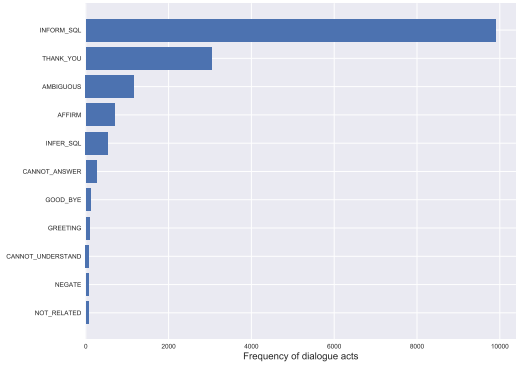

Figure 4.2: Distributions of user dialog action types.

31,148 turns.[28] The average number of tokens in each turn is 11.21. Noticeably, the domain of CoSQL spans over 200 complex databases, overshadowing most other task-oriented dialogue datasets. Comparing to existing context-dependent text-to-SQL datasets, CoSQL contains significantly more turns, out of which 11,039 user utterances are convertible to SQL. In contrast, all NL utterances in ATIS and SParC can be mapped to SQL. CoSQL also has a much larger NL vocabulary.

Dialogue act distribution As shown in Figure 4.2, CoSQL contains a fairly diverse set of user dialogue action types (DATs). Unsurprisingly, `INFORM_SQL` and `THANK_YOU` are the two most commonly seen DATs. Among the rest of DATs, approximately 40% are

28. Following (Budzianowski et al., 2018), in the statistics report we define the # turns in a dialogue to be the total # messages in the dialogue.

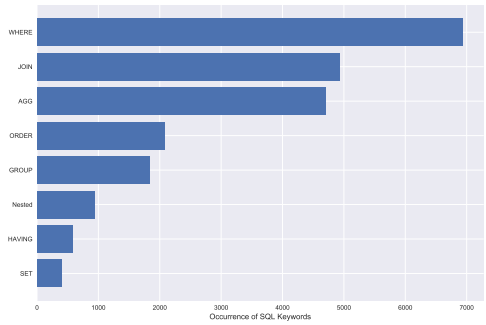

Figure 4.3: SQL keyword counts.

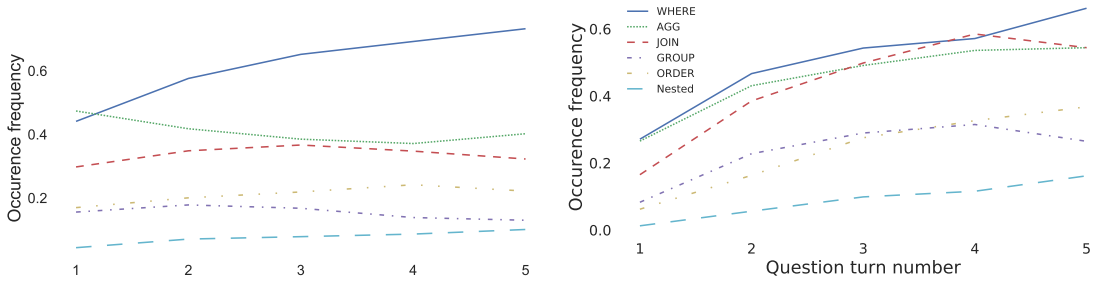

Figure 4.4: Percentage of question sequences that contain a particular SQL keyword at a specific user utterance turn. The keyword occurrences in CoSQL (upper) slightly fluctuates as the interaction proceeds while that in SParC (lower) demonstrates a clear increasing trend.

`AMBIGUOUS`, demonstrating the paramount importance of system clarification in the DB querying process in general. Another 20% of this subgroup is `INFER_SQL`, which signifies questions that cannot be answered without the aid of human inference.

Semantic complexity As shown in Table 4.1, if we consider the column names of the 200 DBs of CoSQL as slots and their entries as values, the number of slot-value pairs far exceed those defined in other task-oriented dialogues. Figure 4.3 shows the total number of occurrences of different SQL keywords in the SQL queries corresponding to these questions. The SQL queries in CoSQL cover all common SQL keywords as well as complicated syntactic structure such as nesting (Figure 4.4).

Semantic changes by turns We compute the frequency of occurrences of common SQL keywords in different turns for both CoSQL and SParC and compare them in Figure 4.4

(upper: CoSQL, lower: SParC). Here we count the turn # based on user utterance only. Since CoSQL and SParC span the same domains, Figure 4.4 reveals a comparison of semantic changes between context-dependent DB questions issued by end users (CoSQL) and expert users (SParC). For CoSQL, the frequencies of all keywords except for `WHERE` do not change significantly throughout the conversation, and the average frequencies of these keywords are in general lower than those of SParC. In addition, `WHERE` occurs slightly more frequently in CoSQL than in SParC. We believe this indicates the exploratory nature of the dialogues we collected, as the users switch their focus more frequently instead of building questions upon previous ones. For example, SQL `AGG` components occur most frequently in the beginning of dialogues, as a result of users familiarizing themselves with the amount of data in the DB or other statistical measures. In contrast, the frequencies of almost all SQL components in SParC increase as the question turn increases. This suggests that questions in SParC have stronger inter-dependency, as the purpose of this corpus is to study text-to-SQL in context.

Cross domain As shown in Table 4.3, the dialogues in CoSQL are randomly split into train, development and test sets by DB with a ratio of 7:1:2 (the same split as SParC and Spider).

	Train	Dev	Test
# Dialogs	2164	292	551
# Databases	140	20	40

Table 4.3: Dataset Split Statistics

4.5 Tasks and Models

CoSQL is meant to be used as the first benchmark for building general-purpose DB querying dialogue systems in arbitrary domains. Such systems take a user question and determine if it can be answered by SQL (user dialogue act prediction). If the question can be answered by SQL, the system translates it into the corresponding SQL query (SQL-grounded dialogue

state tracking), executes the query, returns and shows the result to the user. To improve interpretability and trustworthiness of the result, the system describes the predicted SQL query and result tables to the user for their verification (response generation from SQL and result). Finally, the user checks the results and the system responses and decides if the desired information is obtained or additional questions shall be asked.

Some components relevant to the process above are beyond the scope of our work. First, our response generation task only includes turns where the system's dialogue act is `CONFORM_SQL`. In case the system cannot understand the user's question (the system dialogue act is `CANNOT_ANSWER`) or considers it as unanswerable (`CANNOT_ANSWER`), the system will reply in a standard way to inform the user that it needs clarification or cannot answer that question. The same applies to questions that require human inference (e.g., the system confirms with the user which types of dorms he or she was talking about by asking R_3 instead of immediately translating Q_3 in Figure 1.2). Currently we do not have a task setup to evaluate the quality of system clarifications. Second, some user questions cannot be directly answered by SQL but are possible to be answered with other type of logical reasoning (e.g., Q_3 in Figure 4.9). We exclude these questions from our task design and leave them for future research.

4.5.1 SQL-Grounded Dialogue State Tracking

In CoSQL, user dialogue states are grounded in SQL queries. Dialogue state tracking (DST) in this case is to predict the correct SQL query for each user utterance with `INFORM_SQL` label given the interaction context and the DB schema. In our setup, the system does not have access to gold SQL queries from previous turns, which is different from the traditional DST settings in dialogue management where the history of ground-truth dialogue states is given. Comparing to other context-dependent text-to-SQL tasks such as SParC and ATIS, the DST task in CoSQL also include the ambiguous questions if the user affirms the system clarification of them (e.g., Q_4 in Figure 1.2). In this case, the system clarification is also

given as part of the interaction context to predict the SQL query corresponding to the question.[29][30] For instance, to generate S_4 in Figure 1.2, the input consists of all previous questions (Q_1, Q_2, Q_3), the current user question (Q_4), the DB schema, and the system response R_3.

We benchmark the performance of two strong context-dependent neural text-to-SQL models on this the task, which are the baseline models reported on SParC by (Yu et al., 2019b).

Context-dependent Seq2Seq (CD-Seq2Seq) The model is originally introduced by (Suhr et al., 2018) for the ATIS task. It incorporates interaction history and is able to copy segments of previous generated SQL queries. Yu et al. (2019b) extends it to encode DB schema information such that it works for the cross-domain setting in SParC. We apply the model to our task without any changes.

SyntaxSQL-con SyntaxSQLNet is a SQL-specific syntax-tree based model introduced for Spider (Yu et al., 2018b). Yu et al. (2019b) extends it to take previous questions as input when predicting SQL for the current question. We apply the model to our task without any changes.

4.5.2 Response Generation from SQL and Query Results

This task requires generating a natural language description of the SQL query and the result for each system response labeled as `CONFORM_SQL`. It considers a SQL query, the execution result, and the DB schema. Preserving logical consistency between SQL and NL response is crucial in this task, in addition to naturalness and syntactical correctness. Unlike other

29. If a dialogue contains multiple ambiguous questions, the system clarification to all ambiguous questions will be given as input.

30. The ambiguous questions not confirmed by the user and their system responses are given as part of the conversation history but we do not require a system to predict SQL queries for them.

SQL-to-text generation tasks (Xu et al., 2018), our task maps the SQL query to a statement and summarizes the result in that statement (instead of just mapping it back to the user question).

We experiment with three baseline methods for this task.

Template-based Given the SQL and NL response pairs in the training set, we masked variable values in both the SQL and NL response to form parallel SQL-response templates. Given a new SQL query, we employ rule-based approach to select the closest SQL-response template pair from the set. After that, we fill in the selected response template with the columns, tables, and values of the SQL query and the result to generate the final response (see more in Appendix).

Seq2Seq We experiment with a vanilla Seq2Seq model (Sutskever et al., 2014) with attention (Bahdanau et al., 2015), a standard baseline for text generation tasks.

Pointer-generator Oftentimes the column or table names in the NL response are copied from the input SQL query. To capture this phenomenon, we experiment with a pointer-generator network (See et al., 2017), which addresses the problem of out-of-vocabulary word generation in summarization and other text generation tasks. We use a modified version of the implementation from (Chen and Bansal, 2018).

4.5.3 User Dialogue Act Prediction

Groups	Dialog acts
DB user	`inform_sql`, `infer_sql`, `ambiguous`, `affirm`, `negate`, `not_related`, `cannot_understand`, `cannot_answer`, `greeting`, `goodbye`, `thank_you`
DB expert	`conform_sql`, `clarify`, `reject`, `request_more`, `greeting`, `sorry`, `welcome`, `goodbye`

Table 4.4: Dialog acts in CoSQL. See § 4.8.1 for the comprehensive definition of each dialogue act.

For a real-world DB querying dialogue system, it has to decide if the user question can be mapped to a SQL query or if special actions are needed. We define a series of dialogue acts for the DB user and the SQL expert (Table 4.4).[31] For example, if the user question can be answered by a SQL query, the dialogue act of the question is `INFORM_SQL`. Since the system DATs are defined in response to the user DATs, our task does not include system dialogue acts prediction.

We experiment with two baseline models for this task.

Majority The dialogue acts of all the user questions are predicted to be the majority dialogue act `INFORM_SQL`.

TBCNN-pair We employ TBCNN-pair (Mou et al., 2016), a tree-based CNN model with heuristics for predicting entailment and contradiction between sentences. We change the two sentence inputs for the model to a user utterance and the DB schema, and follow the same method in SQLNet (Xu et al., 2017) to encode each column name.

4.6 Results and Discussion

Model	Question Match		Interaction Match	
	Dev	Test	Dev	Test
CD-Seq2Seq	13.8	13.9	2.1	2.6
SyntaxSQL-con	15.1	14.1	2.7	2.2

Table 4.5: Performance of various methods over all questions (*question match*) and all interactions (*interaction match*).

SQL-grounded dialog state tracking We use the same evaluation metrics used by the SParC dataset (Yu et al., 2019b) to evaluate the model's performance on all questions and interactions (dialogs). The performances of CD-Seq2Seq and SyntaxSQL-con are reported in Table 4.5. The two models achieve less than 16% question-level accuracy and less than

31. §4.8.1 defines the complete set of dialogue action types.

3% on interaction-level accuracy. Since the two models have been benchmarked on both CoSQL and SParC, we cross-compare their performance on these two datasets. Both models perform significantly worse on CoSQL DST than on SParC. This indicates that CoSQL DST is more difficult than SParC. The possible reasons is that the questions in CoSQL are generated by a more diverse pool of users (crowd workers instead of SQL experts), the task includes ambiguous questions and the context contains more complex intent switches.

Model	BLEU		LCR (%)	Grammar
	Dev	Test	Test	Test
Template	9.5	9.3	41.0	4.0
Seq2Seq	15.3	14.1	27.0	3.5
Pointer-generator	16.4	15.1	35.0	3.6

Table 4.6: BLEU scores on the development and test sets, and human evaluations of logic correctness rate (LCR) and grammar check on the 100 examples randomly sampled from the test set.

Response generation Table 4.6 shows the results of three different baselines on three metrics: BLEU score (Papineni et al., 2002), logic correctness rate (LCR), and grammar. To compute LCR and grammar score, we randomly sampled 100 descriptions generated by each model. Three students proficient in English participated in the evaluation, They were asked to choose a score 0 or 1 for LCR, and 1 to 5 for grammar check (the larger, the better). For LCR, the final score was decided by majority vote. We computed the average grammar score.

Interestingly, the human evaluation and BLEU scores do not completely agree. While the template-based method is brittle and requires manual effort, it performs significantly better than the two end-to-end neural models in the human evaluation. Because the SQL-question templates provide natural and grammatical sketch of the output, it serves as an advantage in our human evaluation. However, this approach is limited by the small coverage of the training templates and its LCR is only around 40%. On the other hand, the neural models achieve better BLEU scores than the template-based approach. A possible reason for this is

that they tend to generate words frequently associated with certain SQL queries. However, the neural models struggle to preserve the SQL query logic in the output. Unsurprisingly, pointer-generator performs better than basic Seq2Seq in terms of both BLEU and human evaluation. The low performances of all methods on LCR show that the task is indeed very challenging.

Model	Dev	Test
Majority	63.3	62.8
TBCNN-pair	84.2	83.9

Table 4.7: Accuracy of user dialog act prediction on the development and test sets.

User dialog act prediction Table 4.7 shows the accuracy of the two baselines on predicting user dialog acts. The result of Majority indicates that about 40% of user questions cannot be directly converted into SQL queries. This confirms the necessity of considering a larger set of dialogue actions for building a practical NLIDB system. Even though TBCNN can predict around 85% of user intents correctly, most of the correct predictions are for simple classes such as `INFORM_SQL`, `THANK_YOU`, and `GOODBYE` etc. The F-scores for more interesting and important dialog acts such as `INFER_SQL` and `AMBIGUOUS` are around 10%. This indicates that improving the accuracy on user DAT prediction is still important.

4.7 Summary

In this chapter, we introduce CoSQL, the first large-scale cross-domain conversational text-to-SQL corpus collected under a Wizard-of-Oz setup. Its language and discourse diversity and cross-domain setting raise exciting open problems for future research. Especially, the baseline model performances on the three challenge tasks suggest plenty space for improvement. The data and challenge leaderboard will be publicly available at `https://yale-lily.github.io/cosql`.

Future Work As discussed in Section 4.5, some examples in CoSQL include ambiguous and unanswerable user questions and we do not study how a system can effectively clarify those questions or guide the user to ask questions that are answerable. Also, some user questions cannot be answered with SQL but by other forms of logical reasoning the correct answer can be derived. We urge the community to investigate these problems in future work in order to build practical, robust and reliable conversational natural language interfaces to databases.

4.8 Appendices

This section provides description of dialog actions in A.1, more details on baseline modifications and hyperparameters in A.2, system response guides in A.3, additional dialog examples in CoSQL dataset in Figure 4.9 and 4.10, and the DB user (AMT turkers) and the SQL expert (college computer science students) annotation interfaces in Figure 4.5, 4.6, 4.7, and 4.8.

4.8.1 Description of Dialog Acts

For the DB user, we define the following dialog acts:

`INFORM_SQL:` The user informs his/her request if the user's question can be answered by SQL. The system needs to write SQL.

`INFER_SQL:` If the user's question must be answered by SQL+human inference. For example, users' questions are "are they..?" (yes/no question) or "the 3rd oldest...". SQL cannot directly (or unnecessarily complicated) return the answer, but we can infer the answer based on the SQL results.

`AMBIGUOUS:` The user's question is ambiguous, the system needs to double check the user's intent (e.g. what/did you mean by...?) or ask for which columns to return.

`AFFIRM:` Affirm something said by the system (user says yes/agree).

NEGATE: : Negate something said by the system (user says no/deny).

NOT_RELATED: The user's question is not related to the database, the system reminds the user.

CANNOT_UNDERSTAND: The user's question cannot be understood by the system, the system asks the user to rephrase or paraphrase question.

CANNOT_ANSWER: The user's question cannot be easily answered by SQL, the system tells the user its limitation.

GREETING: Greet the system.

GOOD_BYE: Say goodbye to the system.

THANK_YOU: Thank the system.

For the system, we define the following dialog acts:

CONFIRM_SQL: The system creates a natural language response that describes SQL and result table, and asks the user to confirm if the system understood his/her intention.

CLARIFY: Ask the user to double check and clarify his/her intention when the user's question is ambiguous.

REJECT: Tell the user you did not understand/cannot answer his/her question, or the user question is not related.

REQUEST_MORE: Ask the user if he/she would like to ask for more info.

GREETING: Greet the user.

SORRY: Apologize to the user.

WELCOME: Tell the user he/she is welcome.

GOOD_BYE: Say goodbye to the user.

4.8.2 Modifications and Hyperparameters for Baselines

CD-Seq2Seq We apply the model with the same settings used in SParC without any changes.

SyntaxSQL-con We apply the model with the same settings used in SParC without any changes.

Template-based We first create a list of SQL query patterns without values, column and table names that cover the most cases in the train set of CoSQL. And then we manually changed the patterns and their corresponding responses to make sure that table, column, and value slots in the responses have one-to-one map to the slots in the SQL query. Once we have the SQL-response mapping list, during the prediction, new SQL statements are compared with every templates to find the best template to use. A score will be computed to represent the similarity between the SQL and each template. The score is computed based on the number of each SQL key components existing in the SQL and each template. Components of the same types are grouped together to allow more flexible matching, like count, max, min are grouped to aggregate. A concrete example of templates is shown: `SELECT column0 FROM table0 WHERE column1 comparison0 value0`. `column0,1` and `table0` represent column name and table name respectively. `comparison0` represents one of the comparison operator including `>=, <=, <,>,=,!=, and like`. `value0` represents a value the user uses to constrain the query result.

Seq2Seq We train a word2vec embedding model on the concatenation of the SQL query and response output of the training data for the embedding layer of our Seq2Seq model. We use an embedding dimension of 128, hidden dimension of 256, a single-layer bi-directional LSTM encoder and uni-directional LSTM decoder with attention. We use a batch size of 32, clip the norm of the gradient at 2.0, and do early stopping on the validation loss with a patience of 5. We perform decoding with greedy search.

Pointer-generator We follow the same settings as in the Seq2Seq case with the addition of the copy mechanism during training and testing.

TBCNN-pair The model is modified mainly on the sentence embedding part and classifier part. The input of the modified model is a user utterance and the related column names. Therefore, we replace one of the two sentence embedding modules with a database column name encoding module, which generates representations of the column names related to the sentence. The classifier is modified by adding a label(user dialogue act) number predicting module, which predicts the number of the labels(user dialogue acts) of the user utterance. The label number prediction module is similar to the column number prediction module in SQLNet.

4.8.3 System Response Guide

System response should be standard and professional. We follow the rules below to write responses for different system dialog action type:

`CLARIFY` ”Did you mean...?”, ”What did you mean by...?”, or anything similar.

`REJECT` ”Sorry, I don’t have the answer to your question...” or anything similar.

`REQUEST_MORE` ”Do you want anything else?” or anything similar.

`CONFORM_SQL` We convert SQL written by us back to natural language. (We should use the column names and values in the SQL). Our response has to describe all information in the SQL independently instead of referring to any previous context or subject.

1. If the returned result can be combined with the SQL description, combine them together to generate the response. For example:

   Given SQL:

   SELECT AVG(SALARY) FROM INSTRUCTOR

   Result Returned: 200k

   Your Response:“The average salary of all instructors is 200k.”

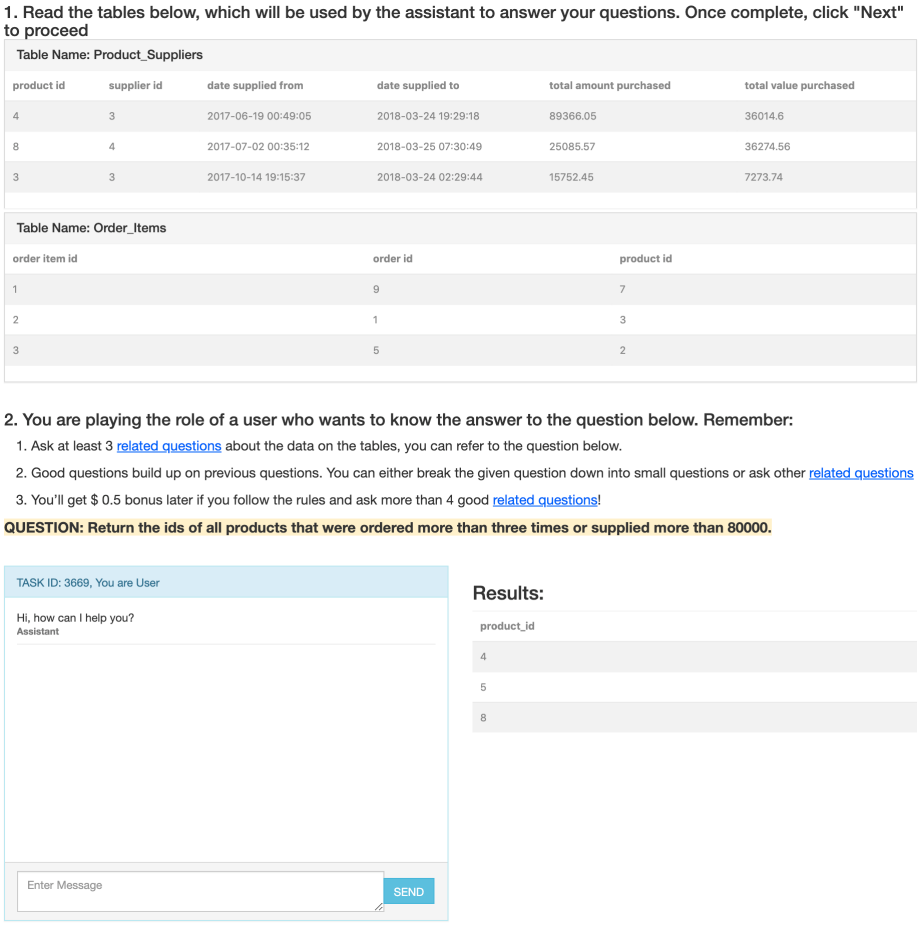

Figure 4.5: DB User Interface

2. If the returned result is too large and cannot be combined with the SQL description, describe them separately. For example:

   Given SQL:

   SELECT AVG(T1.SALARY),T1.DEPARTMENT_ID FROM INSTRUCTOR AS T1 JOIN DEPARTMENT AS T2 ON T1.DEPARTMENT_ID = T2.ID GROUP BY T1.DEPARTMENT_ID

   Result Returned: a long table

   Your Response:"Here is the result table that shows the average salary in each department. For example, the average of CS professors is 250k."

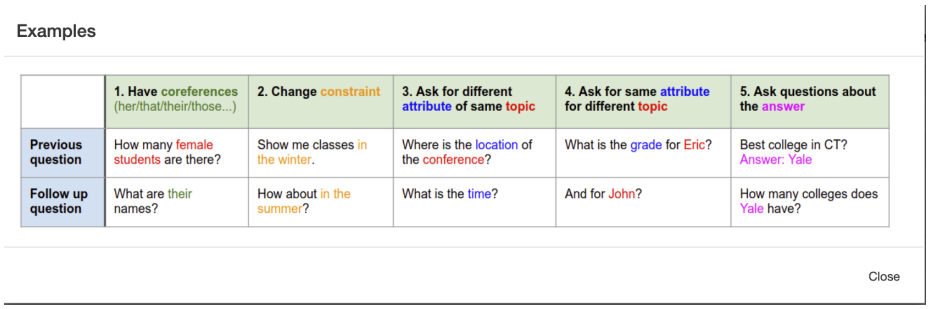

Figure 4.6: DB User Related Questions: a pop-up window when the user clicks highlighted "related questions" in the above interface.

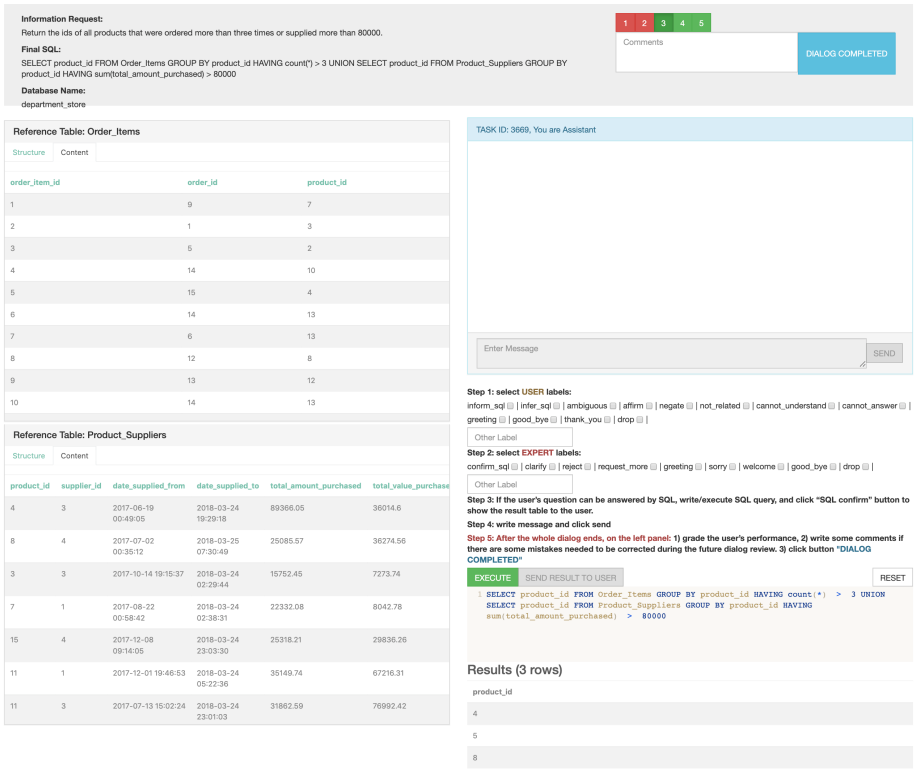

Figure 4.7: SQL Expert Interface

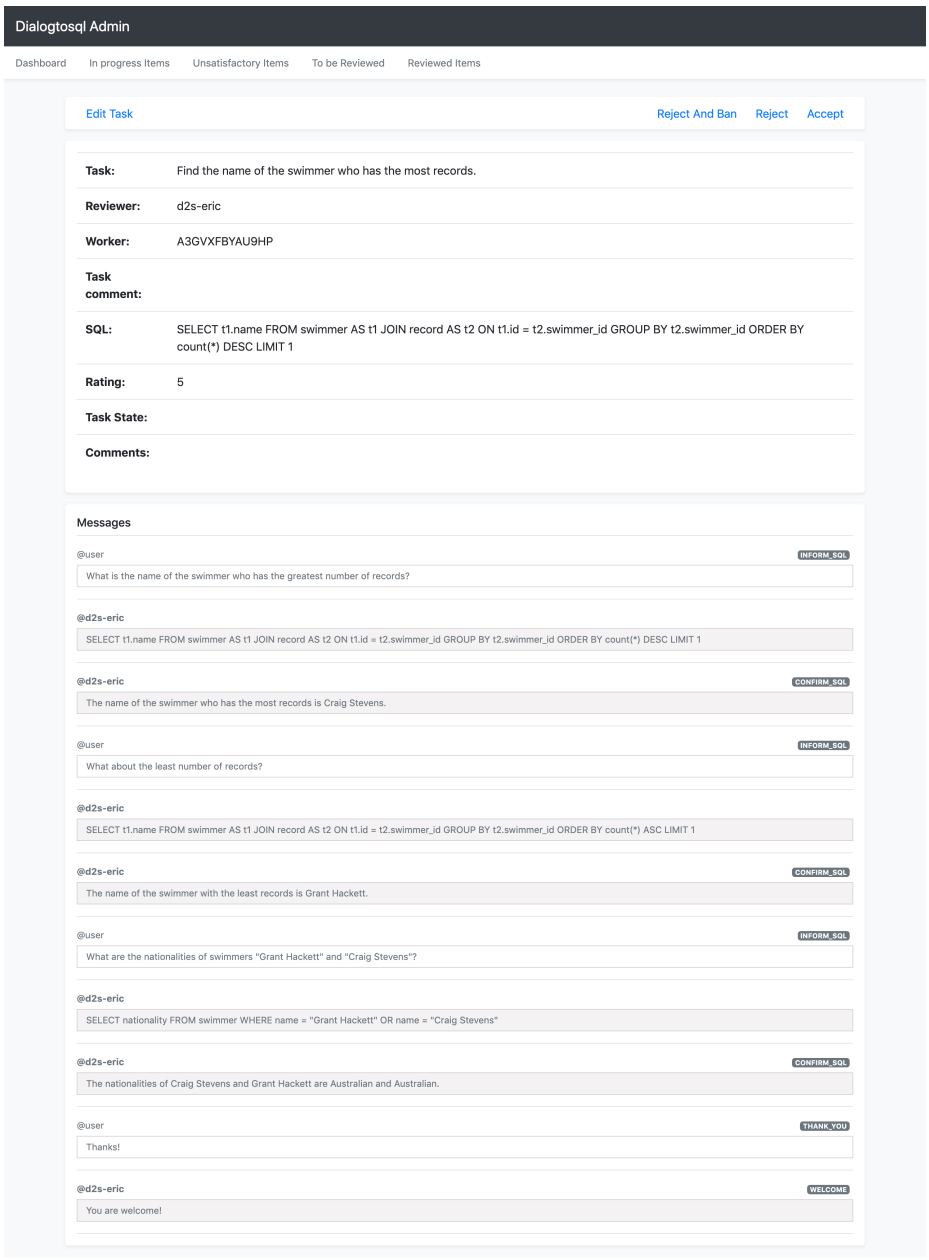

Figure 4.8: Dialogue Review Interface

D_1 : Database about soccer containing 3 tables

...

Q_2 : What are their names? INFORM_SQL

S_2 : `SELECT cName FROM College WHERE enr > 18000`

A_2 : OU | FSU

R_2 : The names of the colleges that have enrollment values greater than 18000 are OU and FSU CONFIRM_SQL

Q_3 : Which one has the higher enrollment? INFER_SQL

S_3 : `SELECT cName FROM College WHERE enr > 18000 ORDER BY enr DESC LIMIT 1`

A_3 : OU

R_3 : Among the results, OU has a higher enrollment. CONFIRM_SQL

...

Figure 4.9: Part of a dialogue example with INFER_SQL user dialog label

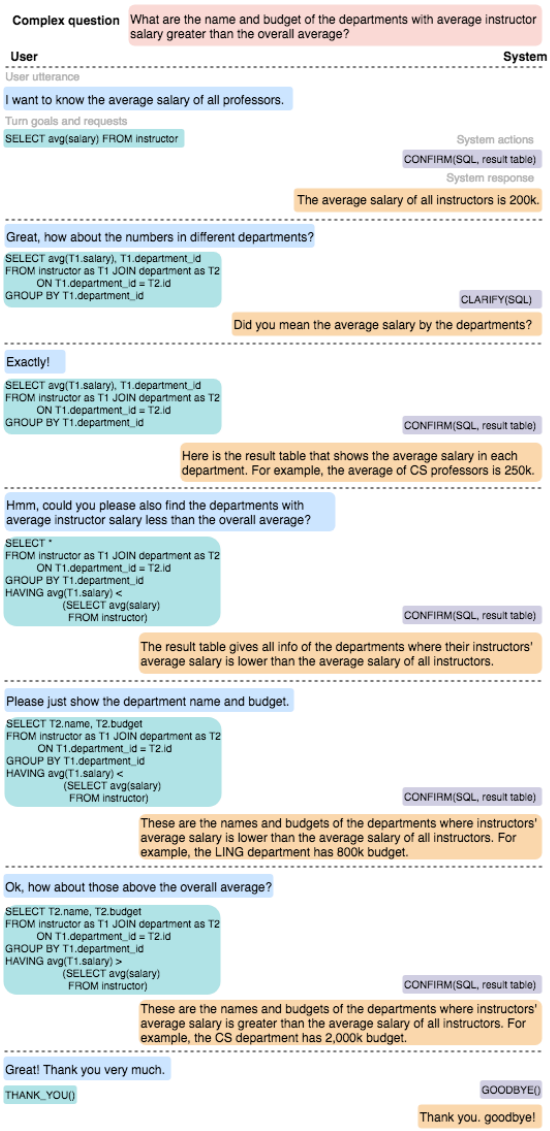

Figure 4.10: A complete dialogue example

Part II

Algorithms

Chapter 5

TypeSQL: Knowledge-based Type-Aware Neural Text-to-SQL Generation

In this chapter, we present a novel approach, TYPESQL, which views text-to-SQL problem as a slot filling task. Additionally, TYPESQL utilizes type information to better understand rare entities and numbers in natural language questions. We test this idea on the WikiSQL dataset and outperform the prior state-of-the-art by 5.5% in much less time. We also show that accessing the content of databases can significantly improve the performance when users' queries are not well-formed. TYPESQL gets 82.6% accuracy, a 17.5% absolute improvement compared to the previous content-sensitive model.

5.1 Introduction

We consider the WikiSQL task proposed by (Zhong et al., 2017), a large scale benchmark dataset for the text-to-SQL problem. Given a natural language question for a table and the table's schema, the system needs to produce a SQL query corresponding to the question. We

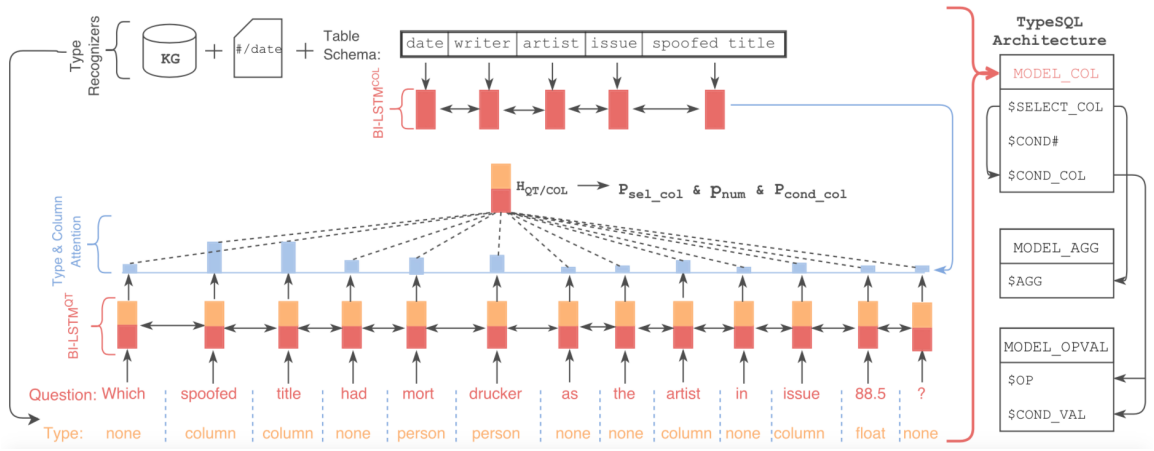

Figure 5.1: TYPESQL consists of three slot-filling models on the right. We only show MODEL_COL on the left for brevity. MODEL_AGG and MODEL_OPVAL have the similar pipelines.

introduce a knowledge-based type-aware text-to-SQL generator, TYPESQL. Based on the prior state-of-the-art SQLNet (Xu et al., 2017), TYPESQL employs a sketch-based approach and views the task as a slot filling problem (Figure 5.2). By grouping different slots in a reasonable way and capturing relationships between attributes, TYPESQL outperforms SQLNet by about 3.5% in half of the original training time.

Furthermore, natural language questions often contain rare entities and numbers specific to the underlying database. Some previous work (Agrawal and Srikant, 2003) already shows those words are crucial to many downstream tasks, such as infering column names and condition values in the SQL query. However, most of such key words lack accurate embeddings in popular pre-trained word embedding models. In order to solve this problem, TYPESQL assigns each word a type as an entity from either the knowledge graph, a column or a number. For example, for the question in Figure 5.1, we label "mort drucker" as PERSON according to our knowledge graph; "spoofed title," "artist" and "issue" as COLUMN since they are column names; and "88.5" as FLOAT. Incorporating this type information, TYPESQL further improves the state-of-the-art performance by about another 2% on the WikiSQL dataset, resulting in a final 5.5% improvement in total.

Moreover, most previous work assumes that user queries contain exact column names

and entries. However, it is unrealistic that users always formulate their questions with exact column names and string entries in the table. To tackle this issue, when scaleability and privacy are not of a concern, the system needs to search databases to better understand what the user is querying. Our content-sensitive model TYPESQL + TC gains roughly 9% improvement compared to the content-insensitive model, and outperforms the previous content-sensitive model by 17.5%.

5.2 Related Work

Semantic parsing maps natural language to meaningful executable programs. The programs could be a range of representations such as logic forms (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Wong and Mooney, 2007; Das et al., 2010; Liang et al., 2011; Banarescu et al., 2013; Artzi and Zettlemoyer, 2013; Reddy et al., 2014; Berant and Liang, 2014; Pasupat and Liang, 2015). As a sub-task of semantic parsing, the text-to-SQL problem has been studied for decades (Warren and Pereira, 1982; Popescu et al., 2003, 2004; Li et al., 2006; Giordani and Moschitti, 2012; Wang et al., 2017b). The methods of the Database community (Li and Jagadish, 2014; Yaghmazadeh et al., 2017) involve more hand feature engineering and user interactions with the systems.

In this work, we focus on recent neural network based approaches (Yin et al., 2016; Zhong et al., 2017; Xu et al., 2017; Wang et al., 2017a; Iyer et al., 2017). Dong and Lapata (2016) introduce a sequence-to-sequence approach to converting text to logical forms. Most of previous work focus on specific table schemas, which means they use a single database in both train and test. Thus, they don't generalize to new databases. Zhong et al. (2017) publish the WikiSQL dataset and propose a sequence-to-sequence model with reinforcement learning to generate SQL queries. In the problem definition of the WikiSQL task, the databases in the test set do not appear in the train and development sets. Also, the task needs to take different table schemas into account. Xu et al. (2017) further improve the results by

SELECT $AGG $SELECT_COL
WHERE $COND_COL $OP $COND_VAL (**AND** $COND_COL $OP $COND_VAL)*

Figure 5.2: SQL Sketch. The tokens starting with "$" are slots to fill. "*" indicates zero or more **AND** clauses.

using a SQL sketch based approach employing a sequence-to-set model.

5.3 Methodology

Like SQLNet, we employ a sketch-based approach and format the task as a slot filling problem. Figure 5.2 shows the SQL sketch. Our model needs to predict all slots that begin with $ in Figure 5.2.

Figure 5.1 illustrates the architecture of TYPESQL on the right and a detailed overview of one of three main models MODEL_COL on the left. We first preprocess question inputs by type recognition (Section 5.3.1). Then we use two bi-directional LSTMs to encode words in the question with their types and the column names separately (Section 5.3.2). The output hidden states of LSTMs are then used to predict the values for the slots in the SQL sketch (Section 5.3.3).

5.3.1 Type Recognition for Input Preprocessing

In order to create one-to-one type input for each question, we, first, tokenize each question into n-grams of length 2 to 6, and use them to search over the table schema and label any column name appears in the question as COLUMN. Then, we assign numbers and dates in the question into four self-explanatory categories: INTEGER, FLOAT, DATE, and YEAR. To identify named entities, we search for five types of entities: PERSON, PLACE, COUNTRY, ORGANIZATION, and SPORT, on Freebase[32] using grams as keyword queries. The five categories cover a majority of entities in the dataset. Thus, we do not use other

32. https://developers.google.com/freebase/

entity types provided by Freebase. Domain-specific knowledge graphs can be used for other applications.

In the case where the content of databases is available, we match words in the question with both the table schema and the content and labels of the columns as COLUMN and match the entry values as the corresponding column names. For example, the type in the Figure 5.1 would be [none, column, column, none, artist, artist, none, none, column, none, column, issue, none] in this case. Other parts in the Figure 5.1 keep the same as the content-insensitive approach.

5.3.2 Input Encoder

As shown in the Figure 5.1, our input encoder consists of two bi-directional LSTMs, BI-LSTM$^{\text{QT}}$ and BI-LSTM$^{\text{COL}}$. To encode word and type pairs of the question, we concatenate embeddings of words and their corresponding types and input them to BI-LSTM$^{\text{QT}}$. Then the output hidden states are $\mathbf{H}_{\text{QT}}$ and $\mathbf{H}_{\text{COL}}$, respectively.

For encoding column names, SQLNet runs a bi-directional LSTM over each column name. We first average the embeddings of words in the column name. Then, we run a single BI-LSTM$^{\text{COL}}$ between column names. This encoding method improves the result by 1.5% and cuts the training time by half. Even though the order of column names does not matter, we attribute this improvement to the fact that the LSTM can capture their occurrences and relationships.

5.3.3 Slot-Filling Model

Next, we predict values for the slots in the SQL sketch. For the slots in Figure 5.2, SQLNet has a separate model for each of them which do not share their trainable parameters. This creates five models for the five slots and one model for `$COND#` (12 BI-LSTMs in total). However, since the predict procedures of `$SELECT_COL`, `$COND_COL`, and `$COND#` are similar, we combine them into a single model. Additionally, `$COND_COL` depends on the

output of $SELECT_COL, which reduces errors of predicting the same column in these two slots $COND_COL Moreover, we group $OP and $COND_VAL together because both depend on the outputs of $COND_COL. Furthermore, we use one model for $AGG because we notice that the $AGG model converges much faster and suffers from overfitting when combined with other models. Finally, TYPESQL consists of three models (Figure 5.1 right):

- MODEL_COL for $SELECT_COL, $COND# and $COND_COL
- MODEL_AGG for $AGG
- MODEL_OPVAL for $OP and $COND_VAL

where the parameters of BI-LSTM$^{\text{QT}}$ and BI-LSTM$^{\text{COL}}$ are shared in each model (6 BI-LSTMs in total).

Since all three models use the same way to compute the weighted question and type representation $\mathbf{H}_{\text{QT/COL}}$ using the column attention mechanism proposed in SQLNet, we first introduce the following step in all three models:

$$\alpha_{\text{QT/COL}} = \textbf{softmax}(\mathbf{H}_{\text{COL}}\mathbf{W}_{ct}\mathbf{H}_{\text{QT}}^{\top})$$
$$\mathbf{H}_{\text{QT/COL}} = \alpha_{\text{QT/COL}}\mathbf{H}_{\text{QT}}$$

where **softmax** applies the softmax operator over each row of the input matrix, $\alpha_{\text{QT/COL}}$ is a matrix of attention scores, and $\mathbf{H}_{\text{QT/COL}}$ is the weighted question and type representation. In our equations, we use $\mathbf{W}$ and $\mathbf{V}$ to represent all trainable parameter matrices and vectors, respectively.

MODEL_COL-$SELECT_COL $\mathbf{H}_{\text{QT/COL}}$ is used to predict the column name in the $SELECT_COL:

$$s = \mathbf{V}^{sel}\mathbf{tanh}(\mathbf{W}_{c}^{sel}\mathbf{H}_{\text{COL}}^{\top} + \mathbf{W}_{qt}^{sel}\mathbf{H}_{\text{QT/COL}}^{\top})$$

$$P_{sel\_col} = \mathbf{softmax}(s)$$

	Dev			Test		
	Acc_{lf}	Acc_{qm}	Acc_{ex}	Acc_{lf}	Acc_{qm}	Acc_{ex}
Content Insensitive						
(Dong and Lapata, 2016)	23.3%	-	37.0%	23.4%	-	35.9%
Augmented Pointer Network (Zhong et al., 2017)	44.1%	-	53.8%	42.8%	-	52.8%
Seq2SQL (Zhong et al., 2017)	49.5%	-	60.8%	48.3%	-	59.4%
SQLNet (Xu et al., 2017)	-	63.2%	69.8%	-	61.3%	68.0%
TypeSQL w/o type-awareness (ours)	-	66.5%	72.8%	-	64.9%	71.7%
TypeSQL (ours)	-	**68.0%**	**74.5%**	-	**66.7%**	**73.5%**
Content Sensitive						
(Wang et al., 2017a)	59.6%	-	65.2%	59.5%	-	65.1%
TypeSQL+TC (ours)	-	**79.2%**	**85.5%**	-	**75.4%**	**82.6%**

Table 5.1: Overall results on WikiSQL. Acc_{lf}, Acc_{qm}, and Acc_{ex} denote the accuracies of exact string, canonical representation, and execute result matches between the synthesized SQL with the ground truth respectively. The top six results are content-insensitive, which means only the question and table schema are used as inputs. The bottom two are content-sensitive, where the models use the question, the table schema, and the content of databases.

	Dev			Test		
	Acc_{agg}	Acc_{sel}	$\text{Acc}_{\text{where}}$	Acc_{agg}	Acc_{sel}	$\text{Acc}_{\text{where}}$
Seq2SQL (Zhong et al., 2017)	90.0%	89.6%	62.1%	90.1%	88.9%	60.2%
SQLNet (Xu et al., 2017)	90.1%	91.5%	74.1%	90.3%	90.9%	71.9%
TypeSQL (ours)	90.3%	**93.1%**	**78.5%**	90.5%	**92.2%**	**77.8%**
TypeSQL+TC (ours)	90.3%	**93.5%**	**92.8%**	90.5%	**92.1%**	**87.9%**

Table 5.2: Breakdown results on WikiSQL. Acc_{agg}, Acc_{sel}, and $\text{Acc}_{\text{where}}$ are the accuracies of canonical representation matches on AGGREGATOR, SELECT COLUMN, and WHERE clauses between the synthesized SQL and the ground truth respectively.

MODEL_COL-$COND# Unlike SQLNet, we compute number of conditions in the WHERE in a simpler way:

$$P_{num} = \textbf{softmax}\left(\mathbf{V}^{num}\textbf{tanh}(\mathbf{W}_{qt}^{num}\sum_i \mathbf{H}_{\text{QT/COL}_i}^{\top})\right)$$

We set the maximum number of conditions to 4.

MODEL_COL-$COND_COL We find that SQLNet often selects the same column name in the $COND_COL as $SELECT_COL, which is incorrect in most cases. To avoid this problem, we pass the weighted sum of question and type hidden states conditioned on the column chosen in $SELECT_COL $\mathbf{H}_{\text{QT/SCOL}}$ (expended as the same shape of $\mathbf{H}_{\text{QT/COL}}$) to the prediction:

$$c = \mathbf{V}^{col}\textbf{tanh}(\mathbf{W}_{c}^{col}\mathbf{H}_{\text{COL}}^{\top} + \mathbf{W}_{qt}^{col}\mathbf{H}_{\text{QT/COL}}^{\top} + \mathbf{W}_{qt}^{scol}\mathbf{H}_{\text{QT/SCOL}}^{\top})$$

$$P_{cond\_col} = \textbf{softmax}(c)$$

MODEL_AGG-$AGG Given the weighted sum of question and type hidden states conditioned on the column chosen in $SELECT_COL $\mathbf{H}_{\text{QT/SCOL}}$, $AGG is chosen from {NULL, MAX, MIN, COUNT, SUM, AVG} in the same way as SQLNet:

$$P_{agg} = \textbf{softmax}\left(\mathbf{V}^{agg}\textbf{tanh}(\mathbf{W}_{qt}^{agg}\mathbf{H}_{\text{QT/SCOL}}^{\top})\right)$$

MODEL_OPVAL-$OP For each predicted condition column, we choose a $OP from $\{=, >, <\}$ by:

$$P_{op} = \textbf{softmax}\left(\mathbf{W}_{t}^{op}\textbf{tanh}(\mathbf{W}_{c}^{op}\mathbf{H}_{\text{COL}}^{\top} + \mathbf{W}_{qt}^{op}\mathbf{H}_{\text{QT/COL}}^{\top})\right)$$

MODEL_OPVAL-$COND_VAL Then, we need to generate a substring from the question for each predicted column. As in SQLNet, a bi-directional LSTM is used for the encoder. It employs a pointer network (Vinyals et al., 2015) to compute the distribution of the next token in the decoder. In particular, the probability of selecting the i-th token w_i in the natural language question as the next token in the substring is computed as:

$$v = \mathbf{V}_t^{val}\textbf{tanh}(\mathbf{W}_{qt}^{val}\mathbf{H}_{\text{QT}}^{i} + \mathbf{W}_{c}^{val}\mathbf{H}_{\text{COL}} + \mathbf{W}_{h}^{val}\mathbf{h})$$

$$P_{cond\_val} = \textbf{softmax}(v)$$

where $\mathbf{h}$ is the hidden state of the previously generated token. The generation process continues until the $\langle\texttt{END}\rangle$ token is the most probable next token of the substring.

5.4 Experiments

Dataset We use the WikiSQL dataset (Zhong et al., 2017), a collection of 87,673 examples of questions, queries, and database tables built from 26,521 tables. It provides train/dev/test splits such that each table is only in one split. This requires model to generalize to not only new questions but new table schemas as well.

Implementation Details We implement our model based on SQLNet (Xu et al., 2017) in PyTorch (Paszke et al., 2017). We concatenate pre-trained Glove (Pennington et al., 2014) and paraphrase (Wieting and Gimpel, 2017) embeddings. The dimensions and dropout rates of all hidden layers are set to 120 and 0.3 respectively. We use Adam (Kingma and Ba, 2015) with the default hyperparameters for optimization. The batch size is set to 64. The same loss functions in (Xu et al., 2017) are used. Our code is available at https://github.com/taoyds/typesql.

Results and Discussion Table 5.1 shows the main results on the WikiSQL task. We compare our work with previous results using the three evaluation metrics used in (Xu et al., 2017). Table 5.2 provides the breakdown results on AGGREGATION, SELECTION, and WHERE clauses.

Without looking at the content of databases, our model outperforms the previous best work by 5.5% on execute accuracy. According to Table 5.2, TYPESQL improves the accuracy of SELECT by 1.3% and WHERE clause by 5.9%. By encoding column names and grouping model components in a simpler but reasonable way, TYPESQL achieves a much higher result on the most challenging sub-task WHERE clause. Also, the further improvement of integrating word types shows that TYPESQL could encode the rare entities and numbers in a better way.

Also, if complete access to the database is allowed, TYPESQL can achieve 82.6% on execute accuracy, and improves the performance of the previous content-aware system by 17.5%. Although (Zhong et al., 2017) enforced some limitations when creating the WikiSQL dataset, there are still many questions that do not have any column name and entity indicator. This makes generating the right SQLs without searching the database content in such cases impossible. This is not a critical problem for WikiSQL but is so for most real-world tasks.

5.5 Summary

We propose TYPESQL for text-to-SQL which views the problem as a slot filling task and uses type information to better understand rare entities and numbers in the input. TYPESQL can use the database content to better understand the user query if it is not well-formed. TYPESQL significantly improves upon the previous state-of-the-art on the WikiSQL dataset.

Although, unlike most of the previous work, the WikiSQL task requires model to generalize to new databases, the dataset does not cover some important SQL operators such as JOIN and GROUP BY. This limits the generalization of the task to other SQL

components. In the future, we plan to advance this work by exploring other more complex datasets under the database-split setting. In this way, we can study the performance of a generalized model on a more realistic text-to-SQL task which includes many complex SQL and different databases.

5.6 Appendices

Implementation Details We implement our model based on SQLNet (Xu et al., 2017) in PyTorch (Paszke et al., 2017). We concatenate pre-trained Glove embeddings (Pennington et al., 2014) and paraphrase embeddings (Wieting and Gimpel, 2017) together. We use Glove 300-dimensional embeddings to initialize embeddings of types, and apply linear transformations converting them to 100-dimension. We keep word embeddings untrainable but type embeddings trainable during training. For the words that have either embeddings, we initialize them with zero vectors. The dimensions and dropout rates of all hidden layers are set to 100 and 0.3 respectively. We use Adam (Kingma and Ba, 2015) with the default hyperparameters for optimization. The batch size is set to 64. The same loss functions in (Xu et al., 2017) are used.

Chapter 6

SyntaxSQLNet: Syntax Tree Networks for Complex and Cross-Domain Text-to-SQL Task

In this chapter we propose SyntaxSQLNet, a syntax tree network to address the *complex* and *cross-domain* text-to-SQL generation task. SyntaxSQLNet employs a SQL-specific syntax tree-based decoder with SQL generation path history and table-aware column attention encoders. We evaluate SyntaxSQLNet on the *Spider* text-to-SQL task, which contains databases with multiple tables and complex SQL queries with multiple SQL clauses and nested queries. We use a database split setting where databases in the test set are unseen during training. Experimental results show that SyntaxSQLNet can handle a significantly greater number of complex SQL examples than prior work, outperforming the previous state-of-the-art model by 7.3% in exact matching accuracy. We also show that SyntaxSQLNet can further improve the performance by an additional 7.5% using a cross-domain augmentation method, resulting in a 14.8% improvement in total. To our knowledge, we are the first to study this complex and cross-domain text-to-SQL task.

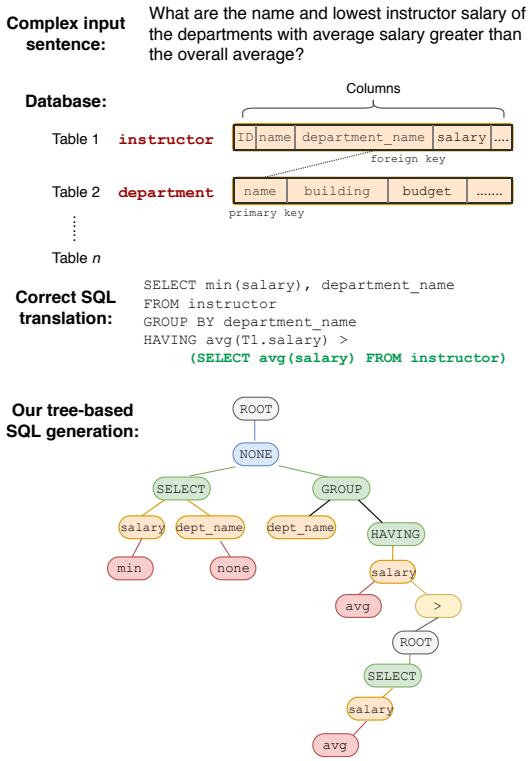

Figure 6.1: To address the complex text-to-SQL generation task, SyntaxSQLNet employs a tree-based SQL generator. For example, our model can systematically generate a nested query as illustrated above.

6.1 Introduction

In this chapter, we consider SPIDER, a new *complex* and *cross-domain* text-to-SQL task that requires models to generalize well to both new SQL queries and databases. The task cannot be solved easily without truly understanding the semantic meanings of the input questions.

We propose SyntaxSQLNet, a SQL specific syntax tree network to address the *Spider* task. Specifically, to generate complex SQL queries with multiple clauses, selections and sub-queries, we develop a syntax tree-based decoder with SQL generation path history. To make our model learn to generalize to new databases with new tables and columns, we also develop a table-aware column encoder. Our contributions are as follows:

- We propose SQL specific syntax tree networks for the complex and cross-domain text-to-SQL task, which is able to solve nested queries on unseen databases. We are the first to develop a methodology for this challenging semantic parsing task.

- We introduce a SQL specific syntax tree-based decoder with SQL path history and table-aware column attention encoders. Even with no hyperparameter tuning, our model can significantly outperform the previous best models, with an 7.3% boost in exact matching accuracy. Error analysis shows that our model is able to generalize, and solve much more complex (e.g., nested) queries in new databases than prior work.
- We also develop a cross-domain data augmentation method to generate more diverse training examples across databases, which further improves the exact matching accuracy by 7.5%. As a result, our model achieves 27.2% accuracy, a 14.8% total improvement compared with the previous best model.

6.2 Related Work

In this work, we focus on recent neural network-based approaches (Yin et al., 2016; Zhong et al., 2017; Xu et al., 2017; Wang et al., 2017a; Iyer et al., 2017; Gur et al., 2018; Suhr et al., 2018). Dong and Lapata (2016) introduce a sequence-to-sequence (seq2seq) approach to converting texts to logical forms. Most previous work focuses on a specific table schema. Zhong et al. (2017) publish the WikiSQL dataset and propose a seq2seq model with reinforcement learning to generate SQL queries. Xu et al. (2017) further improve the results on the WikiSQL task by using a SQL-sketch based approach employing a sequence-to-set model. Dong and Lapata (2018) propose a coarse-to-fine model which achieves the new state-of-the-art performances on several datasets including WikiSQL. Their model first generate a sketch of the target program. Then the model fills in missing details in the sketch.

Our syntax tree-based decoder is related to recent work that exploits syntax information for code generation tasks (Yin and Neubig, 2017; Rabinovich et al., 2017). Yin and Neubig (2017) introduce a neural model that transduces a natural language statement into an abstract syntax tree (AST). While they format the generation process as a seq2seq decoding of rules and tokens, our model uses a sequence-to-set module for each grammar component, and

calls them recursively to generate a SQL syntax tree. Similarly, Rabinovich et al. (2017) propose abstract syntax networks that use a collection of recursive modules for decoding. Our model differs from theirs in the following points. First, we exploit a SQL specific grammar instead of AST. AST-based models have to predict many non-terminal rules before predicting the terminal tokens, involving more steps. Whereas, our SQL-specific grammar enables direct prediction of SQL tokens. Second, our model uses different sequence-to-set modules to avoid the "ordering issue" (Xu et al., 2017) in many code generation tasks. Third, different from (Rabinovich et al., 2017), we pass a pre-order traversal of SQL decoding history to each module. This provides each module with important dependence information: e.g., if a SQL query has `GROUP BY`, it is very likely that the grouped column has appeared in `SELECT` too. Finally, instead of sharing parameters across different modules, we train each module separately, because the parameters of different modules could have different converge times.

In addition to the distinction in model design, our work differs from theirs in the data and task definition. They aim to develop general syntax model for code generation via abstract syntax trees. Instead, we are interested in solving the complex and cross-domain SQL query generation problem; this motivates us to take advantage of SQL specific syntax for decoding, which guides systematic generation of complex SQL queries.

6.3 Methodology

Similar to (Rabinovich et al., 2017), our model structures the decoder as a collection of recursive modules. However, as we discussed in the related work section, we make use of a SQL specific grammar to guide the decoding process, which allows us to take advantage of SQL queries' well-defined structure. Also, modules do not share any parameters so that we train each of them independently.

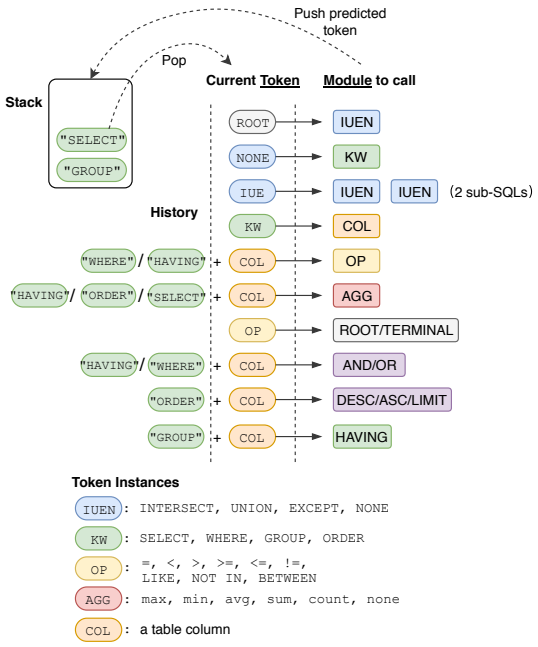

Figure 6.2: Our modules and SQL grammar used in decoding process. A round symbol represents a SQL tokens, a table column, etc. A square symbol indicates a module that predicts the next SQL token from its corresponding token instances with the same color.

6.3.1 Module Overview

Our model decomposes the SQL decoding process into 9 modules to handle the prediction of different SQL components such as keywords, operators, and columns. We provide the overview in this section and more details in later sections.

Figure 6.2 illustrates our modules and SQL grammar used in decoding process. A round symbol represents a SQL token, such as `SELECT`, `WHERE`, a table column, etc. A square symbol indicates a module that predicts the next SQL token from its corresponding token instances with the same color. Specifically, we have the following modules.

- **IUEN Module**, predicting `INTERSECT`, `UNION`, `EXCEPT`, and `NONE`, which determines if we need to call itself again to generate nested queries.
- **KW Module**, predicting keywords from `WHERE`, `GROUP BY`, and `ORDER BY`. All queries in our dataset have `SELECT`.
- **COL Module**, predicting table columns.
- **OP Module**, for $=$, $>$, $<$, $>=$, $<=$, $\neq$, `LIKE`, `NOT IN`, `IN`, `BETWEEN`.

- **AGG Module**, predicting aggregators from `MAX`, `MIN`, `SUM`, `COUNT`, `AVG`, and `NONE`.
- **Root/Terminal Module**, predicting the `ROOT` of a new subquery or terminal value. It also enables our model to generate nested queries.
- **AND/OR Module**, predicting the presence of AND or OR operator between two conditions.
- **DESC/ASC/LIMIT Module**, predicting the keywords associated with `ORDER BY`. It is invoked only when `ORDER BY` is predicted before.
- **HAVING Module**, predicting the presence of `HAVING` for `GROUP BY` clause. It is invoked only when `GROUP BY` is predicted earlier.

6.3.2 SQL Grammar

In order to structure our decoder to generate complex queries, we consider a SQL grammar. It determines which module to be invoked at each recursive decoding step. Figure 6.2 illustrates our SQL grammar. During decoding process, given the current SQL token and the SQL history (the tokens we have gone over to reach the current token), we determine which module to invoke, and predict the next SQL token to generate.

To invoke some modules such as HAVING and OP during decoding, we not only check the type of current token instance but also see whether the type of the previously decoded SQL token is `GROUP` for HAVING module, and `WHERE` or `HAVING` for OP module.

In the grammar, IUEN and Root/Terminal modules are able to generate `ROOT`, which can activate IUEN module again. In this way, our model can recursively generate nested subqueries, and can also predict two or more subqueries in queries that have `EXCEPT`, `INTERSECT`, and `UNION`.

6.3.3 Input Encoder

Our inputs of each module consist of three types of information: question, table schema, and current SQL decoding history path. We encode a question sentence by a bi-directional LSTM, $\mathrm{BiLSTM}^{\mathrm{Q}}$. We encode table schema and history path in the manners described below.

Table-Aware Column Representation

In order to generalize to new databases in testing, it is important to make our model learn to obtain necessary information from a database schema.

SQLNet (Xu et al., 2017) encodes this information by running different bi-directional LSTMs over words in each column name, whereas TypeSQL (Yu et al., 2018a) first obtains embedding for each column name by taking the average embedding of the words constituting the column name, and then runs a single biLSTM on the embeddings of all columns in a table. Yu et al. (2018c) show that the column encoding method of SQLNet outperforms that of TypeSQL in the database split setting, and the result reverses under the example split setting.

While SQLNet and TypeSQL only need the column names as WikiSQL dataset only contains one table per question-SQL pair, *Spider*'s databases contain multiple tables. To address this setting, we propose to use both table and column names to construct column embeddings.

Specifically, given a database, for each column, we first get the list for words in its table name, words in its column name, and the type information of the column (string, or number, primary/foreign key), as an initial input of the column. Next, like SQLNet, the table-aware column representation of the given column is computed as the final hidden state of a BiLSTM running on top of this sequence. This way, the encoding scheme can capture both the global (table names) and local (column names and types) information in the database schema to understand a natural language question in the context of the given

database.

We also experimented with a hierarchical table and column encoding, where we first obtain embedding for each table name and then incorporate that information into column encoding. But this encoding method did not perform as well.

SQL Decoding History

In addition to question and column information, we also pass the SQL query's current decoding history as an input to each module. This enables us to use the information of previous decoding states to predict the next SQL token. For example, in Figure 6.1, the COL module would be more likely to predict `salary` in the subquery by considering the path history which contains `salary` for `HAVING`, and `SELECT` in the main query.

In contract, each module in SQLNet does not consider the previous decoded SQL history. Hence, if directly applied to our recursive SQL decoding steps, each module would just predict the same output every time it is invoked. By passing the SQL history, each module is able to predict a different output according to the history every time it is called during the recursive SQL generation process. Also, the SQL history can improve the performance of each module on long and complex queries because the history helps the model capture the relations between clauses.

Predicted SQL history is used during test decoding. For training, we first traverse each node in the gold query tree in pre-order to generate gold SQL path history for each training example used in different modules.

Attention for Input Encoding

For each module, like SQLNet (Xu et al., 2017), we apply the attention mechanism to encode question representation. We also employs this technique on SQL path history encoding. The specific formulas used are described in the next section.

6.3.4 Module Details

Similarly to SQLNet, we employ a sketch-based approach for each module. We apply a sequence-to-set prediction framework introduced by (Xu et al., 2017), to avoid the order issue that happens in seq2seq based models for SQL generation. For example, in Figure 6.1, `SELECT salary, dept_name` is the same as `SELECT dept_name, salary`. The traditional seq2seq decoder generates each of them one by one in order; hence the model could get penalized even if the prediction and gold label are the same as sets. To avoid this problem, SQLNet predicts them together in one step so that their order does not affect the model's training process. For instance, in Figure 6.1, our model invokes the COL module to predict `salary` and `dept_name`, and push to stack at the same time.

However, SQLNet only covers pre-defined SQL sketches, and its modules do not pass information to one another. To resolve these problems, SyntaxSQLNet employs a syntax tree-based decoding method that recursively calls different modules based on a SQL grammar. Further, the history of generated SQL tokens is passed through modules, allowing SyntaxSQLNet to keep track of the recursive decoding steps.

We first describe how to compute the conditional embedding $\mathbf{H}_{1/2}$ of an embedding $\mathbf{H}_1$ given another embedding $\mathbf{H}_2$:

$$\mathbf{H}_{1/2} = \textbf{softmax}(\mathbf{H}_1 \mathbf{W} \mathbf{H}_2^\top)\mathbf{H}_1.$$

Here $\mathbf{W}$ is a trainable parameter. Moreover, we get a probability distribution from a given score matrix $\mathbf{U}$ by

$$\mathcal{P}(\mathbf{U}) = \textbf{softmax}\left(\mathbf{V}\textbf{tanh}(\mathbf{U})\right),$$

where $\mathbf{V}$ is a trainable parameter.

We denote the hidden states of LSTM on question embeddings, path history, and columns embeddings as $\mathbf{H}_{\text{Q}}$, $\mathbf{H}_{\text{HS}}$, and $\mathbf{H}_{\text{COL}}$ respectively. In addition, we denote the hidden states of LSTM on multiple keywords embeddings and keywords embeddings as $\mathbf{H}_{\text{MKW}}$ and $\mathbf{H}_{\text{KW}}$

respectively. Finally, we use $\mathbf{W}$ to denote trainable parameters that are not shared between modules. The output of each module is computed as follows:

IUEN Module In the IUEN module, since only one of the multiple keywords from $\{\texttt{INTERSECT}, \texttt{UNION}, \texttt{EXCEPT}, \texttt{NONE}\}$ will be used, we compute the probabilities by

$$P_{\text{IUEN}} = \mathcal{P}\left(\mathbf{W}_1 \mathbf{H}_{\text{Q/MKW}}^{\top} + \mathbf{W}_2 \mathbf{H}_{\text{HS/MKW}}^{\top} + \mathbf{W}_3 \mathbf{H}_{\text{MKW}}^{\top}\right)$$

KW Module In the KW module, we first predict the number of keywords in the SQL query and then predict the keywords from $\{\texttt{SELECT}, \texttt{WHERE}, \texttt{GROUP BY}, \texttt{ORDER BY}\}$.

$$P_{\text{KW}}^{\text{num}} = \mathcal{P}\left(\mathbf{W}_1^{\text{num}} {\mathbf{H}_{\text{Q/KW}}^{\text{num}}}^{\top} + \mathbf{W}_2^{\text{num}} {\mathbf{H}_{\text{HS/KW}}^{\text{num}}}^{\top}\right)$$
$$P_{\text{KW}}^{\text{val}} = \mathcal{P}\left(\mathbf{W}_1^{\text{val}} {\mathbf{H}_{\text{Q/KW}}^{\text{val}}}^{\top} + \mathbf{W}_2^{\text{val}} {\mathbf{H}_{\text{HS/KW}}^{\text{val}}}^{\top} + \mathbf{W}_3^{\text{val}} {\mathbf{H}_{\text{KW}}}^{\top}\right)$$

COL Module Similarly, in the COL module, we first predict the number of columns in the SQL query and then predict which ones to use.

$$P_{\text{COL}}^{\text{num}} = \mathcal{P}\left(\mathbf{W}_1^{\text{num}} {\mathbf{H}_{\text{Q/COL}}^{\text{num}}}^{\top} + \mathbf{W}_2^{\text{num}} {\mathbf{H}_{\text{HS/COL}}^{\text{num}}}^{\top}\right)$$
$$P_{\text{COL}}^{\text{val}} = \mathcal{P}\left(\mathbf{W}_1^{\text{val}} {\mathbf{H}_{\text{Q/COL}}^{\text{val}}}^{\top} + \mathbf{W}_2^{\text{val}} {\mathbf{H}_{\text{HS/COL}}^{\text{val}}}^{\top} + \mathbf{W}_3^{\text{val}} {\mathbf{H}_{\text{COL}}}^{\top}\right)$$

OP Module In the OP module, for each predicted column from the COL module that is in the `WHERE` clause, we first predict the number of operators on it then predict which operators to use from $\{=, >, <, >=, <=, \neq, \texttt{LIKE}, \texttt{NOT IN}, \texttt{IN}, \texttt{BETWEEN}\}$. We use $\mathbf{H}_{\text{CS}}$ to denote the embedding of one of the predicted columns from the COL module.

$$P_{\text{OP}}^{\text{num}} = \mathcal{P}\left(\mathbf{W}_1^{\text{num}} {\mathbf{H}_{\text{Q/CS}}^{\text{num}}}^{\top} + \mathbf{W}_2^{\text{num}} {\mathbf{H}_{\text{HS/CS}}^{\text{num}}}^{\top} + \mathbf{W}_3^{\text{num}} {\mathbf{H}_{\text{CS}}}^{\top}\right)$$
$$P_{\text{OP}}^{\text{val}} = \mathcal{P}\left(\mathbf{W}_1^{\text{val}} {\mathbf{H}_{\text{Q/CS}}^{\text{val}}}^{\top} + \mathbf{W}_2^{\text{val}} {\mathbf{H}_{\text{HS/CS}}^{\text{val}}}^{\top} + \mathbf{W}_3^{\text{val}} {\mathbf{H}_{\text{CS}}}^{\top}\right)$$

AGG Module In the AGG module, for each predicted column from the COL module, we first predict the number of aggregators on it then predict which aggregators to use from $\{\texttt{MAX}, \texttt{MIN}, \texttt{SUM}, \texttt{COUNT}, \texttt{AVG}, \texttt{NONE}\}$

$$P_{\text{AGG}}^{\text{num}} = \mathcal{P}\left(\mathbf{W}_1^{\text{num}} {\mathbf{H}_{\text{Q/CS}}^{\text{num}}}^{\top} + \mathbf{W}_2^{\text{num}} {\mathbf{H}_{\text{HS/CS}}^{\text{num}}}^{\top} + \mathbf{W}_3^{\text{num}} {\mathbf{H}_{\text{CS}}}^{\top}\right)$$
$$P_{\text{AGG}}^{\text{val}} = \mathcal{P}\left(\mathbf{W}_1^{\text{val}} {\mathbf{H}_{\text{Q/CS}}^{\text{val}}}^{\top} + \mathbf{W}_2^{\text{val}} {\mathbf{H}_{\text{HS/CS}}^{\text{val}}}^{\top} + \mathbf{W}_3^{\text{val}} {\mathbf{H}_{\text{CS}}}^{\top}\right)$$

Root/Terminal Module To predict nested subqueries, we add a module to predict if there is a new "ROOT" after an operator, which allows the model to decode queries recursively. For each predicted column from the COL module that is in the `WHERE` clause, we first call OP module, and then predict whether the next decoding step is a "ROOT" node or a value terminal node by

$$P_{\text{RT}} = \mathcal{P}\left(\mathbf{W}_1 \mathbf{H}_{\text{Q/CS}}^{\top} + \mathbf{W}_2 \mathbf{H}_{\text{HS/CS}}^{\top} + \mathbf{W}_3 \mathbf{H}_{\text{CS}}^{\top}\right)$$

AND/OR Module For each condition column predicted from the COL module with number bigger than 1, we predict from $\{\texttt{AND}, \texttt{OR}\}$ by

$$P_{\text{AO}} = \mathcal{P}\left(\mathbf{W}_1 \mathbf{H}_{\text{Q}}^{\top} + \mathbf{W}_2 \mathbf{H}_{\text{HS}}^{\top}\right)$$

DESC/ASC/LIMIT Module In this module, for each predicted column from the COL module that is in the `ORDER BY` clause, we predict
from $\{\texttt{DESC}, \texttt{ASC}, \texttt{ DESC LIMIT}, \texttt{ASC LIMIT}\}$ by

$$P_{\text{DAL}} = \mathcal{P}\left(\mathbf{W}_1 \mathbf{H}_{\text{Q/CS}}^{\top} + \mathbf{W}_2 \mathbf{H}_{\text{HS/CS}}^{\top} + \mathbf{W}_3 \mathbf{H}_{\text{CS}}^{\top}\right)$$

HAVING Module In the HAVING module, for each predicted column from the COL module that is in the `GROUP BY` clause, we predict whether it is in the `HAVING` clause by

$$P_{\text{HAVING}} = \mathcal{P}\left(\mathbf{W}_1\mathbf{H}_{\text{Q/CS}}^{\top} + \mathbf{W}_2\mathbf{H}_{\text{HS/CS}}^{\top} + \mathbf{W}_3\mathbf{H}_{\text{CS}}^{\top}\right)$$

6.3.5 Recursive SQL Generation

The SQL generation process is a process of activating different modules recursively. As illustrated in Figure 6.2, we employ a stack to organize our decoding process. At each decoding step, we pop one SQL token instance from the stack, and invoke a module based on the grammar to predict the next token instance, and then push the predicted instance into the stack. The decoding process continues until the stack is empty.

More specifically, we initialize a stack with only `ROOT` at the first decoding step. At the next step, the stack pops `ROOT`. As illustrated in Figure 6.2, `ROOT` actives the `IUEN` module to predict if there is `EXCEPT`, `INTERSECT` or `UNION`. If so, there are two subqueries to be generated in the next step. If the model predicts `NONE` instead, it will be pushed into the stack. The stack pops `NONE` at next step. For example, in Figure 6.2, the current popped token is `SELECT`, which is a instance of keyword (KW) type. It calls the COL module to predict a column name, which will be pushed to the stack.

6.3.6 Data Augmentation

Even though *Spider* already has a significantly larger number of complex queries than existing datasets, the number of training examples for some complex SQL components is still limited. A widely used way is to conduct data augmentation to generate more training examples automatically. Many studies (Berant and Liang, 2014; Iyer et al., 2017; Su and Yan, 2017) have shown that data augmentation can bring significant improvement in performance.

In prior work, data augmentation was typically performed within a single domain dataset. We propose a cross-domain data augmentation method to expand our training data for

complex queries. Cross-domain data augmentation is more difficult than the in-domain setting because question-program pairs tend to have domain specific words and phrases.

To tackle this issue, we first create a list of universal patterns for question-SQL pairs, based on the human labeled pairs from all the different training databases in *Spider*. To do so, we use a script to remove (and later fill in) all the table/column names and value tokens in the labeled question-SQL pairs, and then group together the same SQL query patterns. Consequently, each SQL query pattern has a list of about 5-20 corresponding questions. In our task, we want to generate more complex training examples. Thus, we filter out simple SQL query patterns by measuring the length and the number of SQL keywords used. We obtain about 280 different complex SQL query patterns from over 4,000 SQL labels in the train set of our corpus. We then select the 50 most frequent complex SQL patterns that contain multiple SQL components and nested subqueries.

After this, we manually edit the selected SQL patterns and their corresponding list of questions to make sure that the table/column/value slots in the questions have one-to-one correspondence to the slots in the corresponding SQL query. For each slot, we also add column type or table information. Thus, for example, columns with string type do not appear in the column slot with integer type during data augmentation (i.e., slot refilling) process. In this way, our question-SQL patterns are generated based on existing human labeled examples, which ensures that the generated training examples are natural.

Once we have the one-to-one slot mapping between questions and SQL queries, we apply a script that takes a new database schema with type information and generates new question-SQL examples by filling empty slots. Specifically, for each table in WikiSQL, we first randomly sample 10 question-SQL patterns. We randomly sample columns from the database schema based on its type: for example, if the slot type in the pattern is “number”, and then we only sample from columns with “real” type in the current table. We then refill the slots in both the question and SQL query with the selected column names. Similarly, we also refill table/value slots.

By this data augmentation method, we finally obtain about 98,000 question and SQL pairs using some WikiSQL databases with one single table.

6.4 Experiments

6.4.1 Dataset

In our experiments, we use *Spider* (Yu et al., 2018c), a new large-scale human annotated text-to-SQL dataset with complex SQL queries and cross-domain databases. We follow (Yu et al., 2018c), and use 146, 20, 40 databases for train, development, test, respectively (randomly split). We also include the question-SQL pair examples generated by our data augmentation method in some experiments.

6.4.2 Metrics

We evaluate our model using SQL Component Matching and Exact Matching proposed by (Yu et al., 2018c). To compute the component matching scores, Yu et al. (2018c) first decompose predicted queries on SQL clauses including `SELECT`, `WHERE`, `GROUP BY`, `ORDER BY`, and `KEYWORDS` separately. After that, they evaluate each predicted clause and the ground truth as bags of several sub-components, and check whether or not these two sets of components match exactly. Exact matching score is 1 if the model predicts all clauses correctly for a given example.

To better understand model performance on different queries, (Yu et al., 2018c) divide SQL queries into 4 levels: easy, medium, hard, extra hard. The definition of difficulty is based on the number of SQL components, selections, and conditions.

6.4.3 Experimental Settings

Our model is implemented in PyTorch (Paszke et al., 2017). We build each module based on the TypeSQL (Yu et al., 2018a) implementation. We use fixed, pre-trained GloVe (Pennington et al., 2014) embeddings for question, SQL history, and schema tokens. For each experiment, the dimension and dropout rate of all hidden layers is set to 120 and 0.3 respectively. We use Adam (Kingma and Ba, 2015) with the default hyperparameters for optimization, with a batch size of 64. The same loss functions in (Xu et al., 2017) are used for each module. The code is available on https://github.com/taoyds/syntaxsql.

Method	Test					Dev
	Easy	Medium	Hard	Extra Hard	All	All
Seq2Seq	11.9%	1.9%	1.3%	0.5%	3.7%	1.9%
Seq2Seq+Attention	14.9%	2.5%	2.0%	1.1%	4.8%	1.8%
Seq2Seq+Copying	15.4%	3.4%	2.0%	1.1%	5.3%	4.1%
SQLNet	26.2%	12.6%	6.6%	1.3%	12.4%	10.9%
TypeSQL	19.6%	7.6%	3.8%	0.8%	8.2%	8.0%
SyntaxSQLNet	**48.0%**	**27.0%**	**24.3%**	4.6%	**27.2%**	**24.8%**
-augment	38.6%	17.6%	16.3%	**4.9%**	19.7%	18.9%
-table -augment	37.5%	13.5%	12.4%	1.3%	16.4%	15.9%
-history -table -augment	18.1%	7.0%	0.2%	0.0%	6.8%	6.1%

Table 6.1: Accuracy of Exact Matching on SQL queries with different hardness levels.

Method	SELECT	WHERE	GROUP BY	ORDER BY	KEYWORDS
Seq2Seq	13.0%	1.5%	3.3%	5.3%	8.7%
Seq2Seq+Attention	13.6%	3.1%	3.6%	9.9%	9.9%
Seq2Seq+Copying	12.0%	3.1%	5.3%	5.8%	7.3%
SQLNet	44.5%	19.8%	29.5%	48.8%	64.0%
TypeSQL	36.4%	16.0%	17.2%	47.7%	66.2%
SyntaxSQLNet	**62.5%**	**34.8%**	**55.6%**	**60.9%**	69.6%
-augment	53.9%	24.5%	44.4%	49.5%	**71.3%**
-table -augment	48.9%	20.1%	36.3%	46.8%	69.7%
-history -table -augment	26.7%	14.6%	11.8%	34.9%	64.6%

Table 6.2: F1 scores of Component Matching on all SQL queries on Test set.

6.5 Results and Discussion

Table 6.1 presents SyntaxSQLNet's dev and test results compared to previous state-of-the-art models on the *Spider* dataset with database splitting. Our model with SQL history and data augmentation achieves 27.2% exact matching on all SQL queries, which is about 15% absolute increase compared to the previous best models, SQLNet and TypeSQL.

6.5.1 Comparison to Existing Methods

Even though our individual modules are similar to SQLNet and TypeSQL, our syntax-aware decoder allows the modules to generate complex SQL queries in a recursive manner based on the SQL grammar. In addition, by incorporating the SQL decoding history into modules during the decoding process, SyntaxSQL achieves a significant gain in exact matching for queries of all hardness levels. Specifically, even without our data augmentation technique, SyntaxSQLNet outperforms the previous best, SQLNet, by 7.3%. This result suggests that the syntax and history information is beneficial for this complex text-to-SQL task.

Moreover, the tree-based decoder enables SyntaxSQLNet to systematically generate nested queries, boosting the performance for Hard/Extra Hard. As Table 6.1 shows, SyntaxSQLNet achieves particularly high scores 24.3% and 4.6% for Hard and Extra Hard, which contain nested queries. The Seq2Seq models suffer from generating ungrammatical queries, yielding very low exact matching accuracy on Hard and Extra Hard SQL queries. In contrast, our model generates valid SQL queries by enforcing the syntax.

For the detailed component matching results in Table 6.2, our model consistently outperforms other previous work by significant margins. Specifically, our model improve F1 score for most of the SQL components by more than 10%.

6.5.2 Ablation Study

In order to understand the techniques that are responsible for the performance of our model, we perform an ablation study where we remove one of the proposed techniques from our model at a time. The exact match scores are shown in the same tables as other previous models.

Data Augmentation Our model's exact matching performance on all queries drops 7.5% by excluding data augmentation technique. This drop is particularly large for `GROUP BY` and `ORDER BY` components (Table 6.2), for which the original *Spider* dataset has a relatively small number of training examples. Our cross-domain data augmentation technique provides significantly more examples for column prediction (especially under `GROUP BY` and `ORDER BY` clauses), which greatly benefits the overall model performance.

Column Encoding To see how our table-aware column encoding affects performance of our model, we also report the model's result without using table information for our column encoding. After excluding the table embedding from column embeddings, the test performance further goes down by 3.3%. This drop is especially large for Medium/Hard SQL queries, where the correct column prediction is a key. Additionally, in Table 6.2, the model's performance on `GROUP BY` component decreases dramatically because it is hard to predict group-by columns correctly without table information (e.g. multiple different tables may have a column of the same name "id" in the database). This result shows that the table-aware encoding is important to predict the correct columns in unseen, complex databases (with many foreign keys).

SQL Decoding History In order to gain more insight into how our SQL decoding history addresses complex SQL, we report our model's performance without SQL path history. As shown in the Table 6.1, the model's performance drops about 9.6% on exacting matching metric without considering the previous decoding states in each decoding state. More

importantly, its performance on hard and extra hard SQL queries decreases to 0%. This indicates that our model is able to predict nested queries thanks to the SQL decoding history.

6.5.3 Error Analysis and Future Work

The most common errors are from column prediction. Future work may include developing a database schema encoder that can capture relationships among columns and foreign keys in the database more effectively. Other common errors include incorrect prediction of SQL skeleton structures, aggregators and operators.

There are also a few limitations in our model. For example, SyntaxSQLNet first predicts all the column names in the SQL query, and then chooses tables to generate the `FROM` clause based on the selected columns. Suppose the natural language input is "return the stadium name and the number of concerts held in each stadium." The SQL query predicted by SyntaxSQLNet is

```
SELECT count(*), name FROM stadium GROUP BY stadium_id
```

While the correct answer is

```
SELECT T2.name, count(*) FROM concert AS T1 JOIN stadium AS
T2 ON T1.stadium_id = T2.stadium_id GROUP BY T1.stadium_id
```

Even though SyntaxSQLNet predicts all column names and keywords correctly, its deterministic `FROM` clause generation method fails to join tables ("concert" and "stadium" in this case) together. One possible solution is to predict table names in the `FROM` clause by considering the relations among tables in the database.

6.6 Summary

In this chapter, we presented a syntax tree-based model to address complex and cross-domain text-to-SQL task. Utilizing a SQL specific syntax decoder, as well as SQL path history

and table-aware column attention encoders, our model outperforms previous work by a significant margin. The ablation study demonstrates that our proposed techniques are able to predict nested, complex SQL queries correctly even for unseen databases.

Part III

Language Model Pre-Training

Chapter 7

GraPPa: Grammar-Augmented Pre-Training for Table Semantic Parsing

In this chapter, we present GRAPPA, an effective pre-training approach for table semantic parsing that learns a compositional inductive bias in the joint representations of textual and tabular data. We construct synthetic question-SQL pairs over high-quality tables via a synchronous context-free grammar (SCFG). We pre-train GRAPPA on the synthetic data to inject important structural properties commonly found in table semantic parsing into the pre-training language model. To maintain the model's ability to represent real-world data, we also include masked language modeling (MLM) on several existing table-and-language datasets to regularize our pre-training process. Our proposed pre-training strategy is very data-efficient. When incorporated with strong base semantic parsers, GRAPPA achieves new state-of-the-art results on four popular fully supervised and weakly supervised table semantic parsing tasks.

7.1 Introduction

Tabular data serve as important information source for human decision makers in many domains, such as finance, health care, retail and so on. While tabular data can be efficiently

accessed via the structured query language (SQL), a natural language interface allows such data to be more accessible for a wider range of non-technical users. As a result, table semantic parsing that maps natural language queries over tabular data to formal programs has drawn significant attention in recent years.

Recent pre-trained language models (LMs) such as BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019b) achieve tremendous success on a spectrum of natural language processing tasks, including semantic parsing (Zettlemoyer and Collins, 2005; Zhong et al., 2017; Yu et al., 2018c). These advances have shifted the focus from building domain-specific semantic parsers (Zettlemoyer and Collins, 2005; Artzi and Zettlemoyer, 2013; Berant and Liang, 2014; Li and Jagadish, 2014) to cross-domain semantic parsing (Zhong et al., 2017; Yu et al., 2018c; Herzig and Berant, 2018; Dong and Lapata, 2018; Wang et al., 2020; Lin et al., 2020).

Despite such significant gains, the overall performance on complex benchmarks such SPIDER (Yu et al., 2018c) and WIKITABLEQUESTIONS are still limited, even when integrating representations of current pre-trained language models. As such tasks requires generalization to new databases/tables and more complex programs (e.g., SQL), we hypothesize that current pre-trained language models are not sufficient for such tasks. First, language models pre-trained using unstructured text data such as Wikipedia and Book Corpus are exposed to a significant domain shift when directly applied to table semantic parsing, where jointly modeling the relation between utterances and structural tables is crucial. Second, conventional pre-training objectives do not consider the underlying compositionality of data (e.g., questions and SQLs) from table semantic parsing. To close this gap, we seek to learn contextual representations jointly from structured tabular data and unstructured natural language sentences, with objectives oriented towards table semantic parsing.

In this chapter, we propose a novel grammar-augmented pre-training framework for table semantic parsing (GRAPPA). Inspired by previous work on data synthesis for semantic parsing (Berant and Liang, 2014; Wang et al., 2015; Jia and Liang, 2016; Herzig and Berant,

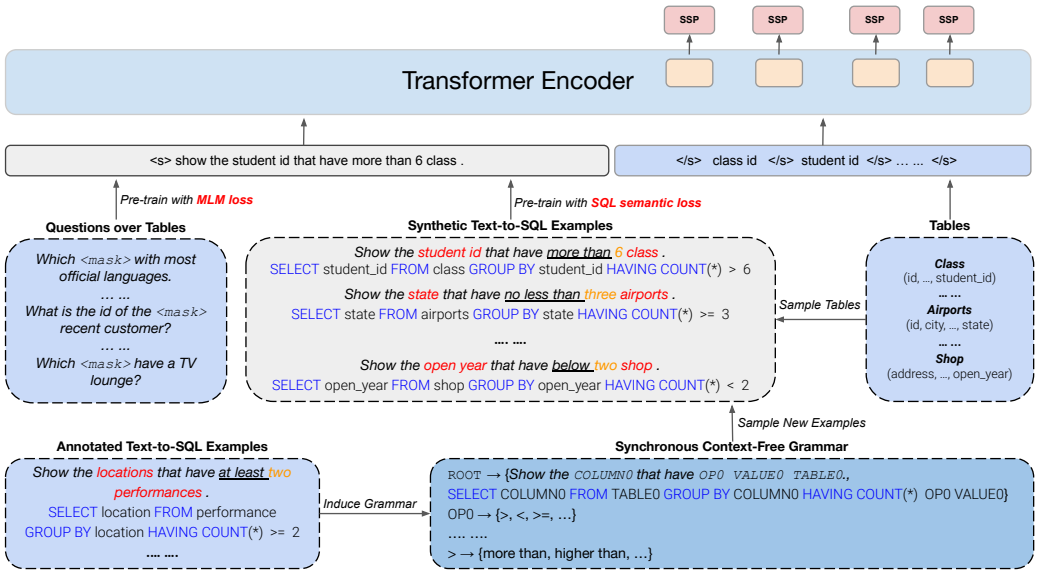

Figure 7.1: An overview of GRAPPA pre-training approach. We first induce a SCFG given some examples in SPIDER. We then sample from this grammar given a large amount of tables to generate new synthetic examples. Finally, GRAPPA is pre-trained on the synthetic data using SQL semantic loss and a small amount of table related utterances using MLM loss.

2018; Andreas, 2020), we induce a synchronous context-free grammar (SCFG) specific to mapping natural language to SQL queries from existing text-to-SQL datasets, which covers most commonly used question-SQL patterns. As shown in Figure 7.1, from a text-to-SQL example we can create a question-SQL template by abstracting over mentions of schema components (tables and fields), values, and SQL operations. By executing this template on randomly selected tables we can create a large number of synthetic question-SQL pairs. We train GRAPPA on these synthetic question-SQL pairs and their corresponding tables using a novel text-schema linking objective that predicts the syntactic role of a table column in the SQL for each pair. This way we encourage the model to identify table schema components that can be grounded to logical form constituents, which is critical for most table semantic parsing tasks.

To prevent overfitting to the synthetic data, we include the masked-language modelling (MLM) loss on several large-scale, high-quality table-and-language datasets and carefully balance between preserving the original natural language representations and enforcing the

compositional interpolation through our synthetic data. We pre-train GRAPPA using 475k synthetic examples and 391.5k examples from existing table-and-language datasets. Our approach dramatically reduces the training time and GPU cost.

We evaluate on four popular semantic parsing benchmarks in both fully supervised and weakly supervised settings. GRAPPA consistently achieves new state-of-the-art results on all of them, significantly outperforming all previously reported results.

7.2 Related Work

Textual-tabular data understanding Real-world data exist in both structured and unstructured forms. Recently the field has witnessed a surge of interest in joint textual-tabular data understanding problems, such as table semantic parsing (Zhong et al., 2017; Yu et al., 2018c), question answering (Pasupat and Liang, 2015; Chen et al., 2020), retrieval (Zhang et al., 2019c), fact-checking (Chen et al., 2019) and summarization (Parikh et al., 2020; Radev et al., 2021). While most work focus on single tables, often obtained from the Web, some have extended modeling to more complex structures such as relational databases (Finegan-Dollak et al., 2018; Yu et al., 2018c; Wang et al., 2020). All of these tasks can benefit from better representation of the input text and different components of the table, and most importantly, an effective contextualization across the two modalities. Our work aims at obtaining high-quality cross-modal representation via pre-training to potentially benefit all downstream tasks.

Data augmentation for semantic parsing Our work was inspired by existing work on data augmentation for semantic parsing (Berant and Liang, 2014; Wang et al., 2015; Jia and Liang, 2016; Iyer et al., 2017; Yu et al., 2018b). Berant and Liang (2014) employed a rule-based approach to generate canonical natural language utterances given a logical form. A paraphrasing model was then used to choose the canonical utterance that best paraphrases the input and to output the corresponding logical form. In contrast, Jia and Liang (2016)

used prior knowledge in structural regularities to induce an SCFG and then directly use the grammar to generate more training data, which resulted in a significant improvement on the tasks. Unlike these works which augment a relatively small number of data and use them directly in end task training, we synthesize a large number of texts with SQL logic grounding to each table cheaply and use them for pre-training.

7.3 Methodology

7.3.1 Motivation

Semantic parsing data is usually related to some formal representations such as logic forms and SQL queries. Numerous prior works (Berant and Liang, 2014; Wang et al., 2015; Jia and Liang, 2016; Iyer et al., 2017; Andreas, 2020) have demonstrated the benefits of augmenting data using context-free grammar. The augmented examples can be used to teach the model to generalize beyond the given training examples.

However, data augmentation becomes more complex and less beneficial if we want to apply it to generate data for a random domain. More and more work (Zhang et al., 2019d; Herzig et al., 2020b; Campagna et al., 2020; Zhong et al., 2020b) shows utilizing augmented data doesn't always result in a significant performance gain in cross-domain semantic parsing end tasks. The most likely reason for this is that models tend to overfit to the canonical input distribution especially the generated utterances are very different compared with the original ones.

Moreover, instead of directly training semantic parsers on the augmented data, our work is the first to use the synthetic examples in pre-training and show it actually works if the overfitting problem is carefully addressed. To address the overfitting problem, in Section 7.3.3, we also include a small set of table related utterances in our pre-training data. We add an MLM loss on them as a regularization factor, which requires the model to balance between real and synthetic examples during the pre-training. We note that this consistently improves

the performance on all downstream semantic parsing tasks (see Section 7.5). Finally, our pre-training method is much more data-efficient and save much more computational power than other prior work (Section 7.6).

7.3.2 Data Synthesis with Synchronous Context-Free Grammar

We follow Jia and Liang (2016) to design our SCFG and apply it on a large amount of tables to populate new examples. For example, as shown in Figure 7.1, by replacing substitutable column mentions ("locations"), table mentions ("performance"), values ("two"), and SQL logic phrases ("at least") with the other possible candidates in the same group, our grammar generates new synthetic text-to-SQL examples with the same underlying SQL logic template. We then pre-train BERT on the augmented examples to force it to discover substitutable fragments and learn the underlying logic template so that it is able to generalize to other similar questions. Meanwhile, BERT also benefits from pre-training on a large number of different columns, table names, and values in the generated data, which could potentially improve schema linking in semantic parsing tasks.

Non-terminals	Production rules
TABLE $\rightarrow t_i$ COLUMN $\rightarrow c_i$ VALUE $\rightarrow v_i$ AGG $\rightarrow \langle$ MAX, MIN, COUNT, AVG, SUM$\rangle$ OP $\rightarrow \langle =, \leq, \neq, \ldots$, LIKE, BETWEEN $\rangle$ SC $\rightarrow \langle$ ASC, DESC $\rangle$ MAX $\rightarrow \langle$"maximum", "the largest"...$\rangle$ $\leq \rightarrow \langle$"no more than", "no above"...$\rangle$ ...	1. ROOT $\rightarrow \langle$"For each COLUMN0 , return how many times TABLE0 with COLUMN1 OP0 VALUE0 ?", `SELECT COLUMN0 , COUNT ( * ) WHERE COLUMN1 OP0 VALUE0 GROUP BY COLUMN0` $\rangle$ 2. ROOT $\rightarrow \langle$"What are the COLUMN0 and COLUMN1 of the TABLE0 whose COLUMN2 is OP0 AGG0 COLUMN2 ?", `SELECT COLUMN0 , COLUMN1 WHERE COLUMN2 OP0 ( SELECT AGG0 ( COLUMN2 ) )` $\rangle$

Table 7.1: Examples of non-terminals and production rules in our SCFG. Each production rule ROOT $\rightarrow \langle \alpha, \beta \rangle$ is built from some $(x, y) \in \mathcal{D}$ by replacing all terminal phrases with non-terminals. t_i, c_i, and v_i stand for any table name, column name, entry value respectively.

Grammar induction To induce a cross-domain SCFG, we study examples in SPIDER since it is a publicly available dataset that includes the largest number of examples with complex compositionalities in different domains. To further show the generality of our approach, we do not develop different SCFG for each downstream task. Given a set of (x, y)

pairs in SPIDER, where x and y are the utterance and SQL query respectively. We first define a set of non-terminal symbols for table names, column names, cell values, operations, etc. For example, in Table 7.1, we group aggregation operations such as MAX as a non-terminal AGG. We can also replace the entities/phrases with their non-terminal types in SQL query to generate a SQL production rule β. Then, we group (x, y) pairs by similar SQL production rule β. We automatically group and count Spider training examples by program templates, and select about 90 most frequent program templates β. For each program template in the grammar, we randomly select roughly 4 corresponding natural language questions, manually replace entities/phrases with their corresponding non-terminal types to create natural language templates α, and finally align them to generate each production rule ROOT $\rightarrow \langle \alpha, \beta \rangle$. The manual alignment approximately takes a few hours. About 500 SPIDER examples are studied to induce the SCFG.

Data augmentation With $\langle \alpha, \beta \rangle$ pairs, we can simultaneously generate pseudo natural questions and corresponding SQL queries given a new table or database. We first sample a production rule, and replace its non-terminals with one of corresponding terminals. For example, we can map the non-terminal AGG to MAX and "maximum" for the SQL query and the natural language sentence, respectively. Also, table content is used in synthesizing our pre-training data. For example, if the sampled production rule contains a value (e.g., VALUE0), we sample a value for the selected column from the table content and add it to the SQL and question templates. This way during pre-training, GRAPPA can access the table content and learn the linking between values and columns.

We use WIKITABLES (Bhagavatula et al., 2015), which contains 1.6 million high-quality relational Wikipedia tables. We remove tables with exactly the same column names and get about 340k tables and generate 413k question-SQL pairs given these tables. Also, we generate another 62k question-SQL pairs using tables and databases in the training sets of SPIDER and WIKISQL. In total, our final pre-training dataset includes 475k question-SQL

examples.

We note that SCFG is usually crude (Andreas, 2020) especially when it is applied to augment data for different domains. In this work we don't focus on how to develop a better SCFG that generates more natural utterances. We see this as a very interesting future work to explore. Despite the fact that the SCFG is crude, our downstream task experiments show that it could be quite effective if some pre-training strategies are applied.

7.3.3 Table Related Utterances

As discussed in Section 7.3.1, GRAPPA is also pre-trained on human annotated questions over tables with a MLM objective. We collected seven high quality datasets for textual-tabular data understanding (Table 7.8 in the Appendix), all of them contain Wikipedia tables or databases and the corresponding natural language utterances written by humans. We only use tables and contexts as a pre-training resource and discard all the other human labels such as answers and SQL queries.

7.3.4 Pre-Training GRAPPA

Unlike all the previous work where augmented data is used in the end task training, we apply the framework to language model pre-training. Training semantic parsers is usually slow, and augmenting a large amount of syntactic pairs directly to the end task training data can be prohibitively slow or expensive. In our work, we formulate text-to-SQL as a multi-class classification task for each column, which can be naturally combined with the MLM objective to pre-train BERT for semantic parsing. Moreover, in this way, the learned knowledge can be easily and efficiently transferred to downstream semantic parsing tasks in the exact same way as BERT (shown in Section 7.5).

GRAPPA is initialized by RoBERTa$_{\text{LARGE}}$ (Liu et al., 2019b) and further pre-trained on the synthetic data with SQL semantic loss and table-related data with MLM loss. As shown in Figure 7.1, we follow Hwang et al. (2019) to concatenate a user utterance and the column

headers into a single flat sequence separated by the </s> token. The user utterance can be either one of the original human utterances collected from the aggregated datasets or the canonical sentences sampled from the SCFG. We add the table name at the beginning of each column if there are some complex schema inputs involving multiple tables. We employ two objective functions for language model pre-training: 1) masked-language modelling (MLM), and 2) SQL semantic prediction (SSP).

MLM objective Intuitively, we would like to have a self-attention mechanism between natural language and table headers. We conduct masking for both natural language sentence and table headers. A small part of the input sequence is first replaced with the special token <mask>. The MLM loss is then computed by the cross-entropy function on predicting the masked tokens. We follow the default hyperparameters from Devlin et al. (2019) with a 15% masking probability.

SSP objective With our synthetic natural language sentence and SQL query pairs, we can add an auxiliary task to train our column representations. The proposed task is, given a natural language sentence and table headers, to predict whether a column appears in the SQL query and what operation is triggered. We then convert all SQL sequence labels into operation classification labels for each column. For example in the Figure 7.1, the operation classification label of the column "locations" is SELECT AND GROUP BY HAVING. In total, there are 254 potential classes for operations in our experiments.

For a column or table indexed by i, we use the encoding of the special token </s> right before it as its representation, denoted as $\mathbf{x_i}$ to predict its corresponding operations. On top of such representations, we apply a two-layer feed-forward network followed by a GELU activation layer (Hendrycks and Gimpel, 2016) and a normalization layer (Ba et al., 2016) to the output representations. Formally, we compute the final vector representation of each

column $\mathbf{y_i}$ by:

$$\mathbf{h} = \text{LayerNorm}(\text{GELU}(W_1 \cdot \mathbf{x_i}))$$

$$\mathbf{y_i} = \text{LayerNorm}(\text{GELU}(W_2 \cdot \mathbf{h}))$$

Finally, $\mathbf{y_i}$ is employed to compute the cross-entropy loss through a classification layer. We sum losses from all columns in each training example for back-propagation. For samples from the aggregated datasets, we only compute the MLM loss to update our model. For samples from the synthetic data we generated, we compute only SSP loss to update our model. More specifically, we mix 391k natural language utterances and 475k synthetic examples together as the final pre-training data. The examples in these two groups are randomly sampled during the pre-training, and MLM loss is computed if the selected example is a natural language question, otherwise SSP for a synthetic example.

7.4 Experiments

We conduct experiments on four *cross-domain* table semantic parsing tasks, where generalizing to unseen tables/databases at test time is required. We experiment with two different settings of table semantic parsing, fully supervised and weakly supervised setting. The data statistics and examples on each task are shown in Table 7.2 and Table 7.7 in the Appendix respectively.

Task & Dataset	# Examples	Resource	Annotation	Cross-domain
SPIDER (Yu et al., 2018c)	10,181	database	SQL	✓
Fully-sup. WIKISQL (Zhong et al., 2017)	80,654	single table	SQL	✓
WIKITABLEQUESTIONS (Pasupat and Liang, 2015)	2,2033	single table	answer	✓
Weakly-sup. WIKISQL (Zhong et al., 2017)	80,654	single table	answer	✓

Table 7.2: Overview of four table-based semantic parsing and question answering datasets in fully-supervised (top) and weakly-supervised (bottom) setting used in this paper. More details in Section 7.4

7.4.1 Supervised Semantic Parsing

We first evaluate GRAPPA on two supervised semantic parsing tasks. In a supervised semantic parsing scenario, given a question and a table or database schema, a model is expected to generate the corresponding program.

SPIDER SPIDER (Yu et al., 2018c) is a large text-to-SQL dataset. It consists of 10k complex question-query pairs where many of the SQL queries contain multiple SQL keywords. It also includes 200 databases where multiple tables are joined via foreign keys. For the baseline model, we use RAT-SQL + BERT (Wang et al., 2020) which is the state-of-the-art model according to the official leaderboard. We followed the official Spider evaluation to report set match accuracy.

Fully-sup. WIKISQL WIKISQL (Zhong et al., 2017) is a collection of over 80k questions and SQL query pairs over 30k Wikipedia tables. We use (Guo and Gao, 2019), a competitive model on WIKISQL built on SQLova (Hwang et al., 2019), as our base model. We adapt the same set of hyperparameters including batch size and maximum input length as in Guo and Gao (2019). For a fair comparison, we only consider single models without execution-guided decoding and report execution accuracy.

7.4.2 Weakly-supervised Semantic Parsing

We also consider weakly-supervised semantic parsing tasks, which are very different from SQL-guided learning in pre-training. In this setting, a question and its corresponding answer are given, but the underlying meaning representation (e.g., SQL queries) are unknown.

WIKITABLEQUESTIONS This dataset contains question-denotation pairs over single Wikipedia tables (Pasupat and Liang, 2015). The questions involve a variety of operations such as comparisons, superlatives, and aggregations, where some of them are hard to

answered by SQL queries.

We used the model proposed by Wang et al. (2019) which is the state-of-the-art parser on this task. This model is a two-stage approach that first predicts a partial "abstract program" and then refines that program while modeling structured alignments with differential dynamic programming. The original model uses GloVe (Pennington et al., 2014) as word embeddings. We modified their implementation to encode question and column names in the same way as we do in our fine-tuning method that uses RoBERTa and GRAPPA.

Weakly-sup. WIKISQL In the weakly-supervised setting of WIKISQL, only the answers (i.e., execution results of SQL queries) are available. We also employed the model proposed by Wang et al. (2019) as our baseline for this task. We made the same changes and use the same experiment settings as described in the previous section for WIKITABLEQUESTIONS.

7.4.3 Implementation of GRAPPA

For fine-tuning RoBERTa, we modify the code of RoBERTa implemented by Wolf et al. (2019) and follow the hyperparameters for fine-tuning RoBERTa on RACE tasks and use batch size 24, learning rate $1e$-5, and the Adam optimizer (Kingma and Ba, 2015). We fine-tune GRAPPA for 300k steps on eight 16GB Nvidia V100 GPUs. The pre-training procedure can be done in less than 10 hours. For all downstream experiments using GRAPPA or RoBERTa, we always use a BERT specific optimizer to fine-tune them with a learning rate of $1e$-5, while using a model-specific optimizer with the respective learning rate for the rest of the base models.

7.5 Experimental Results

We conducted experiments to answer the following two questions: 1) Can GRAPPA provide better representations for table semantic parsing tasks? 2) What is the benefit of two pre-

Models	Dev.	Test
Global-GNN (Bogin et al., 2019)	52.7	47.4
EditSQL (Zhang et al., 2019d)	57.6	53.4
IRNet (Guo et al., 2019)	61.9	54.7
RYANSQL (Choi et al., 2020)	70.6	60.6
TranX (Yin et al., 2020)	64.5	-
RAT-SQL (Wang et al., 2019)	62.7	57.2
w. BERT-large	69.7	65.6
w. RoBERTa-large	69.6	-
w. GRAPPA (MLM)	71.1(+1.4)	-
w. GRAPPA (SSP)	**73.6(+3.9)**	67.7(+2.1)
w. GRAPPA (MLM+SSP)	**73.4(+3.7)**	**69.6(+4.0)**

Table 7.3: Performance on SPIDER. We run each model three times by varying random seeds, and the average scores are shown.

Models	Dev.	Test
(Dong and Lapata, 2018)	79.0	78.5
(Shi et al., 2018)	84.0	83.7
(Hwang et al., 2019)	87.2	86.2
(He et al., 2019)	89.5	88.7
(Lyu et al., 2020)	89.1	89.2
(Guo and Gao, 2019)	90.3	89.2
w. RoBERTa-large	91.2	90.6
w. GRAPPA (MLM)	91.4	90.7
w. GRAPPA (SSP)	91.2	90.7
w. GRAPPA (MLM+SSP)	91.2	**90.8**
w. RoBERTa-large (10k)	79.6	79.2
w. GRAPPA (MLM+SSP) (10k)	**82.3(+2.7)**	**82.2(+3.0)**

Table 7.4: Performance on fully-sup. WIKISQL. All results are on execution accuracy without execution-guided decoding.

training objectives, namely MLM and SSP? Since GRAPPA is initialized by RoBERTa, we answer the first question by directly comparing the performance of base parser augmented with GRAPPA and RoBERTa on table semantic parsing tasks. For the second question, we report the performance of GRAPPA trained with MLM, SSP and also a variant with both of them (MLM+SSP).

Overall results We report results on the four aforementioned tasks in Tables 7.3, 7.4, 7.5, and 7.6 respectively. Overall, base models augmented with GRAPPA significantly outperforms the ones with RoBERTa by 3.7% on SPIDER, 1.8% on WIKITABLEQUESTIONS, and 2.4% on weakly-sup. WIKISQL, and achieve new state-of-the-art results across all four tasks. In most cases, the combined objective of MLM+SSP helps GRAPPA achieve better performance when compared with independently using MLM and SSP. Moreover, on the low-resource setting, GRAPPA outperforms RoBERTa by 3.0% in fully-sup. WIKISQL and 3.9% in WIKITABLEQUESTIONS. Detailed results for each task are discussed as follows.

SPIDER Results on SPIDER are shown in Table 7.3. When augmented with GRAPPA, the model achieves significantly better performance compared with the baselines using BERT and RoBERTa. Our best model, GRAPPA with MLM+SSP achieves the new state-of-the-art performance, surpassing previous one (RAT-SQL+BERT-large) by a margin of 4%. Notably,

most previous top systems use pre-trained contextual representations (e.g., BERT, TaBERT), indicating the importance of such representations for the cross-domain parsing task.

Fully sup. WIKISQL Results on WIKISQL are shown in Table 7.4. All GRAPPA models achieve nearly the same performance as RoBERTa. We suspect it is the relatively large training size and easy SQL pattern of WIKISQL make the improvement hard, comparing to SPIDER. Hence, we set up a low-resource setting where we only use 10k examples from the training data. As shown in the bottom two lines of Table 7.4, GRAPPA improves the performance of the SQLova model by 3.0% compared to RoBERTa, indicating that GRAPPA can make the base parser more sample-efficient.

WIKITABLEQUESTIONS Results on WIKITABLEQUESTIONS are shown in Table 7.5. By using RoBERTa and GRAPPA to encode question and column inputs, the performance of Wang et al. (2019) can be boosted significantly (>6%). Compared with RoBERTa, our best model GRAPPA (MLM+SSP) can further improve the performance by 1.8%, leading to a new state-of-the-art performance on this task. Similar to the low-resource case for WIKISQL, we also show the performance of the model when trained with only 10% of the training data. As shown at the bottom two lines Table 7.5, GRAPPA (MLM + SSP) specifically outperforms RoBERTa, again showing its superiority of providing better representations.

Weakly sup. WIKISQL Results on weakly supervised WIKISQL are shown in Table 7.6. GRAPPA with MLM+SSP again achieves the best performance when compared with other baselines, obtain the new state-of-the-art results of 84.7% on this task. It is worth noting that our best model here is also better than many models trained in the fully-supervised setting in Table 7.4. This suggests that inductive biases injected in pre-trained representation of GRAPPA can significantly help combat the issue of spurious programs introduced by learning from denotations (Pasupat and Liang, 2015; Wang et al., 2019) when gold programs are not available.

Models	Dev.	Test
(Liang et al., 2018)	42.3	43.1
(Dasigi et al., 2019)	42.1	43.9
(Agarwal et al., 2019)	43.2	44.1
(Herzig et al., 2020b)	-	48.8
(Yin et al., 2020)	52.2	51.8
(Wang et al., 2019)	43.7	44.5
$w.$ RoBERTa-large	50.7(+7.0)	50.9(+6.4)
$w.$ GRAPPA (MLM)	51.5(+7.8)	51.7(+7.2)
$w.$ GRAPPA (SSP)	51.2(+7.5)	51.1(+6.6)
$w.$ GRAPPA (MLM+SSP)	**51.9(+8.2)**	**52.7(+8.2)**
$w.$ RoBERTa-large $\times$10%	37.3	38.1
$w.$ GRAPPA (MLM+SSP) $\times$10%	**40.4(+3.1)**	**42.0(+3.9)**

Table 7.5: Performance on WIKITABLEQUESTIONS. Results trained on 10% of the data are shown at the bottom.

Models	Dev.	Test
(Liang et al., 2018)	72.2	72.1
(Agarwal et al., 2019)	74.9	74.8
(Min et al., 2019)	84.4	83.9
(Herzig et al., 2020b)	85.1	83.6
(Wang et al., 2019)	79.4	79.3
$w.$ RoBERTa-large	82.3 (+2.9)	82.3 (+3.0)
$w.$ GRAPPA (MLM)	83.3 (+3.9)	83.5 (+4.2)
$w.$ GRAPPA (SSP)	83.5(+4.1)	83.7 (+4.4)
$w.$ GRAPPA (MLM+SSP)	**85.9 (+6.5)**	**84.7 (+5.4)**

Table 7.6: Performance on weakly-sup. WIKISQL. We use (Wang et al., 2019) as our base model.

7.6 Analysis

Pre-training objectives GRAPPA trained with both MLM and SSP loss consistently outperforms the one trained with one of them (MLM+SSP vs. MLM only or SSP only). GRAPPA (MLM) usually improves the performance by around 1% such as 1.4% gain on SPIDER (dev), 0.8% on WIKITABLEQUESTIONS, and 1.2% on weakly-sup. WIKISQL. By pre-training on the synthetic text-to-SQL examples, GRAPPA (SSP), we can see a similar performance gain on these tasks too except 3.9% improvement on SPIDER dev, which is what we expected (grammar is overfitted to SPIDER). By pre-training with both MLM and SSP on the combined data, GRAPPA (MLM+SSP) consistently and significantly outperforms the one pre-trained with MLM or SSP separately (e.g., about +2% on Spider, +1.5% on WikiTableQuestions, and +1.2% on weakly-sup WikiSQL.). This contributes to our key argument in the chapter: in order to effectively enforce compositional interpolation in LM, pre-training on synthetic data should be regularized properly (using SSP+MLM together instead of SSP or MLM only) in order to balance between preserving the original BERT encoding ability and injecting compositional inductive bias, otherwise, the improvements are not robust and limited (using SSP or MLM only).

Generalization As mentioned in Section 7.3.2, we design our SCFG solely based on SPIDER, and then sample from it to generate synthetic examples. Despite the fact that GRAPPA pre-trained on such corpus is optimized to the SPIDER data distribution, which is very different from WIKISQL and WIKITABLEQUESTIONS, GRAPPA is still able to improve performance on the two datasets. In particular, for WIKITABLEQUESTIONS where the underlying distribution of programs (not necessarily in the form of SQL) are latent, GRAPPA can still help a parser generalize better, indicating GRAPPA can be beneficial for general table understanding even though it is pre-trained on SQL specific semantics. We believe that incorporating rules from a broader range of datasets (e.g. WIKITABLEQUESTIONS) would further improve the performance. However, in this chapter, we study rules from only the SPIDER dataset and test the effectiveness on other unseen datasets with different different underlying rules on purpose in order to show the generality of our method.

Even though GRAPPA is pre-trained on synthetic text-to-SQL data, the proposed pre-training method can also be applied to many other semantic parsing tasks with different formal programs (e.g., logic forms); and we also demonstrated the effectness of GRAPPA on non text-to-SQL tasks (weakly-supervised WIKISQL and WIKITABLEQUESTIONS where no programs are used, training is supervised by only answers/cell values) the underlying distribution of programs (not necessarily in the form of SQL) are latent. Furthermore, to design the SCFG and synthesize data with the corresponding programs labeled, we can use any formal programs such as the logic form or SParQL, and then employ the data to pre-train GraPPa. In this chapter we choose SQL as the formal program to represent the formal representation of the questions simply because more semantic parsing datasets are labeled in SQL.

Pre-training time and data Our experiments on the SPIDER and WIKITABLEQUESTIONS tasks show that longer pre-training doesn't improve and can even hurt the performance of the pre-trained model. This also indicates that synthetic data should be carefully used

in order to balance between preserving the original BERT encoding ability and enforcing compositional interpolation bias. The best result on SPIDER is achieved by using GRAPPA pre-trained for only 5 epochs on our relatively small pre-training dataset. Compared to other recent pre-training methods for semantic parsing such as TaBERT (Yin et al., 2020) and TAPAS (Herzig et al., 2020a), GRAPPA achieves the state-of-the-art performance (incorporated with strong base systems) on the four representative table semantic parsing tasks *in less 10 hours on only 8 16GB Nvidia V100 GPUs* (6 days on more than 100 V100 GPUs and 3 days on 32 TPUs for TaBERT and TAPAS respectively) Moreover, we encourage future work on studying how the size and quality of synthetic data would affect the end task performance. Also, GRAPPA (MLM+SSP) consistently outperforms other settings, which indicates that using MLM on the human annotated data is important.

Pre-training vs. training data augmentation Many recent work (Zhang et al., 2019d; Herzig et al., 2020b; Campagna et al., 2020; Zhong et al., 2020b) in semantic parsing and dialog state tracking show that training models on a combination of the extra synthetic data and original training data does not improve or even hurt the performance. For example, (Zhong et al., 2020b) synthesize data on training databases in several semantic parsing tasks including SPIDER, and find that training with this data augmentation leads to overfitting on the synthetic data and decreases the performance. In contrast, our pre-training approach could effectively utilize a large amount of synthesized data and improve downstream task performance. Also, the base parser with a GRAPPA encoder could usually converge to a higher performance in shorter time (see Section 7.8.1).

7.7 Conclusion and Future Work

In this chapter, we proposed a novel and effective pre-training approach for table semantic parsing. We developed a context-free grammar to automatically generate a large amount of question-SQL pairs. Then, we introduced GRAPPA, which is an LM that is pre-trained

on the synthetic examples with SQL semantic loss. We discovered that, in order to better leverage augmented data, it is important to add MLM loss on a small amount of table related utterances. Results on four semantic parsing tasks demonstrated that GRAPPA significantly outperforms RoBERTa.

While the pre-training method is surprisingly effective in its current form, we view these results primarily as an invitation for more future work in this direction. For example, this work relies on a hand-crafted grammar which often generates unnatural questions; Further improvements are likely to be made by applying more sophisticated data augmentation techniques. Also, it would be interesting to study the relative impact of the two objectives (MLM and SSP) by varying the respective number of pre-training examples. Furthermore, pre-training might benefit from synthesizing data from a more compositional grammar with a larger logical form coverage, and also from supervising by a more compositional semantic signals.

7.8 Appendices

Task	Question	Table/Database	Annotation
SPIDER	Find the first and last names of the students who are living in the dorms that have a TV Lounge as an amenity.	database with 5 tables e.g.`student`, `dorm_amenity`, ...	SELECT T1.FNAME, T1.LNAME FROM STUDENT AS T1 JOIN LIVES_IN AS T2 ON T1.STUID=T2.STUID WHERE T2.DORMID IN (SELECT T3.DORMID FROM HAS_AMENITY AS T3 JOIN DORM_AMENITY AS T4 ON T3.AMENID=T4.AMENID WHERE T4.AMENITY_NAME= 'TV LOUNGE')
Fully-sup. WIKISQL	How many CFL teams are from York College?	a table with 5 columns e.g. `player`, `position`, ...	SELECT COUNT CFL TEAM FROM CFLDRAFT WHERE COLLEGE = 'YORK'
WIKITABLEQUESTIONS	In what city did Piotr's last 1st place finish occur?	a table with 6 columns e.g. `year`, `event`, ...	"Bangkok, Thailand"
Weakly-sup. WIKISQL	How many CFL teams are from York College?	a table with 5 columns e.g. `player`, `position`,...	2

Table 7.7: Examples of the inputs and annotations for four semantic parsing tasks. SPIDER and Fully-sup. WIKISQL require full annotation of SQL programs, whereas WIKITABLEQUESTIONS and Weakly-sup. WIKISQL only requires annotation of answers (or denotations) of questions.

	Train Size	# Table	Task
TabFact	92.2K	16K	Table-based fact verification
LogicNLG	28.5K	7.3K	Table-to-text generation
HybridQA	63.2K	13K	Multi-hop question answering
WikiSQL	61.3K	24K	Text-to-SQL generation
WikiTableQuestions	17.6K	2.1K	Question answering
ToTTo	120K	83K	Table-to-text generation
Spider	8.7K	1K	Text-to-SQL generation

Table 7.8: Aggregated datasets for table-and-language tasks.

7.8.1 Additional Analysis

Training coverage As shown in Figure 7.2, on the challenging end text-to-SQL SPIDER task, RAT-SQL initialized with GRAPPA outperforms RAT-SQL using RoBERTa by about 14% in the early training stage. This shows that GRAPPA already captures some semantic knowledge in pre-training. Finally, GRAPPA is able to keep the competitive edge by 4%.

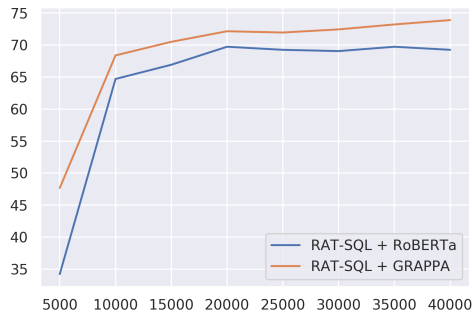

Figure 7.2: The development exact set match score in SPIDER vs. the number of training steps. RAT-SQL initialized with our pre-trained GRAPPA converges to higher scores in a shorter time than RAT-SQL w. BERT.

What if the task-specific training data is also used with the MLM or SSP objective in pre-training? Although we did not do the same experiments, we would like to point to the RAT-SQL chapter (Wang et al., 2020) for some suggestions. They add a similar alignment loss (similar to SSP) on the SPIDER training data and found that it doesn't make a statistically significant difference (in Appendix B).

Chapter 8

SCoRe: Pre-Training for Context Representation in Conversational Semantic Parsing

Conversational Semantic Parsing (CSP) is the task of converting a sequence of natural language queries to formal language (e.g., SQL, SPARQL) that can be executed against a structured ontology (e.g. databases, knowledge bases). To accomplish this task, a CSP system needs to model the relation between the unstructured language utterance and the structured ontology while representing the multi-turn dynamics of the dialog. Pre-trained language models (LMs) are state-of-the-art for various natural language processing tasks. However, existing pre-trained LMs that use language modeling training objectives over free-form text have limited ability to represent natural language references to contextual structural data. In this work, we present SCoRe, a new pre-training approach for CSP tasks designed to induce representations that capture the alignment between the dialogue flow and the structural context. We demonstrate the broad applicability of SCoRe to CSP tasks by combining SCoRe with strong base systems on four different tasks (SParC, CoSQL, MWoZ, and SQA). We show that SCoRe can improve the performance over all these base

systems by a significant margin and achieves state-of-the-art results on three of them. Our implementation and checkpoints of the model will be available at Anonymous URL.

8.1 Introduction

The goal of task-oriented dialog systems is to assist the user in completing a certain task by performing an action or retrieving relevant information (Tur and Mori, 2011). They are often built on top of a structured ontology grounded in a knowledge base, a database, or a set of API calls. This in contrast to open-domain dialog systems (also referred to as chit-chat systems) where the goal is to maximize engagement with users in open-ended conversations (Jafarpour et al., 2010; Ritter et al., 2011).

A key component of task-oriented conversational systems is Conversational Semantic Parsing (CSP), which converts each utterance in the dialog into a formal language query (e.g., SQL, SPARQL) that can be executed against the structured ontology. CSP has been extensively studied in several academic and industrial research settings such as dialog systems (e.g., dialog state tracking in MWOZ (Budzianowski et al., 2018)), interacting with physical agents (e.g., (Chai et al., 2018)), context-dependent semantic parsing (e.g., SPARC (Yu et al., 2019b)), SQL-grounded state tracking (e.g., COSQL (Yu et al., 2019a)), and sequential question answering (e.g., SQA (Iyyer et al., 2017)). These settings differ in some respect, but they share the same overall objective and key challenge: *how to jointly represent the natural language utterances and underlying structured ontology while taking into consideration the multi-turn dynamics of the dialog*.

Similar to many other natural language tasks, recent work in CSP has significantly benefited from advances in language model pre-training. However, existing general-purpose pre-trained language models, e.g. BERT (Devlin et al., 2019), are pre-trained on free-form text data using language model objectives. This limits their ability in modeling the structural context or the multi-turn dynamics of the dialogs. This presents an opportunity to improve

pre-trained LMs to specifically address these limitations for CSP tasks. Recent work has demonstrated the benefits of adapting pre-trained LMs to specific domains (Gururangan et al., 2020) or tasks (Zhang et al., 2019b) via a second phase of pre-training. For example, open-domain dialogue language models such as DialoGPT (Zhang et al., 2020a) and ConveRT (Henderson et al., 2019) are pre-trained on the Reddit data and applied to dialog response generation and retrieval tasks.

In this chapter, we introduce SCoRe (**S**tructured & **S**equential **Co**ntext **Re**presentation), a language model pre-training approach for CSP tasks. SCoRe adapts general pre-trained LMs by introducing a second phase of pre-training using multiple pre-training objectives that capture both multi-turn dynamics and the structural contexts in a dialog. In contrast to open-domain dialogs, CSP datasets are usually much smaller due to the difficulty and expense of obtaining and labeling data (mapping natural language utterances to formal language). Unlike most prior work on contextualized LMs which are pre-trained on free text, according to the finding where questions in CSP tasks usually can be mapped into formal representations, we propose to train SCoRe on synthesized conversational semantic parsing data with multiple training objectives that aim to ground utterances into the schema of the underlying ontology and to model the relationship between different utterances in the multi-turn conversation. In this way, SCoRe can effectively inject structural and conversational inductive biases in LMs that can translate to many CSP tasks. SCoRe uses an order of magnitude smaller dataset for the second stage of pre-training, does not require changes to the pre-trained model architecture, can be used as a drop-in replacement of general pre-trained LMs with any semantic parsing model, and can be used out-of-the-box in many CSP tasks.

We apply SCoRe to four different conversational semantic parsing tasks: (1) sequential text-to-SQL (SParC), (2) conversational text-to-SQL (CoSQL), (3) dialog state tracking (MWoZ), and (4) weakly-supervised sequential question answering (SQA). The fours tasks represent different scenarios, types of ontologies, supervision signals, system responses,

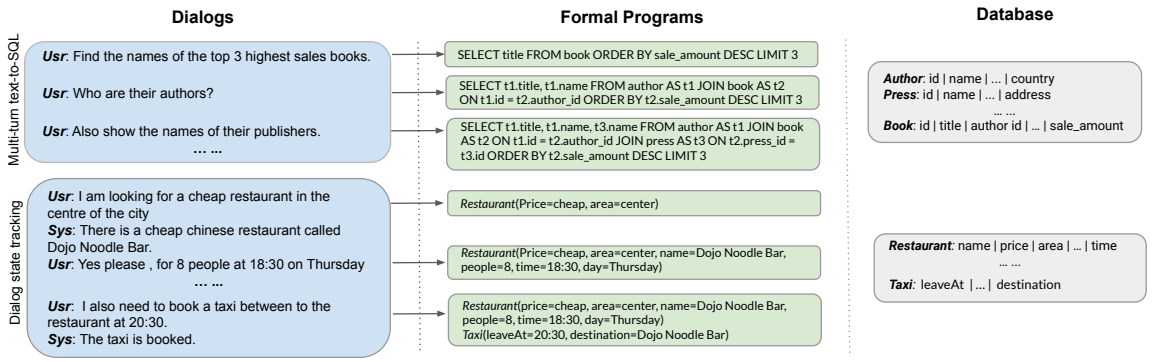

Figure 8.1: Examples of conversational semantic parsing tasks from SPARC and MWOZ datasets.

and domains (see Table 8.1 for a detailed comparison and Figure 8.1 for examples). We demonstrate that: (1) SCORE training objectives can effectively incorporate synthesized data, (2) a single pre-trained SCORE model can be used for several CSP tasks and can be combined with many baseline systems with different model architectures and (3) SCORE significantly improve all baseline systems and achieves new state-of-the-art results on three benchmarks (SPARC, SPARC, and MWOZ) and comparable performance to state-of-the-art results on the fourth (SQA).

8.2 Related Work

Conversational Semantic Parsing Conversational semantic parsing is one of the most important research topics in conversational AI and has been studied in different settings including task-oriented dialogues, question answering, and text-to-SQL. Task-oriented dialog systems (Henderson et al., 2014; Wen et al., 2016; Mrkšić et al., 2017; Budzianowski et al., 2018) aim to help users accomplish a specific task (e.g. flight booking) and often pre-define slot templates grounded in a domain-specific ontology. In comparison, several other datasets were recently introduced for cross-domain conversational text-to-SQL tasks (SPARC and COSQL (Yu et al., 2019a,b)) and sequential questions answers over tables (Iyyer et al., 2017). While the previous work has achieved significant progress in different

datasets separately, to the best of our knowledge, we are the first to study four different CSP tasks together (sequential text-to-SQL, conversational text-to-SQL, dialog state tracking, and weakly-supervised sequential question answering) by addressing the shared key challenge of learning representations in pre-trained language models that capture the alignment between the dialogue flow and the structural context.

Conversational Language Model Pre-training Several recent efforts have demonstrated the value of adapting pre-trained LMs to specific tasks using different pre-training objectives, e.g., summarization (Zhang et al., 2019b), knowledge inference (Sun et al., 2019b; Liu et al., 2019a), etc. Closest to our work is adapting pre-trained LMs for open-domain chit-chat models and for tabular data representation. The former focuses on improving response generation on open-ended dialogues by adding a pre-training step on open-domain conversations data, such as Reddit data (Zhang et al., 2020a; Henderson et al., 2019). For example, Wu et al. (2020) introduced ToD-BERT, a pre-trained language model combining 9 high-quality human-human task-oriented dialogue datasets to conduct language model and response selection pre-training. However, they use language modeling training objectives over free-form text and therefore have limited ability to represent structural data. The latter has focused on improving language model pre-training for encoding tabular data (Yin et al., 2020; Herzig et al., 2020b), but they focus on the single turn semantic parsing setting. Our approach is different from previous work because we address the challenge of conversational semantic parsing tasks by learning pretrained representation for both the multi-turn dynamics of the dialog and the relation between the unstructured language utterance and the structured ontology. Furthermore, our pre-training approach is much more data-efficient than prior LM pre-training work and saves a lot of time and computing resources (Appendix 8.7.4 for more details). Our pre-training step can be done within only one day using 8 V100 GPUs.

Using Synthesized Data for Semantic Parsing Synthesized data has been frequently used in semantic parsing to alleviate the challenge of labeled data scarcity. For example,

Wang et al. (2015) proposed a method for training semantic parsers in new domains by generating logical forms and canonical utterances and then paraphrasing the canonical utterances via crowd-sourcing. Similar approaches were used to train semantic parsers in other domains and settings (Zhong et al., 2017; Su et al., 2017; Cheng et al., 2018; Shah et al., 2018). Another line of work has proposed using synthesized data to adapt single turn semantic parsing models to new domains (Jia and Liang, 2016; Yoo et al., 2018; Campagna et al., 2019) and task-oriented dialogues (Campagna et al., 2020). However, they reported that combining synthetic data and the supervised data does not yield significant improvements, consistent with results by Herzig et al. (2020b). By contrast, we introduce a new data synthesis procedure for conversational text-to-SQL dialogues and use it in a different way by pretraining language models to induce better representations for many CSP tasks. Our synthesized data can be easily generated without human involvement and the pre-trained models add value to different tasks simultaneously.

8.3 Approach

The key challenge of CSP is to capture the relationship between the natural language utterance and the structured ontology in the multi-turn dialog dynamics. To this end, we inject structural and conversational inductive biases in SCoRe by introducing two objective functions: *Column Contextual Semantics (CCS)* objective and the *Turn Contextual Switch (TCS)* objective. Furthermore, to prevent SCoRe from overfitting to the linguistic pattern of our synthesized data, we use the *Masked Language Modeling (MLM)* objective on human-generated utterances as regularization. Because the size of existing semantic parsing datasets is limited, we produce synthesized data for pretraining SCoRe by sampling from the context-free grammar that is induced from complex text-to-SQL examples in different domains.

Dataset	Structured Ontology	Annotation (Supervision)	Cross Domain	System Response	# Dialogs	# Turns
SPARC	database	SQL (supervised)	✓	✗	4,298	12,726
COSQL	database	SQL (supervised)	✓	✓	3,007	15,598
MWOZ	domain ontology	slot-value (supervised)	✗	✓	8,438	113,556
SQA	table	denotation (weakly-supervised)	✓	✗	6,066	17,553

Table 8.1: Comparison of CSP datasets. Examples from two of the datasets are shown in Figure 8.1. Cross-domain means the train and test sets have different domains, so MWOZ is not cross-domain.

8.3.1 Preliminaries

Task Definition In the CSP task, at each turn t, we aim to produce a formal representation q_t given the current utterance u_t, the interaction history $h_t = [u_1, u_2, \ldots, u_{t-1}]$, and the schema c (table and column names, slots, etc.) of the target database (ontology) d. To cover different variants of the problem, we consider four popular CSP tasks shown in Table 8.1: SPARC (sequential text-to-SQL), COSQL (conversational text-to-SQL), MWOZ (dialogue state tracking), and SQA (weakly supervised sequential question answering). They have different target formal language and structured ontology:

- For the **utterance u**, it is the user question for SPARC and SQA, while for COSQL and MWOZ, u is the combination of a user query and a system response.
- For the **database d**, SPARC and COSQL use multi-table databases; for MWOZ, the pre-defined ontology d can also be viewed as a database; for SQA, d is a single table.
- For the **formal representation q**, it is the SQL query for SPARC and COSQL; in MWOZ it is the slot-value pairs that can be viewed as simple SQL queries consisting of `SELECT` and `WHERE` clauses; and for SQA, q is the latent program.

Base Architecture The base architecture of SCORE takes as input a single turn of a CSP dialog $\langle u_t, h_t \rangle$ jointly with the underlying database schema c. Given this *contextualized conversational input* $C_t = \langle u_t, h_t, c \rangle$, SCORE encodes it into *contextualized conversation representations* $\vec{\boldsymbol{S}}_t$ for each token in C_t. The encoder architecture follows RoBERTa (Liu et al., 2019b). It is then followed by a linear layer and normalized (Ba et al., 2016) to

produce final representations $\vec{\boldsymbol{h}}_t$ for each token:

$$C_t = \langle u_t, h_t, c \rangle,\ \vec{\boldsymbol{S}}_t = \text{RoBERTa}(C_t),\ \boldsymbol{h}_{t,i} = \text{LayerNorm}(\text{GELU}(\boldsymbol{W_1}\boldsymbol{S}_{t,i}))\ \forall\, \boldsymbol{S}_{t,i} \in \vec{\boldsymbol{S}}_t, \tag{8.1}$$

where GELU is an activation by Hendrycks and Gimpel (2016) and $\boldsymbol{W_1}$ is a learned parameter matrix.

To build C_t, we first concatenate current utterances u_t and dialog history h_t separated by a special token <s>, as this simple strategy has been shown effective in state-of-the-art CSP systems (Zhang et al., 2019d; Wu et al., 2019; Liu et al., 2020; Heck et al., 2020). To incorporate the database schema, we follow (Hwang et al., 2019) to concatenate all column names as a single sequence. Column names are separated by the special token </s> and prefixed by their corresponding table name.

8.3.2 SCoRe Pre-training

SCoRe addresses the challenges of CSP by *pre-training a task-oriented language model contextualized by the conversational flow and the underlying ontology*. In pre-training, the SCoRe model is self-supervised by two novel objectives in addition to the established Masked Language Modeling (MLM) objective. These objectives facilitate the accurate representation of the conversational flow between dialog turns and how this flow maps to the desired columns in the ontology.

Column Contextual Semantics The first challenge of CSP is capturing the alignment between the natural language utterance and the underlying database schema. To address it, we optimize the SCoRe model with the auxiliary objective of *Column Contextual Semantics (CCS)*. For each column in the database schema c, CCS targets the *operations* that should be performed on this column in a given conversational turn. Specifically, each formal representation q is decomposed into operations on columns and tables, e.g. `GROUP BY` and

HAVING for SQL queries, or WHERE for the slot-value pairs. In this way, our data covers 148 column operations. We use the encoding of the special token </s> right before each column or table name to predict its corresponding operations, and then compute the CCS loss:

$$\mathcal{L}_{\text{CCS}}(C_t) = \sum\nolimits_{i \in c} \text{CrossEntropy}_{148}(\text{LayerNorm}(\boldsymbol{W}_2\, \boldsymbol{h}^c_{t,i}), \text{CCS}(q_t)) \tag{8.2}$$

where $\boldsymbol{h}^c_{t,i}$ is the contextualized representation of the i^{th} column's special token </s> in the contextualized input C_t, $\text{CCS}(q_t)$ returns the column operation label for the current formal representation q_t, $\text{CrossEntropy}_{148}$ computes the 148-way cross-entropy between the column operation prediction and label, and $\boldsymbol{W}_2$ is a learned parameter matrix.

Turn Contextual Switch The second challenge of CSP is capturing the conversational context flow and how it is grounded into the formal representations. The TCS objective aims to capture this grounding of context flow. To this end, it targets predicting *the difference in formal representations between dialog turns* based on the natural language utterance.

Based on the context-free grammar of SQL, we identify 26 possible *turn difference operations* that a conversational turn could elicit. They encode changes between different turns of user queries (the system response is not involved here) since we assume that most turn contextual shifts are from the user. For example, INS(WHERE) indicates inserting a new WHERE condition and DEL(SELECT.agg) indicates removing an aggregate operation from a SELECT statement (e.g. when an utterance *"Show all the ages instead."* elicits a change SELECT MAX(age) ... → SELECT age ...). We use the encoding of the special token </s> right before each turn to predict the context switch label between this turn and the previous history:

$$\mathcal{L}_{\text{TCS}}(C_t) = \text{CrossEntropy}_{26}(\text{LayerNorm}(\boldsymbol{W}_3 \boldsymbol{H}^s_t), \text{TCS}(q_t, q_{t-1})) \tag{8.3}$$

where $\boldsymbol{H}^s_t \in \mathbb{R}^{(t-1)\times d}$ is the contextualized representation of all previous turns in C_t with hidden dimension d, $\text{TCS}(q_t, q_{t-1})$ returns the turn difference operations from q_{t-1} to q_t,

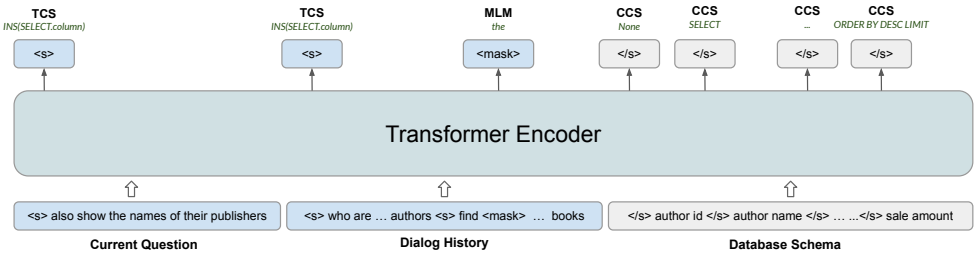

Figure 8.2: Pre-training of a SCORE encoder on a SPARC text-to-SQL example from Figure 8.1.

and $\boldsymbol{W}_3$ is a learned parameter matrix. We don't use this objective to pre-train SCORE for MWOZ because the context switch label between turns is relatively simple in MWOZ (only `select` and `where` changes).

Masked Language Modeling As in prior work on large-scale language models (Devlin et al., 2019), we use the *Masked Language Modeling (MLM)* objective to facilitate contextual representation learning for natural language utterances. Importantly for regularization, we only apply this loss on *in-domain human-annotated* natural language data. Namely, it includes utterances in SPARC, COSQL, and SQA as well as nine task-oriented dialog datasets processed by Wu et al. (2020) for MWOZ (see data statistics in Figure 8.4). Formally, the MLM loss is given by:

$$\mathcal{L}_{\text{MLM}}(C_t) = \sum_m \text{CrossEntropy}_{\text{Vocab}}(\text{LayerNorm}(\boldsymbol{W}_4 \boldsymbol{h}_t^m)) \tag{8.4}$$

where $\boldsymbol{h}_t^m$ are the contextualized representations of the masked 15% of tokens in C_t, and $\boldsymbol{W}_4$ is a learned parameter matrix.

Pre-Training Setup and Steps To summarize the pre-training steps, we first collect a dataset $\mathcal{D}_{\text{nat}}$ of combined human-annotated natural language questions (without labels) from existing CSP tasks (as mentioned above), and create a large synthesized conversational data $\mathcal{D}_{\text{syn}}$ that is generated by a grammar induced from a small set of SPARC annotated examples (See 8.3.3). After that, we incorporate both two datasets in pre-training. More

specifically, synthetic and natural examples are randomly sampled during pre-training. The total pre-training loss is the sum of the three objectives with CCS and TCS only applied to $\mathcal{D}_{\text{syn}}$ and MLM only to $\mathcal{D}_{\text{nat}}$:

$$\mathcal{L} = \sum_{C_t \in \mathcal{D}_{\text{syn}}} (\mathcal{L}_{\text{CCS}}(C_t) + \mathcal{L}_{\text{TCS}}(C_t)) + \sum_{C_t \in \mathcal{D}_{\text{nat}}} \mathcal{L}_{\text{MLM}}(C_t) \tag{8.5}$$

Figure 8.2 shows an overview of SCoRe pre-training on an example SParC dialogue from Figure 8.1. We report additional implementation details for pre-training SCoRe in Section 8.4.3 and Appendix 8.7.3.

8.3.3 Data Synthesis

We re-use the synthetic dataset of 120k synthetic task-oriented dialogues for MWoZ, introduced by Campagna et al. (2020). In this work, we introduce a complementary procedure to synthesize data for conversational text-to-SQL dialogues. We use about 400k tables in WikiTables (Bhagavatula et al., 2015) (after filtering and cleaning), WikiSQL, and Spider datasets as underlying databases d, and then synthesize about one dialog for each table. Finally, we synthesize 435k text-to-SQL conversations in total. Table 8.12 in Appendix 8.7.2 shows an example of the synthesized question-SQL pairs and their corresponding templates in our grammar.

Algorithm 1 Data synthesis algorithm

$\tilde{h} \leftarrow \emptyset$
$r_s \leftarrow \textsc{Sample}(G_s)$
$\tilde{u}_0, \tilde{q}_0 \leftarrow \textsc{RandAssignSlots}(d, r_s)$
$\tilde{h} += (\tilde{u}_0, \tilde{q}_0)$
$\tilde{u}_p, \tilde{q}_p \leftarrow \tilde{u}_0, \tilde{q}_0$
for $t \leftarrow 1$ to T **do**
    if $\textsc{Rand}(0, 1) < 0.2$ **then**
        $r_s \leftarrow \textsc{Sample}(G_s)$
        $\tilde{u}_t, \tilde{q}_t \leftarrow \textsc{RandAssignSlots}(d, r_s)$
    else
        $r_c \leftarrow \textsc{Sample}(G_c)$
        if $\textsc{ConstraintCheck}(r_c, \tilde{q}_p)$ **then**
            $\tilde{u}_t, \tilde{q}_t \leftarrow \textsc{EditAssign}(\tilde{q}_p, r_c)$
        end if
    end if
    $\tilde{h} += (\tilde{u}_t, \tilde{q}_t, r_c)$
    $\tilde{u}_p, \tilde{q}_p \leftarrow \tilde{u}_t, \tilde{q}_t$
end for
return $\tilde{h}$

To this end, we use only 500 dev examples from SPARC to induce two utterance-SQL generation grammars: (1) a single-turn context-free grammar G_s for generating context-independent question-SQL pairs, and (2) a follow-up context-free grammar G_c for follow-up question-SQL pairs. The single-turn grammar G_s contains a list of synchronous question-SQL templates where typed slots (`COLUMN0`, `OP0`, `VALUE0`, ...) represent mentions of tables, columns, values, and SQL operations. The follow-up grammar G_c contains context

switch labels and lists of follow-up question templates. For example, if the context switch label is `INS(SELECT.column0)`, the corresponding question could be *"How about show `column0` too?"*. To ensure generalization, we only induce the grammars from the SPARC training set. Appendix 8.7.2 shows examples of the grammar rules and synthesized utterances.

The data synthesis procedure using the two grammars is shown in Algorithm 1. Given a database d and a sampled single-turn question-SQL template, the function RANDASSIGNSLOTS samples values (column names, cell values, and SQL operations) for typed slots in the template and returns the first synthesized question $\tilde{u}_0$ and the corresponding SQL query $\tilde{q}_0$. To generate T follow-up question-SQL pairs, the function CONSTRAINTCHECK$(r_c, \tilde{q}_p)$ checks if the previous query $\tilde{q}_p$ satisfies constraints of the sampled template r_c (e.g. contains its mentioned nonterminal). Finally, EDITASSIGN$(\tilde{q}_p, r_c)$ edits the previous SQL $\tilde{q}_p$ to generate the current follow-up SQL label $\tilde{q}_t$ and samples values for typed slots in the template to generate the corresponding follow-up question $\tilde{u}_t$.

8.4 Experiment Settings

8.4.1 Datasets and Evaluation Metrics

We evaluate SCORE on four popular CSP tasks: SPARC (sequential text-to-SQL), COSQL (conversational text-to-SQL), MWOZ (dialogue state tracking), and SQA (sequential question answering), summarized in Table 8.1.

SPARC (Yu et al., 2019b) [33] is a large collection of sequences of inter-related context-dependent question-SQL pairs. It contains 4.3K questions sequences and 12k+ questions. **COSQL** (Yu et al., 2019a) [34] is a large conversational text-to-SQL corpus, with 3k dialogues, collected under the Wizard-of-Oz (WOZ) setting. We focus on the SQL-grounded dialogue

33. https://yale-lily.github.io/sparc

34. https://yale-lily.github.io/cosql

state tracking task which maps user intents into SQL queries if possible given the interaction history. Both SPARC and COSQL cover 200 complex DBs spanning 138 domains.

MWOZ (Budzianowski et al., 2018; Eric et al., 2019) [35] is a corpus of over 10k human-human written task-oriented dialogs created through a WOZ crowdsourcing setting. We focus on the belief state tracking task in MWOZ which maps multi-turn user utterances to slot-value annotations.

SQA (Iyyer et al., 2017) [36] is constructed from a subset of WikiTableQuestions (Pasupat and Liang, 2015) by decomposing highly compositional questions into a sequence of simple questions. The task is weakly-supervised because each resulting decomposed question is only annotated with answers as one or more table cells, while the logic program is latent. It has 6,066 question sequences with 17,553 questions in total on 982 unique open-domain tables from Wikipedia.

We adopt the official metrics defined for each of the tasks. For SPARC and COSQL, we report question match accuracy (QM): the exact set match accuracy (Yu et al., 2018c) over SQL templates and interaction match accuracy (IM): the ratio of interactions for which all questions are predicted correctly. For MWOZ, we report joint goal accuracy (JGA) which is similar to the IM accuracy used in SPARC and COSQL. Finally, for SQA, we report denotation QM and IM accuracies.

8.4.2 Base Models and other Baselines

For SPARC and COSQL, we use RAT-SQL (Wang et al., 2020) as our base model. Since it is originally developed for single-turn text-to-SQL, we extend it to a multi-turn setting by concatenating current utterances and dialog history (see Section 8.3.2). Note that RAT-SQL alone, without SCORE, achieves better or comparable results to state-of-the-art models developed for SPARC and COSQL.

35. https://github.com/budzianowski/multiwoz

36. http://aka.ms/sqa

For MWOZ, we employ Trippy (Heck et al., 2020). It achieves state-of-the-art performance on MWOZ and uses $\text{BERT}_{\text{base}}$ to encode user and system utterances and dialog history. We report higher results (around 2%) for Trippy than reported by Heck et al. (2020) since we train it for more epochs (25 vs. 10). To show the improvement of SCORE is not tied to specific base systems, we also experiment with another strong base model SOM-DST (Kim et al., 2020) for MWOZ and follow the same experimental details to train it.

For SQA, we use the weakly-supervised semantic parser proposed by Wang et al. (2019). The model first generates an abstract program given an input question and then instantiates it by searching for alignments between slots in the abstract program and question spans. As it is originally developed for single-turn questions, we extend it to the multi-turn setting in the same way as RAT-SQL.

We report additional implementation details for all base models in Appendix 8.7.3. In addition to reporting results for all base models with SCORE, we also report original base models results (with BERT and/or RoBERTa) and several other state-of-the-art baselines for each task.

8.4.3 Dataset Usage in Pre-training

In our experiments and ablation study, we train several versions of SCORE with different objectives and datasets: (1) SCORE (MLM): pre-trained on annotated natural questions using MLM. (2) SCORE (CCS+TCS): pre-trained on only synthesized data, which achieves the best results on SParC, CoSQL, and SQA. (3) SCORE (CCS+TCS+MLM): pre-trained on the synthesized data using CCS+TCS and annotated natural questions using MLM.

Furthermore, note that the synthesized data is generated using grammar induced by about 500 examples from only SPARC. Therefore, no COSQL or SQA data are seen in any pre-training steps. For MWOZ, Campagna et al. (2020) study only the dev examples to induce the data synthesis grammar.

	SPARC				CoSQL			
	Dev		Test		Dev		Test	
Models	QM	IM	QM	IM	QM	IM	QM	IM
SyntaxSQL (Yu et al., 2018b)	18.5	4.3	20.2	5.2	-	-	14.2	2.2
GAZP + BERT (Zhong et al., 2020b)	48.9	29.7	45.9	23.5	42.0	12.3	39.7	12.8
EditSQL + BERT (Zhang et al., 2019d)	47.2	29.5	47.9	25.3	39.9	12.3	40.8	13.7
IGSQL + BERT	50.7	32.5	51.2	29.5	44.1	15.8	42.5	15.0
R^2SQL + BERT	-	-	55.8	30.8	-	-	46.8	17.0
RAT-SQL + BERT (Wang et al., 2019)	56.8	33.4	-	-	48.4	19.1	-	-
+ RoBERTa	58.2	36.7	-	-	50.1	19.3	-	-
+ SCoRe	**62.2**	**42.5**	**62.4**	**38.1**	**52.1**	**22.0**	**51.6**	**21.2**

Table 8.2: The SPARC and CoSQL accuracy over all questions (QM) and all interactions (IM). The scores of IGSQL + BERT and R^2SQL + BERT are from the official leaderboards.

Models	MWoZ 2.1
DST-reader (Gao et al., 2019)	36.40
TRADE (Wu et al., 2019)	46.60
DS-DST (Zhang et al., 2019a)	51.21
SOM-DST (Kim et al., 2020)	52.57
DS-picklist (Zhang et al., 2019a)	53.30
TripPy (Heck et al., 2020)	55.29
SimpleToD (Hosseini-Asl et al., 2020)	55.72
TripPy (ours)	58.37
+ SCoRe	**60.48**

Table 8.3: Joint goal accuracies (JGA) on MWoZ 2.1 test set. All models use a BERT-like encoder/GPT.

	SQA	
Models	QM	IM
Pasupat and Liang (2015)	33.2	7.7
Neelakantan et al. (2016)	40.2	11.8
Iyyer et al. (2017)	44.7	12.8
Sun et al. (2019a)	45.6	13.2
Müller et al. (2019)	55.1	28.1
Herzig et al. (2020b)	**67.2**	**40.4**
Wang et al. (2019) + RoBERTa	62.8	33.2
Wang et al. (2019) + SCoRe	**65.4**	**38.1**

Table 8.4: Question (QM) and interaction (IM) accuracy on the SQA test set.

8.5 Results and Analysis

Overall Results The results of SPARC and CoSQL, MWoZ, and SQA are in Table 8.2, 8.3, and 8.4 respectively. We run each main experiment three times with different random seeds and report the mean. Overall, SCoRe gains significant improvements over BERT and RoBERTa on all tasks, achieving state-of-the-art performances on SPARC, CoSQL, and MWoZ.

For SPARC and CoSQL in Table 8.2, compared with RoBERTa, SCoRe boosts the performance by 4.0% QM / 5.8% IM on SPARC, and 2.0% QM / 2.7% IM on CoSQL. This demonstrates the effectiveness of SCoRe on contextual semantic parsing tasks. In

addition, on MWOZ dialog state tracking task in Table 8.3, TripPy achieves 60.5% JGA by replacing BERT with SCORE, outperforming the prior state-of-the-art (Hosseini-Asl et al., 2020) by 4.8%. This indicates that dialog state tracking also benefits from SCORE. Finally, SCORE also achieves higher performance than RoBERTa on weakly supervised sequential question answering SQA task. As Table 8.4 shows, SCORE improves QM by 2.6% and IM by 4.9% over RoBERTa with (Wang et al., 2019) as the base model. This demonstrates that the enhanced ability of semantic parsing and context modeling in SCORE is transferable to denotation-based CSP tasks.

Learning Objective	SPARC	COSQL	MWOZ	SQA
MLM only	37.0(+0.3)	20.3(+1.0)	59.47(+1.10)	34.7(+1.5)
CCS only	41.3(+4.6)	21.2(+1.9)	59.32(+0.95)	32.7(-0.5)
CCS+TCS	**42.5(+5.8)**	**22.0(+2.7)**	-	**38.1(+4.9)**
CCS+TCS+MLM	38.6(+1.9)	21.7(+2.4)	**60.48(+2.11)**	33.7(+0.5)

Table 8.5: The effect of SCORE pre-training objectives. Improvements are shown in the parentheses.

What is the effect of each pre-training objective? Table 8.5 shows an ablation study on different pre-training objectives. We find that the best SCORE results are achieved by pre-training on only synthesized data (CCS+TCS) without any natural questions (MLM) on SPARC, COSQL, and SQA but not on MWOZ. By adding MLM to CCS+TCS (CCS+TCS vs. CCS+TCS+MLM), MLM actually hurts the performance (-3.9% on SParC, -0.3% on CoSQL, and -4.4% on SQA) while increases for MWOZ. One possible reason is that questions in MWOZ are more diverse in language but less compositional while semantic compositionality and turn changes are more important in the other three CSP tasks. Also, the synthesized data used to pre-train SCORE for SPARC and COSQL is generated by the grammar induced by SPARC, which might overfit to SPARC. In addition, SCORE pre-trained with only MLM loss improves the performance (1.0%) but not as large as CCS+TCS (+5.5% on SPARC, +1.7% on COSQL, and +3.4% on SQA). Finally, we test the effectiveness of TCS on SPARC, COSQL, and SQA by adding TCS to CCS (CCS only

vs. CCS+TCS), SCoRe gains improvements of 1.2% on SParC and 0.8% on CoSQL, and 4.4% on SQA.

	QM	Q1	Q2	Q3	Q4
RAT-SQL + BERT	56.8	**71.1**	53.6	47.8	31.8
+RoBERTa	58.2	68.7	58.5	48.9	35.2
+ SCoRe	**62.2**	70.6	**63.5**	**52.6**	**45.5**

Table 8.6: Detailed results on the dev set of SParC. Q_i is the accuracy of the i^{th} conversation question.

Does SCoRe improve question match accuracy on individual turns? Table 8.6 shows detailed results of SCoRe's question accuracy for individual conversation turns on the SParC dev set. SCoRe provides a significant improvement for every conversation turn except the first (in which the task is more similar to single-turn semantic parsing). CoSQL and SQA exhibit similar behavior and are presented in Appendix 8.7.1.

	CoSQL	MWoZ
no syn	48.4	58.37
with syn	48.6	58.45

Table 8.7: Effect of synthetic data as training data augmentation.

What if we use the synthesized data to simply augment the training data? To answer this, we compare the results of the base models trained with or without the synthesized data on CoSQL and MWoZ. As shown in Table 8.7, the extra synthetic data does not significantly improve the performance, indicating that directly augmenting the synthetic data to the training set is not effective. The similar findings are reported in many recent work (Zhang et al., 2019d; Herzig et al., 2020b; Campagna et al., 2020; Zhong et al., 2020b). In contrast, pre-training on the synthesized data with our objectives improves the performance on the downstream tasks.

	MWOZ
SOM-DST + BERT	52.57
+ SCORE on syn. text-to-SQL	53.57
+ SCORE on syn. MWOZ	54.61

Table 8.8: Performance of SCORE pre-trained on different synthesized data on MWOZ.

How general is SCORE and its synthetic grammar? For generalization in task settings, we have shown that the pre-training strategy of SCORE can improve the performance over different CSP tasks including semantic parsing (SPARC and COSQL), dialog state tracking (MWOZ), and weakly supervised table question answering (SQA). In addition, we demonstrate the effectiveness of SCORE on different *base models*. To this end, we experiment with a different base model SOM-DST for MWOZ. As shown in Table 8.8, SCORE can still improve the performance with a different base model on MWOZ (SOM-DST+BERT vs. SOM-DST+SCORE on syn. MWOZ).

To demonstrate the generalization in synthetic grammar and data, as shown in Table 8.2 and 8.4, SCORE (TCS+CCS) is pre-trained on the synthesized data of the grammar induced from SPARC *only*, and it still improves the performance on COSQL (+2.7%) and SQA (+4.9%) where *no* any CoSQL and SQA annotated data is seen in any pre-training steps. Moreover, in Table 8.8 we show that SCORE pre-trained on the text-to-SQL synthesized data could also surprisingly improve the performance on MWOZ. We expect that higher performance could be achieved with SCORE pre-trained on task-specific synthesized data. Finally, our pre-training approach can be applied to *any* existing LMs including larger seq2seq LMs (e.g., BART (Lewis et al., 2020b), T5 (Raffel et al., 2020)).

	QM	IM
RoBERTa	53.3	21.2
SCORE	**57.1**	**26.1**

Table 8.9: Performance of SCORE on 10% training data of SQA.

Can SCoRe deliver more value when in-domain data is limited (e.g., in a low-resource setting)? We want to answer this question similar to experiments other investigations of LMs as few-shot learners (Wu et al., 2020; Brown et al., 2020; Schick and Schütze, 2020). To this end, we compare RoBERTa and SCoRe under a few-shot setting on SQA when only 10% of training data is available. We choose SQA because its annotation is most different from the synthetic text-to-SQL dataset we use for pretraining. Table 8.9 demonstrates that SCoRe delivers even larger improvements compared to the RoBERTa baseline when only 10% training data is available (3.8% vs 2.6%).

8.6 Summary

We presented SCoRe a new pre-training approach for conversational semantic parsing. The training objectives of SCoRe aim to induce natural language representations that capture the multi-turn dynamics, compositional semantic of the target language, and the references to the structural ontology appearing in the dialog. SCoRe can be used with many semantic parsing models as a drop-in replacement for general pretrained LMs. We demonstrated SCoRe effectiveness by using it as a feature representation encoder with strong baseline models for a wide range of CSP tasks. In particular, our empirical results on four different CSP tasks demonstrated that SCoRe can be used to significantly improve the performance of existing strong baseline models by simply replacing an existing pre-trained LM with our SCoRe pre-trained model. Furthermore, we are able to achieve state-of-the-art results on three of these tasks. We hope SCoRe will encourage further exploration of the benefits and limitations of pre-training approaches for CSP systems.

	QM	IM	Q1	Q2	Q3	Q4	Q5
RAT-SQL + BERT	48.4	19.1	54.6	48.4	**47.5**	43.9	31.0
+RoBERTa	50.1	19.3	59.7	50.9	46.3	46.5	**32.4**
+ SCoRe	**52.1**	**22.0**	**60.8**	**53.0**	**47.5**	**49.1**	**32.4**

Table 8.10: Detailed results of CoSQL on the dev set. Q_i is the accuracy of the i^{th} question in the conversation.

	QM	IM	Q1	Q2	Q3
Wang et al. (2019)	51.0	22.0	68.3	48.0	38.5
+RoBERTa	62.8	33.2	77.2	61.7	52.1
+SCoRe	**65.4**	**38.1**	**78.3**	**65.3**	**54.9**
Few-Shot (10% training data)					
Wang et al. (2019)					
+RoBERTa	53.3	21.2	71.0	52.5	36.6
+SCoRe	**57.1**	**26.7**	**74.6**	**56.7**	**40.7**

Table 8.11: Detailed results of SQA on the test set. Q_i is the accuracy of the i^{th} question in the conversation.

8.7 Appendices

8.7.1 Detailed Results

8.7.2 Synthesized Examples & Templates

Table 8.12 shows an example of the synthesized question-SQL pairs and their corresponding templates in our grammars.

8.7.3 Implementation Details

SCoRe

For pre-training SCoRe on synthesized text-to-SQL data, we use RoBERTa $_{\text{large}}$ and pre-train it with batch size 12, gradient accumulation step 2, and maximum length 248. We use a learning rate $1e$-5 and gradually reduce the learning rate without a warm-up period using

Turn #	Question-SQL Template	Synthesized Question-SQL
1	"Find the number of TABLE0 with COLUMN0 OP0 VALUE0" SELECT COUNT(*) ORDER BY COLUMN0 OP0 VALUE0	"Find the number of football team with team hometown is not murrieta, california?" SELECT COUNT(*) WHERE TEAM_HOMETOWN != "MURRIETA, CALIFORNIA"
2	"Can you give me their COLUMN1?" TCS: REPLACE(SELECT.COLUMN0), DEL(SELECT.AGG)	"Can you give me their football team player?" SELECT FOOTBALL_TEAM_PLAYER WHERE TEAM_HOMETOWN != "MURRIETA, CALIFORNIA"
3	"How about only show those with AS0 COLUMN2?" TCS: ADD(ORDERBY_AS0.COLUMN2)	"How about only show those with the largest age?" SELECT FOOTBALL_TEAM_PLAYER WHERE TEAM_HOMETOWN != "MURRIETA, CALIFORNIA" ORDER BY AGE DESC LIMIT 1
4	"AS1?" TCS: REPLACE(ORDERBY_AS1.COLUMN2)	"The smallest?" SELECT FOOTBALL_TEAM_PLAYER WHERE TEAM_HOMETOWN != "MURRIETA, CALIFORNIA" ORDER BY AGE AS LIMIT 1

Table 8.12: An example of synthetic conversational text-to-SQL data.

Adam (Kingma and Ba, 2015) with epsilon $1e$-8. $\text{BERT}_{\text{base}}$ is used in pre-training SCoRE on synthesized MWoZ data because it contains longer conversations. We set the maximum length to 512 and batch size 24. All SCoRE are pre-trained for 30 epochs, which usually take less than half a day on 8 V100 GPUs. We experimented with SCoRE pre-trained for 5, 10, and 30 epochs and found that most of the best downstream performances occur when base systems incorporate with SCoRE pre-trained for less than 10 epochs. Our implementation is based on the Transformers library (Wolf et al., 2019).

Base Models

RAT-SQL: For a fair comparison, all RAT-SQL experiments are trained for 40k steps. We adopt the same hyperparameters as (Shaw et al., 2018) except for learning rates. We find that learning rates of $1e$-4 and $1e$-5 for RAT and BERT respectively produce more stable results.

TripPy: We use the same hyperparameters for training TripPy on MWoZ as in (Heck et al., 2020) except we train it for 25 epochs (as opposed to 10 epochs as reported in (Heck et al., 2020)). When we train TripPy for 25 epochs, we get a new result that is higher (around

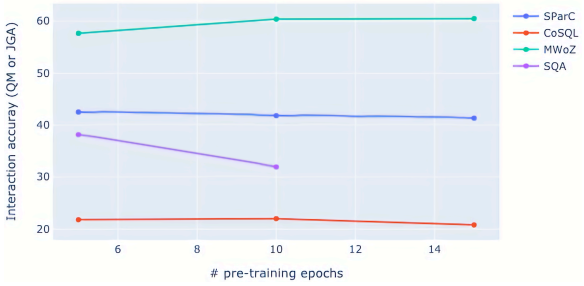

Figure 8.3: The effect of pre-training time.

2%) than the one reported in (Heck et al., 2020). Similarly, when we train TripPy with SCoRE, we train it for 25 epochs.

SOM-DST: We use the same hyperparameters from Kim et al. (2020) for all SOM-DST experiments on MWoZ.

Wang et al. (2019): We use the same hyperparameters from Wang et al. (2019) for SQA experiments. Note that Herzig et al. (2020b) outperform Wang et al. (2019) on SQA because (1) they don't generate logic forms but select table cells and applying aggregation operators. Wang et al. (2019) generate latent programs, yet the grammar of the latent program can only cover 87% questions. (2) They reduce the search space by reusing the previous question answer. We choose Wang et al. (2019) as our base model because generating symbolic programs has many practical advantages (even at a cost of around 1% accuracy drop), such as showing interpretable reasoning steps, enabling formal reasoning, and operationalization without GPU/TPU accelerators.

8.7.4 Pre-Training Cost

We test the performance of SCoRE with respect to the number of pre-training epochs. Figure 8.3 shows that the best performance of the downstream tasks is usually achieved in early epochs, more specifically 5 for SParC and CoSQL and 15 for MWoZ. Longer pre-training time does not improve or even hurts the performance. One possible reason is that longer pre-training makes SCoRE overfit to the synthesized data whose utterances are unnatural.

As for the data, as shown in Table 8.5, even if SCoRE is pre-trained with only a relatively small amount of synthesized data (without the MLM loss), most of the tasks can achieve much higher performances. With a relatively smaller training corpus and shorter training time compared to other pre-trained language models, SCoRE is efficient in time and data.

8.7.5 Additional Results

Effect of TCS We ran the TCS only experiment on SPARC, and will add TCS only results (including for other tasks) to Table 8.5 in the final version. SCoRE (TCS only) outperforms RoBERTa by 2.4% so far (note: training is still going on) on SPARC (39.1% vs. 36.7%). Also, as discussed in Section 8.5, we also provide a secondary evidence by testing the effectiveness of TCS on SPARC, COSQL, and SQA by adding TCS to CCS (CCS only vs. CCS+TCS), SCoRE (with TCS) gains improvements of 1.2% on SPARC and 0.8% on COSQL, and 4.4% on SQA.

Incorporating Additional Examples Used in Synthetic Grammar Induction As we mentioned in Section 8.3.3, we used about 500 examples from SPARC to induce the grammar for data synthesis in pre-training. For a fair comparison, we also report the results of incorporating the additional SPARC examples in COSQL and SQA. More specifically, we directly concatenate the additional SPARC examples to COSQL training set, and train RAT-SQL+RoBERTa on it, which slightly improves the performance (19.6% vs. 19.3%) but not as large as SCoRE (22.0% vs. 19.3%).' Also, because SQA is weakly-supervised sequential question answering, which differs from SPARC, we first fine-tune RoBERTa on the additional SPARC examples using CCS, and then apply it to SQA. In this way, the RoBERTa trained with additional SPARC examples achieves a similar performance as the original one (62.7% vs 62.8%).

Performance Comparison with ToD-BERT ToD-BERT is pre-trained on human-annotated questions with both MLM and response contrastive objectives. To compare TOD-BERT with SCoRe, we ran experiments of RAT-SQL + ToD-BERT on SParC. SCoRe (62.2%) outperforms ToD-BERT (54.6%) by 7.6%.

Comparison with Finetuning Larger Language Models Based on our experiments and other published results, we didn't find existing larger LMs (BART (Lewis et al., 2020b), T5 (Raffel et al., 2020), GPT-2 (Radford et al., 2019)) outperform custom models + BERT on CSP tasks. Our evidence is based on Spider (Yu et al., 2018c), which is the single-turn version of SParC and CoSQL. For T5, Shaw et al. (2020) applied T5 as seq2seq to Spider, and compared with RAT-SQL + BERT-Large, T5-Base performs much worse (57.1% vs. 69.6%), and T5-3B improves only 0.3, but it is 6 times larger. Moreover, for Bart, we have performed experiments on Spider and we found that BART cannot outperform custom models + BERT: RAT-SQL + BERT 69.7%, RAT-SQL + BART encoder 67.8%, BART encoder + decoder (406M, as a seq2seq task) 62.4%. In Rubin and Berant (2020), BART didn't outperform BERT either. As for GPT-2, Wu et al. (2020) and Hosseini-Asl et al. (2020) found it does not outperform BERT on MWoZ.

8.7.6 Task-Oriented Dialogue Datasets

Name	# Dialogue	# Utterance	Avg. Turn	# Domain
MetaLWOZ (Lee et al., 2019)	37,884	432,036	11.4	47
Schema (Rastogi et al., 2019)	22,825	463,284	20.3	17
Taskmaster (Byrne et al., 2019)	13,215	303,066	22.9	6
MWOZ (Budzianowski et al., 2018)	10,420	71,410	6.9	7
MSR-E2E (Li et al., 2018)	10,087	74,686	7.4	3
SMD (Eric and Manning, 2017)	3,031	15,928	5.3	3
Frames (Asri et al., 2017)	1,369	19,986	14.6	3
WOZ (Mrkšić et al., 2016)	1,200	5,012	4.2	1
CamRest676 (Wen et al., 2016)	676	2,744	4.1	1

Figure 8.4: Data statistics of human-annotated task-oriented dialogue datasets used in (Wu et al., 2020).

Chapter 9

Conclusion and Future Work

Datasets The project in Chapter 2 is Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In this chapter, We present SPIDER, a large-scale, complex, and cross-domain semantic parsing and text-to-SQL dataset. It consists of 10,181 questions and 5,693 unique complex SQL queries on 200 databases with multiple tables, covering 138 different domains. We define a new complex and cross-domain semantic parsing and text-to-SQL task where different complex SQL queries and databases appear in train and test sets. In this way, the task requires the model to generalize well to both new SQL queries and new database schemas. SPIDER is distinct from most of the previous semantic parsing tasks because they all use a single database and the exact same programs in the train set and the test set. We experiment with various state-of-the-art models and the best model achieves only 12.4% exact matching accuracy on a database split setting. This shows that SPIDER presents a strong challenge for future research.

The project in Chapter 3 is SParC: Cross-Domain Semantic Parsing in Context. We present SPARC, a dataset for cross-domain Semantic Parsing in Context. It consists of 4,298 coherent question sequences (12k+ individual questions annotated with SQL queries), obtained from controlled user interactions with 200 complex databases over 138 domains.

We provide an in-depth analysis of SPARC and show that it introduces new challenges compared to existing datasets. SPARC (1) demonstrates complex contextual dependencies, (2) has greater semantic diversity, and (3) requires generalization to new domains due to its cross-domain nature and the unseen databases at test time. We experiment with two state-of-the-art text-to-SQL models adapted to the context-dependent, cross-domain setup. The best model obtains an exact set match accuracy of 20.2% over all questions and less than 10% over all interaction sequences, indicating that the cross-domain setting and the contextual phenomena of the dataset present significant challenges for future research.

Furthermore, Chapter 4 presents CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Natural Language Interfaces to Databases. We introduce COSQL, a corpus for building cross-domain, general-purpose database (DB) querying dialogue systems. It consists of 30k+ turns plus 10k+ annotated SQL queries, obtained from a Wizard-of-Oz (WOZ) collection of 3k dialogues querying 200 complex DBs spanning 138 domains. Each dialogue simulates a real-world DB query scenario with a crowd worker as a user exploring the DB and a SQL expert retrieving answers with SQL, clarifying ambiguous questions, or otherwise informing of unanswerable questions. When user questions are answerable by SQL, the expert describes the SQL and execution results to the user, hence maintaining a natural interaction flow. COSQL introduces new challenges compared to existing task-oriented dialogue datasets: (1) the dialogue states are grounded in SQL, a domain-independent executable representation, instead of domain-specific slot-value pairs, and (2) because testing is done on unseen databases, success requires generalizing to new domains. COSQL includes three tasks: SQL-grounded dialogue state tracking, response generation from query results, and user dialogue act prediction. We evaluate a set of strong baselines for each task and show that COSQL presents significant challenges.

Algorithms The project in Chapter 5 is TypeSQL: Knowledge-based Type-Aware Neural Text-to-SQL Generation. In this chapter, we present a novel approach, TYPESQL, which

views the SQL generation problem as a slot filling task. Additionally, TYPESQL utilizes type information to better understand rare entities and numbers in natural language questions. We test this idea on the WikiSQL dataset and outperform the prior state-of-the-art by 5.5% in much less time. We also show that accessing the content of databases can significantly improve the performance when users' queries are not well-formed. TYPESQL gets 82.6% accuracy, a 17.5% absolute improvement compared to the previous content-sensitive model.

The project presented in Chapter 6 is SyntaxSQLNet: Syntax Tree Networks for Complex and Cross-Domain Text-to-SQL Task (Yu et al., EMNLP 2018). In this chapter, we propose SYNTAXSQLNET, a syntax tree network to address the complex and cross-domain text-to-SQL generation task. Most existing studies in text-to-SQL tasks do not require generating complex SQL queries with multiple clauses or sub-queries, and generalizing to new, unseen databases. SYNTAXSQLNET employs a SQL-specific syntax tree-based decoder with SQL generation path history and table-aware column attention encoders. We evaluate SYNTAXSQLNET on the SPIDER text-to-SQL task, which contains databases with multiple tables and complex SQL queries with multiple SQL clauses and nested queries. We use a database split setting where databases in the test set are unseen during training. Experimental results show that SYNTAXSQLNET can handle a significantly greater number of complex SQL examples than prior work, outperforming the previous state-of-the-art model by 7.3% in exact matching accuracy. We also show that SYNTAXSQLNET can further improve the performance by an additional 7.5% using a cross-domain augmentation method, resulting in a 14.8% improvement in total. To our knowledge, we are the first to study this complex and cross-domain text-to-SQL task.

Language Model Pre-Training Finally, I also worked on pre-training for table-based semantic parsing. Semantic parsing tasks pose two main challenges: (1) handling question complexity and (2) generalizing across domains. More precisely, a complex question in any domain can refer to entities and values that do not appear in free text and can utilize

implicit references that are domain-specific or even ambiguous. Therefore, tasks under these settings require not only the encoding of natural language questions, which BERT excels at but also the encoding of the table's schema and values in a manner that captures their relationship with references in the question as well as logical operations over those relationships. Many existing systems, even when using BERT, still struggle on tasks that require the alignment of both such encodings. To this end, we propose a BERT fine-tuning objective that can efficiently perform such schema linking and logic encoding, and introduce GRAPPA (chapter 7) and SCORE (chapter 8), a BERT-based model fined-tuned with that objective on a large number of tables and questions generated by context-free grammar templates. Despite its simplicity, we show that GRAPPA and SCORE significantly outperform BERT on multiple different table-related semantic parsing tasks under both fully and weakly supervised settings.

9.1 Future Work

Looking forward, despite the tremendous progress in building conversational natural language interfaces, much work remains to make such systems viable for real-world applications. Building upon my past work, I plan to explore the following new directions and challenges.

Trustworthy and Explainable NLI While deep learning has become the de facto approach to build intelligent systems, the improvement of performance often comes at the cost of interpretability. Complex neural networks permit easy architectural and operational variations for state-of-the-art accuracy, yet they provide little transparency about their inner decision-making mechanisms. I am interested in how natural language can promote interpretable AI: language is not only the means of communication between humans, but it also offers a medium for an intelligent system to explain and rationalize its solutions. To this end, I would like to empower NLI systems with the ability to automatically extract

or generate human-readable language explanations to justify their predictions or actions. When an NLI system returns results (such as a number or a displayed table) to a query, it is crucial to explain how the system behaved as it did, which better enables the user to verify and interpret these results. An advantage for language grounding is that the predicted formal program directly represents how systems understand the user's input and could be explicitly translated back into natural language for user verification. In particular, I would like to develop more controllable models such that the generated text remains faithful to the conditioned text input (logic-perseverance high-fidelity). Furthermore, it is reasonable to have a system that fails in some cases as long as it confirms with the users the failures. I aim to develop models that could show their prediction confidence and present their prediction for user quick verification. Finally, an NLI system should handle and detect all kinds of questions, including unanswerable ones. Instead of assuming all the questions can be mapped into some formal programs, the system should act or provide different feedback to the users and interact with them differently.

Ubiquitous NLI Except for databases, data requests over the web, mobile applications, vision, and robots are widely seen in daily life. To accomplish a task, the user interacts with them differently by asking natural language requests. For example, the user can interact with a mobile application via a sequence of low-level user interface actions, such as clicking an element, swiping to check the rest of a list, and typing a string. In particular, I am passionate about unifying formal representations used in language grounding in different environments and studying them together. In this way, we need not develop a different isolated NLI that is designed to accomplish a specific task. We can have a single system that could answer users' questions over databases, understand and execute housework commands, search over the web, manipulate the apps to book flights, summarize meeting discussions, among other tasks. Finally, in the longer term, I hope to develop an NLI system that utilizes multiple data sources such as text, knowledge bases, databases, and online tables to respond to a

user query. Such a system requires reasoning over multiple data sources. Thus, a single target program such as SQL is not sufficient. Rather, this system should be able to convert user queries to multiple target programs to reason over multiple sources and then fuse this information to derive a correct answer.

Human-Centered NLI Moreover, allowing users to input, provide feedback, and verify results from multi-modalities could improve users' engagement, help them formalize the queries, and build trust between them and the system. A robust and user-friendly natural language interface needs to be able to correct its errors, learn from the user, and deal with ambiguous and unanswerable questions. Most of these functions are not well-studied. I will explore how an NLI system can effectively clarify ambiguous questions or guide the user to form valid input to the NLI. To do so, I hope to bring together ideas and technologies from the fields of human-computer interaction (HCI), dialog systems, and interactive learning. Specifically, my future work will focus on answering the following questions: (a) how does a natural language interface engage in disambiguation, or adjust its capability set, when uncertain about a user question or its response? (b) how can the system detect ambiguous and unanswerable questions, and then generate informative responses to clarify and guide a user to form valid input to the NLI? (c) how can the system learn from the user's correction actions over time to improve its performance? (d) how can a system communicate its uncertainty in response generation with different groups of people with a different background?

Ecologically Valid NLI Despite many recent advances in developing next-generation NLIs, the extent to which these improvements on related benchmarks can translate into more practical and useful NLI tasks reflecting real user needs requires further investigation. Many recent NLI benchmarks opted for cheaper and more scalable methods such as crowdsourcing. However, data used for training and evaluating the learned NLIs do not fully reflect the intents and linguistic phenomena found in real-world applications. To resolve this, I plan

to design more realistic NLI tasks by collecting data from diverse target user settings. Furthermore, domain adaptation is critical to leveraging the existing general-purpose NLI datasets to improve a system's performance on real data. Non-technical users tend to implicitly express their requests and ignore some domain knowledge, and injecting this domain knowledge into the NLI systems learned on general data is a key challenge. Finally, NLI evaluation should reflect the real performance with real users. The key issue with the current evaluation procedure is that it does not account for errors that the system makes throughout the conversation. Evaluating under the assumption of ground-truth inputs does not measure how well the system can recover from its own mistakes. I would like to measure NLIs through a human-in-the-loop evaluation that assesses whether the interaction as a whole is successful over various datasets collected in different ways. The evaluation system also has to interact with real users that closely represent the target user population with different backgrounds, and multiple evaluation indicators should be included such as user engagement and user confidence with the system.

Bibliography

Rishabh Agarwal, Chen Liang, Dale Schuurmans, and Mohammad Norouzi. Learning to generalize from sparse and underspecified rewards. In *ICML*, 2019.

Rakesh Agrawal and Ramakrishnan Srikant. Searching with numbers. *IEEE Trans. Knowl. Data Eng.*, 15(4):855–870, 2003.

Miltiadis Allamanis, Daniel Tarlow, Andrew D. Gordon, and Yi Wei. Bimodal modelling of source code and natural language. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2123–2132. JMLR.org, 2015.

Jacob Andreas. Good-enough compositional data augmentation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7556–7566, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.676.

Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association forComputational Linguistics*, 2013.

Yoav Artzi and Luke S. Zettlemoyer. Bootstrapping semantic parsers from conversations. In *EMNLP*, 2011.

Layla El Asri, Hannes Schulz, Shikhar Sharma, Jeremie Zumer, Justin Harris, Emery

Fine, Rahul Mehrotra, and Kaheer Suleman. Frames: A corpus for adding memory to goal-oriented dialogue systems. *CoRR*, abs/1704.00057, 2017.

Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *ArXiv*, abs/1607.06450, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, 2013.

Jonathan Berant and Percy Liang. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1415–1425, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

Núria Bertomeu, Hans Uszkoreit, Anette Frank, Hans-Ulrich Krieger, and Brigitte Jörg. Contextual phenomena and thematic relations in database qa dialogues: results from a wizard-of-oz experiment. In *Proceedings of the Interactive Question Answering Workshop at HLT-NAACL 2006*, pages 1–8. Association for Computational Linguistics, 2006.

Chandra Bhagavatula, Thanapon Noraset, and Doug Downey. Tabel: Entity linking in web tables. In *International Semantic Web Conference*, 2015.

Ben Bogin, Matt Gardner, and Jonathan Berant. Global reasoning over database structures for text-to-sql parsing. *ArXiv*, abs/1908.11214, 2019.

Antoine Bordes and Jason Weston. Learning end-to-end goal-oriented dialog. *ICLR*, 2017.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Pawel Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gasic. Multiwoz - a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. In *EMNLP*, 2018.

Giovanni Campagna, Silei Xu, Mehrad Moradshahi, Richard Socher, and Monica S. Lam. Genie: A generator of natural language semantic parsers for virtual assistant commands. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, page 394–410. Association for Computing Machinery, 2019.

Giovanni Campagna, Agata Foryciarz, Mehrad Moradshahi, and Monica S. Lam. Zero-shot transfer learning with synthesized data for multi-domain dialogue state tracking. In *Proceedings of 58th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2020.

Joyce Y. Chai and Rong Jin. Discourse structure for context question answering. In *HLT-NAACL 2004 Workshop on Pragmatics in Question Answering*, 2004.

Joyce Y Chai, Qiaozi Gao, Lanbo She, Shaohua Yang, Sari Saba-Sadiya, and Guangyue Xu. Language to action: Towards interactive task learning with physical agents. In *IJCAI*, pages 2–9, 2018.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. Tabfact: A large-scale dataset for table-based fact verification. *arXiv preprint arXiv:1909.02164*, 2019.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Wang. Hybridqa: A dataset of multi-hop question answering over tabular and textual data. *arXiv preprint arXiv:2004.07347*, 2020.

Yen-Chun Chen and Mohit Bansal. Fast abstractive summarization with reinforce-selected sentence rewriting. In *Proceedings of ACL*, 2018.

Jianpeng Cheng, Siva Reddy, and Mirella Lapata. Building a neural semantic parser from a domain ontology. *ArXiv*, abs/1812.10037, 2018.

Donghyun Choi, Myeong Cheol Shin, Eunggyun Kim, and Dong Ryeol Shin. Ryansql: Recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases. *ArXiv*, abs/2004.03125, 2020.

Eunsol Choi, He He, Mohit Iyyer, Mark Yatskar, Wen-tau Yih, Yejin Choi, Percy Liang, and Luke Zettlemoyer. Quac: Question answering in context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2174–2184. Association for Computational Linguistics, 2018.

Shumo Chu, C. Wang, Konstantin Weitz, and A. Cheung. Cosette: An automated prover for sql. In *CIDR*, 2017.

Shumo Chu, Brendan Murphy, Jared Roesch, Alvin Cheung, and Dan Suciu. Axiomatic foundations and algorithms for deciding semantic equivalences of sql queries. *Proceedings of the VLDB Endowment*, 11(11):1482–1495, 2018.

Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. Expanding the scope of the atis task: The atis-3 corpus. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics.

Dipanjan Das, Nathan Schneider, Desai Chen, and Noah A. Smith. Probabilistic frame-semantic parsing. In *NAACL*, 2010.

Pradeep Dasigi, Matt Gardner, Shikhar Murty, Luke S. Zettlemoyer, and Eduard H. Hovy. Iterative search for weakly supervised semantic parsing. In *NAACL-HLT*, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

Bhuwan Dhingra, Lihong Li, Xiujun Li, Jianfeng Gao, Yun-Nung Chen, Faisal Ahmed, and Li Deng. End-to-end reinforcement learning of dialogue agents for information access. In *ACL*, 2016.

Li Dong and Mirella Lapata. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

Li Dong and Mirella Lapata. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742. Association for Computational Linguistics, 2018. URL http://aclweb.org/anthology/P18-1068.

Mihail Eric, Rahul Goel, Shachi Paul, Abhishek Sethi, Sanchit Agarwal, Shuyag Gao, and Dilek Hakkani-Tur. Multiwoz 2.1: Multi-domain dialogue state corrections and state tracking baselines. *arXiv preprint arXiv:1907.01669*, 2019.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan Dhanalakshmi Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. Improving text-to-sql evaluation methodology. In *ACL 2018*. Association for Computational Linguistics, 2018.

Stefan L. Frank. Uncertainty reduction as a measure of cognitive load in sentence comprehension. *Topics in cognitive science*, 5 3:475–94, 2013.

Shuyang Gao, Abhishek Sethi, Sanchit Agarwal, Tagyoung Chung, and Dilek Hakkani-Tur. Dialog state tracking: A neural reading comprehension approach. In *SIGDial*, 2019.

Alessandra Giordani and Alessandro Moschitti. Translating questions to sql queries with generative parsers discriminatively reranked. In *COLING (Posters)*, pages 401–410, 2012.

Todd J Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 31–40, 2007.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao pei Liu Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-sql in cross-domain database with intermediate representation. In *ACL*, 2019.

Tong Guo and Huilin Gao. Content enhanced bert-based text-to-sql generation. *ArXiv*, abs/1910.07179, 2019.

Izzeddin Gur, Semih Yavuz, Yu Su, and Xifeng Yan. Dialsql: Dialogue based structured query generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1339–1349. Association for Computational Linguistics, 2018. URL http://aclweb.org/anthology/P18-1124.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online, July 2020. Association for Computational Linguistics.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. *ArXiv*, abs/2002.08909, 2020.

John Hale. Uncertainty about the rest of the sentence. *Cognitive science*, 30 4:643–72, 2006.

Pei hao Su, Milica Gasic, Nikola Mrksic, Lina Maria Rojas-Barahona, Stefan Ultes, David Vandyke, Tsung-Hsien Wen, and Steve J. Young. On-line active reward learning for policy optimisation in spoken dialogue systems. *ACL*, 2016.

Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen. X-sql: reinforce schema representation with context. *ArXiv*, abs/1908.08113, 2019.

M. Heck, Carel van Niekerk, Nurul Lubis, Christian Geishauser, Hsien-Chin Lin, M. Moresi, and Milica Gavsi'c. Trippy: A triple copy strategy for value independent neural dialog state tracking. In *SIGdial*, 2020.

Charles T. Hemphill, John J. Godfrey, and George R. Doddington. The atis spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*, 1990. URL http://aclweb.org/anthology/H90-1021.

Matthew Henderson, Blaise Thomson, and Steve Young. Deep neural network approach for the dialog state tracking challenge. In *Proceedings of the SIGDIAL 2013 Conference*, pages 467–471, Metz, France, August 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W13-4073.

Matthew Henderson, Blaise Thomson, and Jason D. Williams. The second dialog state tracking challenge. In *SIGDIAL Conference*, 2014.

Matthew Henderson, Iñigo Casanueva, Nikola Mrkvsi'c, P. Su, Tsung-Hsien, and Ivan Vulic. Convert: Efficient and accurate conversational representations from transformers. *ArXiv*, abs/1911.03688, 2019.

Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *ArXiv*, abs/1610.02136, 2016.

Jonathan Herzig and Jonathan Berant. Decoupling structure and lexicon for zero-shot semantic parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1619–1629. Association for Computational Linguistics, 2018.

Jonathan Herzig, P. Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. Tapas: Weakly supervised table parsing via pre-training. In *ACL*, 2020a.

Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. Tapas: Weakly supervised table parsing via pre-training. *arXiv preprint arXiv:2004.02349*, 2020b.

Ehsan Hosseini-Asl, B. McCann, Chien-Sheng Wu, Semih Yavuz, and R. Socher. A simple language model for task-oriented dialogue. *ArXiv*, abs/2005.00796, 2020.

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia, July 2018. Association for Computational Linguistics.

Wonseok Hwang, Jinyeung Yim, Seunghyun Park, and Minjoon Seo. A comprehensive exploration on wikisql with table-aware word contextualization. *ArXiv*, abs/1902.01069, 2019.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2073–2083, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1195. URL https://www.aclweb.org/anthology/P16-1195.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettle-

moyer. Learning a neural semantic parser from user feedback. *CoRR*, abs/1704.08760, 2017.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Mapping language to code in programmatic context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2018.

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. Search-based neural structured learning for sequential question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1821–1831, Vancouver, Canada, July 2017. Association for Computational Linguistics.

Sina Jafarpour, Christopher JC Burges, and Alan Ritter. Filter, rank, and transfer the knowledge: Learning to chat. *Advances in Ranking*, 10:2329–9290, 2010.

Robin Jia and Percy Liang. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1002. URL https://www.aclweb.org/anthology/P16-1002.

Tsuneaki Kato, Jun'ichi Fukumoto, Fumito Masui, and Noriko Kando. Handling information access dialogue through qa technologies-a novel challenge for open-domain question answering. In *Proceedings of the Workshop on Pragmatics of Question Answering at HLT-NAACL 2004*, 2004.

Sung-Dong Kim, Sohee Yang, Gyuwan Kim, and S. Lee. Efficient dialogue state tracking by selectively overwriting memory. In *ACL*, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *The 3rd International Conference for Learning Representations, San Diego*, abs/1412.6980, 2015.

Walter Lasecki, Ece Kamar, and Dan Bohus. Conversations in the crowd: Collecting data for task-oriented dialog learning. In *In Proceedings of the Human Computation Workshop on Scaling Speech and Language Understanding and Dialog through Crowdsourcing at HCOMP 2013.*, January 2013. URL https://www.microsoft.com/en-us/research/publication/conversations-crowd-collecting-data-task-oriented-dialog-learning/.

Walter S. Lasecki, Mitchell Gordon, Danai Koutra, Malte F. Jung, Steven P. Dow, and Jeffrey P. Bigham. Glance: Rapidly coding behavioral video with the crowd. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, New York, NY, USA, 2014. ACM.

Ji Young Lee and Franck Dernoncourt. Sequential short-text classification with recurrent and convolutional neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 515–520, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1062. URL https://www.aclweb.org/anthology/N16-1062.

Roger P. Levy. Expectation-based syntactic comprehension. *Cognition*, 106:1126–1177, 2008.

Mike Lewis, Marjan Ghazvininejad, Gargi Ghosh, Armen Aghajanyan, Sida Wang, and Luke Zettlemoyer. Pre-training via paraphrasing. *arXiv preprint arXiv:2006.15020*, 2020a.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020b. Association for Computational Linguistics. doi:

10.18653/v1/2020.acl-main.703. URL https://www.aclweb.org/anthology/2020.acl-main.703.

Fei Li and HV Jagadish. Constructing an interactive natural language interface for relational databases. *VLDB*, 2014.

Jiwei Li, Will Monroe, Alan Ritter, Daniel Jurafsky, Michel Galley, and Jianfeng Gao. Deep reinforcement learning for dialogue generation. In *EMNLP*, 2016.

Yunyao Li, Huahai Yang, and HV Jagadish. Constructing a generic natural language interface for an xml database. In *EDBT*, volume 3896, pages 737–754. Springer, 2006.

Chen Liang, Mohammad Norouzi, Jonathan Berant, Quoc V. Le, and Ni Lao. Memory augmented policy optimization for program synthesis and semantic parsing. In *NeurIPS*, 2018.

P. Liang, M. I. Jordan, and D. Klein. Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL)*, pages 590–599, 2011.

Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system. In *LREC*, 2018.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. In *Findings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP Findings 2020, November 16th-20th, 2020*, 2020.

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomás Kociský, Fumin Wang, and Andrew Senior. Latent predictor networks for code generation. In *ACL (1)*. The Association for Computer Linguistics, 2016.

Q. Liu, B. Chen, Jiaqi Guo, Jian-Guang Lou, Bin Zhou, and Dongmei Zhang. How far are we from effective context modeling ? an exploratory study on semantic parsing in context. *ArXiv*, abs/2002.00652, 2020.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Qi Ju, Haotang Deng, and Ping Wang. K-bert: Enabling language representation with knowledge graph, 2019a.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke S. Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692, 2019b.

Reginald Long, Panupong Pasupat, and Percy Liang. Simpler context-dependent logical forms via model projections. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1456–1465. Association for Computational Linguistics, 2016.

Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, volume abs/1908.02265, pages 13–23, 2019.

Qin Lyu, Kaushik Chakrabarti, Shobhit Hathi, Souvik Kundu, Jianwen Zhang, and Zheng Chen. Hybrid ranking network for text-to-sql. Technical Report MSR-TR-2020-7, Microsoft Dynamics 365 AI, March 2020.

Kathleen R. McKeown. *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*. Cambridge University Press, New York, NY, USA, 1985. ISBN 0-521-30116-5.

Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 3111–3119, 2013.

Scott Miller, David Stallard, Robert J. Bobrow, and Richard M. Schwartz. A fully statistical approach to natural language interfaces. In *ACL*, 1996.

Sewon Min, Danqi Chen, Hannaneh Hajishirzi, and Luke Zettlemoyer. A discrete hard em approach for weakly supervised question answering. In *EMNLP*, 2019.

Lili Mou, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. Natural language inference by tree-based convolution and heuristic matching. In *ACL*, 2016.

Nikola Mrksic, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gasic, Pei hao Su, David Vandyke, Tsung-Hsien Wen, and Steve J. Young. Multi-domain dialog state tracking using recurrent neural networks. In *ACL*, 2015.

Nikola Mrkšić, Diarmuid Ó Séaghdha, Tsung-Hsien Wen, Blaise Thomson, and Steve Young. Neural belief tracker: Data-driven dialogue state tracking. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1777–1788. Association for Computational Linguistics, 2017.

Thomas Müller, Francesco Piccinno, Massimo Nicosia, Peter Shaw, and Yasemin Altun. Answering conversational questions on structured data without logical forms. In *EMNLP/IJCNLP*, 2019.

Arvind Neelakantan, Quoc V. Le, Martín Abadi, Andrew McCallum, and Dario Amodei. Learning a natural language interface with neural programmer. *ArXiv*, abs/1611.08945, 2016.

Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. Learning to generate pseudo-code from source code using statistical machine translation (t). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '15, 2015.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318. Association for Computational Linguistics, 2002.

Ankur P Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. Totto: A controlled table-to-text generation dataset. *arXiv preprint arXiv:2004.14373*, 2020.

Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1470–1480, 2015.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. *NIPS 2017 Workshop*, 2017.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543. ACL, 2014.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In Marilyn A. Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human*

Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers), pages 2227–2237. Association for Computational Linguistics, 2018.

Matthew E. Peters, Mark Neumann, Robert Logan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A. Smith. Knowledge enhanced contextual word representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 43–54, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1005. URL https://www.aclweb.org/anthology/D19-1005.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157. ACM, 2003.

Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Proceedings of the 20th international conference on Computational Linguistics*, page 141. Association for Computational Linguistics, 2004.

P. J. Price. Evaluation of spoken language systems: the atis domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*, pages 91–95, 1990.

Chris Quirk, Raymond Mooney, and Michel Galley. Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 878–888, 2015.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. Abstract syntax networks for code gener-

ation and semantic parsing. In *ACL (1)*, pages 1139–1149. Association for Computational Linguistics, 2017.

Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Nazneen Fatema Rajani, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, Yangxiaokang Liu, Nadia Irwanto, Jessica Pan, Faiaz Rahman, Ahmad Zaidi, Murori Mutuma, Yasin Tarabar, Ankit Gupta, **Tao Yu**, Yi Chern Tan, Xi Victoria Lin, Caiming Xiong, and Richard Socher. Dart: Open-domain structured data record to text generation. In *Submitted to North American Chapter of the Association for Computational Linguistics (NAACL)*, 2021.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. In *Technical report, OpenAI*, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140): 1–67, 2020.

Prajit Ramachandran, Peter Liu, and Quoc Le. Unsupervised pretraining for sequence to sequence learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 383–391, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

Antoine Raux, Brian Langner, Dan Bohus, Alan W. Black, and Maxine Eskénazi. Let's go public! taking a spoken dialog system to the real world. In *INTERSPEECH*, 2005.

Siva Reddy, Mirella Lapata, and Mark Steedman. Large-scale semantic parsing without

question-answer pairs. *Transactions of the Association for Computational Linguistics*, 2: 377–392, 2014.

Siva Reddy, Danqi Chen, and Christopher D. Manning. Coqa: A conversational question answering challenge. *CoRR*, abs/1808.07042, 2018.

Alan Ritter, Colin Cherry, and William B. Dolan. Data-driven response generation in social media. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 583–593, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics.

Ohad Rubin and Jonathan Berant. Smbop: Semi-autoregressive bottom-up semantic parsing. *arXiv preprint arXiv:2010.12412*, 2020.

Timo Schick and Hinrich Schütze. It's not just size that matters: Small language models are also few-shot learners. *arXiv preprint arXiv:2009.07118*, 2020.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1073–1083, 2017.

Stephanie Seneff and Joseph Polifroni. Dialogue management in the mercury flight reservation system. In *Proceedings of the 2000 ANLP/NAACL Workshop on Conversational Systems - Volume 3*, ANLP/NAACL-ConvSyst '00, pages 11–16, 2000.

Pararth Shah, Dilek Hakkani-Tür, Gokhan Tür, Abhinav Rastogi, Ankur Bapna, Neha Nayak, and Larry Heck. Building a conversational agent overnight with dialogue self-play, 2018.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume*

2 (Short Papers), New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? *arXiv preprint arXiv:2010.12725*, 2020.

Tianze Shi, Kedar Tatwawadi, Kaushik Chakrabarti, Yi Mao, Oleksandr Polozov, and Weizhu Chen. Incsql: Training incremental text-to-sql parsers with non-deterministic oracles. *arXiv preprint arXiv:1809.05054*, 2018.

Yu Su and Xifeng Yan. Cross-domain semantic parsing via paraphrasing. *CoRR*, abs/1704.05974, 2017.

Yu Su, Ahmed Hassan Awadallah, Madian Khabsa, P. Pantel, M. Gamon, and Mark J. Encarnación. Building natural language interfaces to web apis. *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017.

Alane Suhr, Srinivasan Iyer, and Yoav Artzi. Learning to map context-dependent sentences to executable formal queries. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2238–2249. Association for Computational Linguistics, 2018. URL http://aclweb.org/anthology/N18-1203.

Yibo Sun, Duyu Tang, Nan Duan, Jingjing Xu, X. Feng, and B. Qin. Knowledge-aware conversational semantic parsing over web tables. In *NLPCC*, 2019a.

Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Xuyi Chen, Han Zhang, Xin Tian, Danxiang Zhu, Hao Tian, and Hua Wu. Ernie: Enhanced representation through knowledge integration, 2019b.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Lappoon R Tang and Raymond J Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In *ECML*, volume 1, pages 466–477. Springer, 2001.

Gokhan Tur and Renato De Mori. *Spoken language understanding: Systems for extracting semantic information from speech*. John Wiley & Sons, 2011.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc., 2015.

Marilyn A. Walker, Alexander I. Rudnicky, Rashmi Prasad, John S. Aberdeen, Elizabeth Owen Bratt, John S. Garofolo, Helen F. Hastie, Audrey N. Le, Bryan L. Pellom, Alexandros Potamianos, Rebecca J. Passonneau, Salim Roukos, Gregory A. Sanders, Stephanie Seneff, and David Stallard. Darpa communicator: cross-system results for the 2001 evaluation. In *INTERSPEECH*, 2002.

Bailin Wang, Ivan Titov, and Mirella Lapata. Learning semantic parsers from denotations with latent structured alignments and abstract programs. In *Proceedings of EMNLP*, 2019.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.677. URL https://www.aclweb.org/anthology/2020.acl-main.677.

Chenglong Wang, Marc Brockschmidt, and Rishabh Singh. Pointing out sql queries from text. *Technical Report*, 2017a.

Chenglong Wang, Alvin Cheung, and Rastislav Bodik. Synthesizing highly expressive sql queries from input-output examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 452–466. ACM, 2017b.

Chenglong Wang, Po-Sen Huang, Alex Polozov, Marc Brockschmidt, and Rishabh Singh. Execution-guided neural program decoding. In *ICML workshop on Neural Abstract Machines and Program Induction v2 (NAMPI)*, 2018.

Yushi Wang, Jonathan Berant, Percy Liang, et al. Building a semantic parser overnight. In *ACL (1)*, pages 1332–1342, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1129. URL https://www.aclweb.org/anthology/P15-1129.

David HD Warren and Fernando CN Pereira. An efficient easily adaptable system for interpreting natural language queries. *Computational Linguistics*, 8(3-4):110–122, 1982.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina Maria Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve J. Young. A network-based end-to-end trainable task-oriented dialogue system. *CoRR*, 2016.

John Wieting and Kevin Gimpel. Pushing the limits of paraphrastic sentence embeddings with millions of machine translations. *arXiv preprint arXiv:1711.05732*, 2017.

Sam Wiseman, Stuart Shieber, and Alexander Rush. Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2253–2263. Association for Computational Linguistics, 2017. doi: 10.18653/v1/D17-1239. URL http://aclweb.org/anthology/D17-1239.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

Yuk Wah Wong and Raymond J. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-2007)*, Prague, Czech Republic, June 2007.

Chien-Sheng Wu, Andrea Madotto, Ehsan Hosseini-Asl, Caiming Xiong, Richard Socher, and Pascale Fung. Transferable multi-domain state generator for task-oriented dialogue systems. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Florence, Italy, July 2019. Association for Computational Linguistics.

Chien-Sheng Wu, Steven C.H. Hoi, Richard Socher, and Caiming Xiong. TOD-BERT: Pre-trained natural language understanding for task-oriented dialogue. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

Kun Xu, Lingfei Wu, Zhiguo Wang, Mo Yu, Liwei Chen, and Vadim Sheinin. Sql-to-text generation with graph-to-sequence model. *EMNLP*, 2018.

Xiaojun Xu, Chang Liu, and Dawn Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*, 2017.

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. Sqlizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages*, 1 (OOPSLA):63:1–63:26, October 2017. ISSN 2475-1421.

Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. In *ACL (1)*, pages 440–450. Association for Computational Linguistics, 2017.

Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. Neural enquirer: Learning to query tables in natural language. In *Proceedings of the Twenty-Fifth International Joint*

Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016, pages 2308–2314, 2016.

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. Learning to mine aligned code and natural language pairs from stack overflow. In *International Conference on Mining Software Repositories (MSR)*, 2018.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. Tabert: Pretraining for joint understanding of textual and tabular data. *arXiv preprint arXiv:2005.08314*, pages 8413–8426, July 2020. doi: 10.18653/v1/2020.acl-main.745.

Kang Min Yoo, Youhyun Shin, and Sang goo Lee. Data augmentation for spoken language understanding via joint variational generation, 2018.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. Typesql: Knowledge-based type-aware neural text-to-sql generation. In *Proceedings of the 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2018a.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. Syntaxsqlnet: Syntax tree networks for complex and cross-domain text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2018b.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018c.

Tao Yu, Rui Zhang, He Yang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao

Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing*. Association for Computational Linguistics, 2019a.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Irene Li Heyang Er, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Vincent Zhang Jonathan Kraft, Caiming Xiong, Richard Socher, and Dragomir Radev. Sparc: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy, 2019b. Association for Computational Linguistics.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. Grappa: Grammar-augmented pre-training for table semantic parsing. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2020.

Tao Yu, Rui Zhang, Oleksandr Polozov, Chris Meek, and Ahmed Hassan Awadallah. SCoRe: Pre-training for context representation in conversational semantic parsing. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2021.

John M. Zelle and Raymond J. Mooney. Learning to parse database queries using inductive logic programming. In *AAAI/IAAI*, pages 1050–1055, Portland, OR, August 1996. AAAI Press/MIT Press.

Luke Zettlemoyer and Michael Collins. Online learning of relaxed ccg grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in*

Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), 2007. URL http://aclweb.org/anthology/D07-1071.

Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *UAI*, 2005.

Luke S. Zettlemoyer and Michael Collins. Learning context-dependent mappings from sentences to logical form. In *ACL/IJCNLP*, 2009.

Jian-Guo Zhang, Kazuma Hashimoto, Chien-Sheng Wu, Yao Wan, Philip S Yu, Richard Socher, and Caiming Xiong. Find or classify? dual strategy for slot-value predictions on multi-domain dialog state tracking. *arXiv preprint arXiv:1910.03544*, 2019a.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. *arXiv preprint arXiv:1912.08777*, 2019b.

Li Zhang, Shuo Zhang, and Krisztian Balog. Table2vec: Neural word and entity embeddings for table population and retrieval. In *Proceedings of the 42Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, pages 1029–1032, New York, NY, USA, 2019c. ACM.

Rui Zhang, Tao Yu, He Yang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. Editing-based sql query generation for cross-domain context-dependent questions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing*. Association for Computational Linguistics, 2019d.

Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, and Bill Dolan. Dialogpt: Large-scale generative pre-training for conversational response generation. In *ACL, system demonstration*, 2020a.

Yusen Zhang, Xiangyu Dong, Shuaichen Chang, **Tao Yu**, Peng Shi, and Rui Zhang. Did you ask a good question? a cross-domain question intention classification benchmark for text-to-sql. In *Interactive and Executable Semantic Parsing Workshop at EMNLP*, 2020b.

Ruiqi Zhong, Tao Yu, and Dan Klein. Semantic evaluation for text-to-sql with distilled test suite. In *The 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2020a.

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017.

Victor Zhong, Caiming Xiong, and Richard Socher. Global-locally self-attentive encoder for dialogue state tracking. In *ACL*, 2018.

Victor Zhong, M. Lewis, Sida I. Wang, and Luke Zettlemoyer. Grounded adaptation for zero-shot executable semantic parsing. *The 2020 Conference on Empirical Methods in Natural Language Processing*, 2020b.