Abstract of "Bootstrapping Generalization in Neural Text-to-SQL Semantic Parsing Models" by Nathaniel Weir, Sc.B., Brown University, May 2019.

The focus of this thesis is to introduce a novel method for bootstrapping the generalizable language understanding of data-driven neural text-to-SQL translation models in unseen domains. Such models are of high interest because of their high performance as the central component of Natural Language Interfaces to Databases (NLIDBs). Current approaches rely on shallow, isolated supervised learning techniques in which models overfit to simple patterns from small, manually curated datasets in order to perform competitively on various text-to-SQL benchmarks. However, their learnt patterns are not extensible and lead to models with poor generalization beyond the specific language shown in the isolated training scenario. The models struggle to grasp the compositional nature of both natural language and SQL, and thus cannot output novel SQL patterns nor abstract learnt patterns in order to translate queries over databases in new domains. To address these shortcomings, we introduce a new training pipeline that leverages template-based synthetic training data generation in order to construct a more robust translation model without relying on manually-curated data in a target domain. Through evaluation, we show that this synthetic data serves a variety of purposes in multiple grounded scenarios, including an off-the-shelf single-domain case in which a target database schema is available before the training step, and a more general, cross-domain case in which the schema is not available until the translation step.

Bootstrapping Generalization in Neural Text-to-SQL Semantic Parsing Models

by
Nathaniel Weir
Sc.B., Brown University, 2019

A Thesis submitted in partial fulfillment of the requirements for Honors
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2019

This thesis by Nathaniel Weir is accepted in its present form by
the Department of Computer Science as satisfying the research requirement
for the awardment of Honors.

Date _____ _____

Ugur Cetintemel, Reader
(Dept. of Computer Science)

Date _____ _____

Elie Bienenstock, Reader
(Dept. of Applied Mathematics)

Acknowledgements

I would like to thank Ellie Pavlick, Ugur Cetintemel, and Carsten Binnig for their invaluable guidance for the past couple years on these projects. I'd also like to thank P. Ajie Utama, who served as my primary role model for NLP research and helped to shape the kinds of problems I find exciting and worth pursuing.

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Background

1.1 Motivation

Structured query language (SQL), despite its expressiveness, may hinder users with little or no technical exposure from exploring and making use of the data stored in an relational database. In order to effectively leverage their data sets, users are required to have prior knowledge about the schema of their database, such as entities and relations, as well as a working understanding of the syntax and semantics of SQL. These requirements set a high bar for entry for democratized data exploration and thus have triggered new efforts to develop alternative interfaces that allow non-technical users to explore and interact with their data conveniently and effectively. Natural Language Interfaces to Databases (NLIDBs) provide such a bridge across the expertise gap; they allow users to express their intentions with all the expressiveness of a natural language utterance, which is mapped directly to its corresponding, possibly non-trivially constructed, executable SQL query by a translation model. Figure 1.1 shows an example of a hospital database question[1] whose corresponding SQL is not easily inferred, particularly given the relative simplicity of the natural language. A medical professional who might want to query their hospital database in this way would otherwise need the technical experience to use a standard database interface and compose a complex SQL query, while an NLIDB would cut out the intermediary steps and allow the doctor to explore the data without any such overhead.

The basic architecture of a NLIDB is quite straightforward: given a database and natural language utterance, the NLIDB computes a SQL query that is executed against the database schema so as to retrieve a table visualization. A general architecture is shown in Figure 1.2.

The task of translating natural language (NL) text into SQL has gained much traction in both the database and NLP research communities. The former views it as an extension of the general task of Neural Semantic Parsing (NSP), which entails the use of deep models to convert ad-hoc natural language utterances into computationally tractable logical forms. In this scenario, SQL takes the

[1] In this work, we will use the term *question* to refer to a natural language utterance, while using the term *query* to denote the corresponding SQL form.

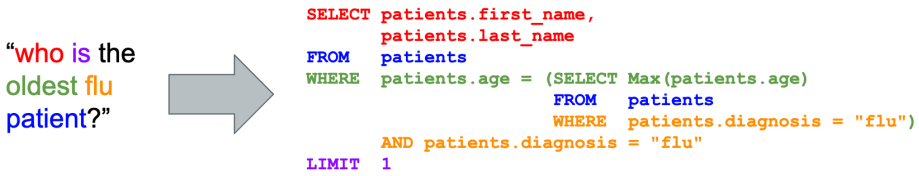


Figure 1.1: Example NL Query with Unintuitive Corresponding SQL Syntax

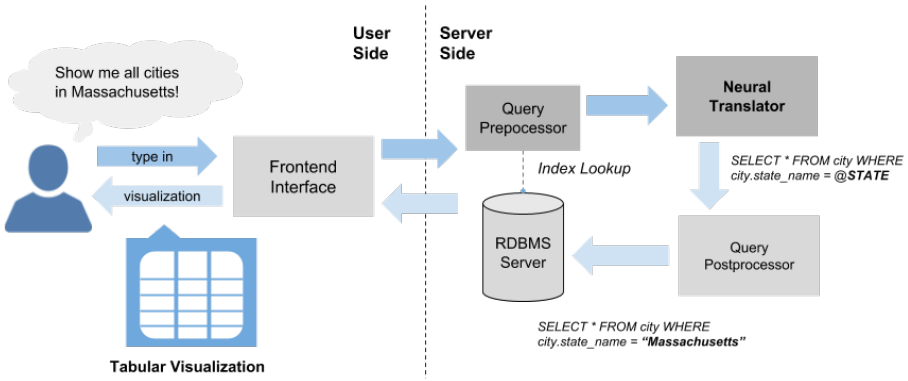


Figure 1.2: Overview of NLIDB Architecture

place of a typical logical form such as λ-calculus. Recent work has applied the de facto state-of-the-art model for NSP tasks, a bi-LSTM sequence-to-sequence (seq2seq) neural network [38], to the text-to-SQL task [37]. In this initial approach to neural text-to-SQL generation follows a typical deep learning proof-of-concept pipeline: the authors construct a set of manually-written examples of queries over a particular (single) database, split somewhat arbitrarily into train/validate/test sets, and upon training the model is assumed to automatically induct the linguistic and semantic patterns in both the source and target forms that will then be generalized onto the 'unseen' training set.

That seq2seq models drastically outperform previous statistical or rule-based approaches on this particular task without requiring the overhead of e.g. feature engineering is indeed an important accomplishment. However, successful performance in this supervised setting should not be conflated with successfully 'solving' the text-to-SQL semantic parsing task for a variety of reasons, some of which we will outline here as being the most important barriers keeping deep models from performing effectively in off-the-shelf NLIDBs for commercial use.

Firstly, the most critical assumption in the supervised learning paradigm is in its descriptor– that it is supervised. In other words, the model is assumed to have access to this well-curated dataset that contains a diverse set of language utterances and comprehensively reflects the type of linguistic patterns on which the model will eventually be 'tested.' Arbitrarily splitting up the curated examples into train/test sets implies that the vocabularies and semantic phenomena seen during train and evaluation times are essentially identical, as are the flavor of SQL queries that the model might need to output. A model being tested on the query 'what is the capital of Colorado?'

is quite likely to be successful if it has been previously trained on 'what is the capital of California?' and 'what is the capital of Connecticut?'. This assumed availability of applicable training data that sufficiently corresponds to the query patterns a model will have to translate online quickly falls apart, particularly in the case of an NLIDB, which will need to translate ad-hoc queries over whatever database its nontechnical user will be relying upon it to explore. Each new domain contains new ontological information, new notions of 'easy' queries to translate, and new ways to contextually ask for data. The overhead of crafting such examples– and writing the corresponding SQL–cannot be put upon the system user; this is contradictory to the purpose of a system geared towards helping nontechnical users.

The problems posed by diverse domains extend to the nature of the target databases themselves. Not only can domains differ ontologically– i.e. the difference between translating queries over a geographical versus a medical dataset–, but the database schemas are also variable within a particular context. Database entities and relations are defined at the whim of the dataset curator, and different schemata are possible given a particular ontology. For example, over one geographical schema, the utterance 'what is the capital of California?' might map to the query `SELECT state.capital FROM state where state.state_name = 'california'`, while over a second geographical schema it might translate to `SELECT capital.capital_name FROM capital WHERE capital.state_name = 'california'`. The knowledge of the particular target schema must therefore be injected, either into the collected training examples, or as we will see later, into the input of the neural network itself.

A second, somewhat related critical assumption of the supervised learning scenario is that a small, manually curated dataset over a single database will facilitate a model learning generalizable representations of language. The tendency of neural networks–particularly those such as bi-LSTMs with thousands of parameters– to overfit to limited training data portends a frustratingly brittle translation model that falls apart upon exposure to unseen language patterns, even seemingly simple ones or sequences that differ by only a word from those seen during training. The isolated nature of the training pipeline– i.e. that the network starts the process with a fully randomly initialized decision mechanism (barring the clever addition of e.g. fine tune-able, pre-trained word embeddings)– combined with a shallow objective function (essentially 'get these examples right and don't care about any others') lead to the model having no guaranteed agency to truly understand the semantics of language and instead relying on exploitable biases in the data. For example, during training the model might be exposed to the word 'how' in the context of 'how high is Mount Everest?'. If during evaluation it is asked 'how large is Alaska?', there is little chance it will handle the query effectively, regardless of whether it has seen other ways to ask for `SELECT state.area FROM state WHERE state.name = 'alaska'`. Moreover, it might see the utterance 'how long is the mississippi river?' and, exploiting that 'how' has only ever been seen in the context of mountain height, ignore any other semantic clues and output some SQL query that retrieves the height of Mount Mississippi.

From a philosophical NLP perspective, the network becoming brittle as a result of unnatural bias in the training data is incredibly worrying because it contradicts the role of language as a natural,

SQL: `SELECT COUNT(*) FROM patients WHERE diagnosis=flu`	
Canonical Utterance	What is the count of patients where diagnosis is flu ?
Syntactic Reordering	Of patients where diagnosis is flu, what is the count?
Morphological Modification	What is the count of patients diagnosed with flu ?
Mixed-Category Modification	What is the total sum of patients where diagnosis is being influenza ?
Implicitly Reference Database Element	How many patients with flu are there ?
Semantically Equivalent Phrase	Count flu-diagnosed patients

Table 1.1: Example of Categorically Reforming a Simple NL Utterance into Semantically Equivalent Phrases.

abstractable form of communication. Attempting to induct deep models of semantics leads to a failure to incorporate the common sense and linguistic principles that humans easily and unconsciously use to comprehend never-before-heard utterances.[2] For a particular SQL query over some schema, there are infinitely many ways to express a corresponding NL utterance, and all up to a high level of complexity can be easily handled by a human listener. An example of the ways to rephrase even simple a simple utterance is shown in Table 1.1. Even a simple typology of paraphrasing methods leads to an explosion of possible utterance, exhibiting the infinite expressiveness of natural language relative to SQL. There exists a need to incorporate robustness to such linguistic variety into neural models. Such robustness ideally corresponds to building a representation of language more faithful to human cognition, one that cannot be simply inducted from words in a single, isolated domain. Humans learn the meanings of words alongside seeing them in a diverse array of contexts and usages; hypothetically, a model that observes particular words in a similar way will construct more domain-general representations of them, with the understanding that they serve semantic roles outside of very particular, domain-specific patterns of questions.

For example, consider that in the queries 'what river runs through California?' and 'which athlete runs the 100-meter sprint?', the word 'runs' describes two distinct types of entity relationships; one might refer to a between-tables relationship ('river' and 'state' tables) while the other might refer to a link between a table and a column (the 'event' column of a 'runner' table). A general representation of 'runs' that acknowledges its multiple meanings and ontological roles as applied to novel database schemata is required to handle both cases–ideally, such a representation can be induced by showing 'run' to the model in multiple contexts during training.

Finally, it is just as important for a text-to-SQL translation model to grasp the compositional structure of its target language, SQL, as it is to do so for NL. SQL follows a specific syntax and grammar that defines what does and does not comprise a well-formed, executable query. A purely distributional, data driven supervised learner has no access to this structure, and as with the source

[2]There exists a large amount of work in the NLP community on exactly this problem, the search for ways to incorporate linguistic and cognitive theory into the distributional semantics upon which neural networks rely for language understanding. It is hotly debated whether such a learning paradigm, in which representations of language are crammed into a 'vector bottleneck' from which the connectionist decision process is derived, has the potential to 'solve language' to a sufficient extent. It is important to keep this debate in mind as we explore one way to augment the typical, controversially simple learning pipeline so as to make incremental progress towards this larger goal.

0. Full SQL Query	`SELECT Max(state.area) FROM state WHERE state.country = 'usa'`
1. Literals Anonymized	`SELECT Max(state.area) FROM state WHERE state.country = value`
2. Literals + Entities Anonymized	`SELECT Max(table1.col1) FROM table1 WHERE table1.col2 = value`
3. General SQL Pattern	`SELECT func(table1.col1) FROM table1 WHERE table1.col2 comp value`

Table 1.2: Illustration of Differing Levels of SQL Pattern Abstraction

language, it is assumed that the model will derive this system from examples alone. Again, given a naive training split of a couple thousand examples or fewer, this is a tall task. The famous semanticist Gottlob Frege argued that extracting semantic meaning is akin to capturing a tree of logical forms that explains interrelation and function between concepts. This is literally the case for SQL, where meaning is directly convertible to a tree structure. Until recently, present-day neural semantic parsing models specifically have not captured this spirit– they capture meaning only in the latent patterns of data pieces; they are not forced to perceive a grammar or typography of functions upon which they can understand any ad-hoc utterance or query pattern. Because they can not generalize well to unseen domains, this lack of structured perception is inherently limiting the capacity of the model. In contrast, non-deep, statistical semantic parsing models from previous decades *are* exposed to grammatical structure and so can make less context-isolated decisions, though these decisions do not have the ease nor computational complexity of data-driven approaches.

There remains a need to reincorporate such structure. When one abstracts away a level or two of specificity by removing named database elements (i.e. in Figure 1.2), it becomes evident that the small training set curated by one or a couple authors is bound to cover only a small subset of SQL query patterns. Upon evaluation of supervised models, it becomes evident that a network is incredibly unlikely to ever output a SQL pattern of level-2 abstraction that was not observed during training, even if it is a simple recombination of seen patterns. If the model has seen queries (1a) and (1b) during training, it is still very unlikely to ever output query (1c) despite the relatively simple compositional operation required to construct it from the seen sequences. While the seq2seq model is generative and thus outputs the 'most likely sequence' given the input utterance, in the supervised paradigm the translation process is instead essentially one of classification, with possible classes comprising only the level-2 abstractions observed during training, with the only generative part of the procedure being the filling in of literals and entities.

(1a) `SELECT * FROM state WHERE state.country = 'usa'`

(1b) `SELECT Max(state.area) FROM state`

(1c) `SELECT Max(state.area) FROM state WHERE state.country = 'usa'`

The use of a modular, grammar-aware neural decoder by [44] as discussed in future sections is a step towards reincorporating compositionality so as to avoid this drawback, as it leverages structure that is known to exist so as to not need the black-box decision process of the neural net to induce understanding of the structure on its own. However, patterns not available in the training set for a

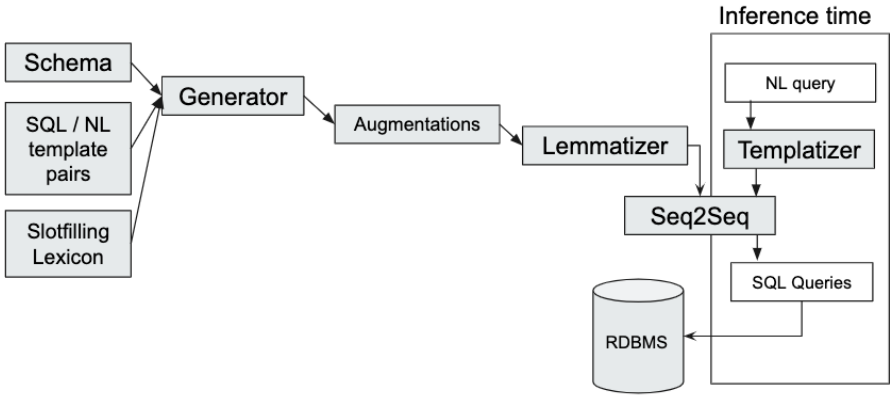


Figure 1.3: Overview of Synthetic Data Generation, Training and Inference Pipeline

particular domain are still unlikely to be outputted, even with with awareness. As a result, there remains a need to inject into the pipeline the possibility of outputting unseen patterns.

1.2 Contribution

The main contribution of this work addresses the issues stemming from the training data bottleneck from a novel angle. We introduce a bootstrapping approach for training a neural NLIDB on any new database schema in a domain-agnostic way with no reliance on manually-curated training examples. Using a set of over 100 domain-independent seed templates, our technique generates vast amounts of domain-specific training examples of natural language utterances paired with their corresponding SQL queries. Each SQL pattern in the grammar of templates is matched with many distinct NL patterns. These NL and SQL templates contain slots to be filled in by database elements, while the NL templates also contain phrase slots to be filled in using a back-end English lexicon. The set of SQL patterns covered includes aggregations, simple nested queries and column joins.

The set of templates is designed to define a set of universal SQL templates over which the trained model will perform robustly to domain-general utterances and their linguistic variation. We further add variety and domain specificity to the training data through a number of augmentation steps that draw inspiration from various works in computer vision and machine translation, such as random word dropout and automatic paraphrasing. The produced training data leads to a trained model that is more robust to linguistic variation than its predecessors.

This synthesis of training data using the principle of *distant (or weak) supervision* [10, 11][3]. The basic idea of distant supervision is to leverage various heuristics and existing datasets to automatically generate large (potentially noisy) training data instead of handcrafting it.

We initially present our translation model training framework as a working end-to-end system, which requires minimal manual annotation effort by a user who is not well-versed in SQL–it only

[3]We do not distinguish between weak and distant supervision as the difference is subtle and both terms are often used interchangeably in the literature.

requires schema elements to be paired with simple natural language utterances. An overview of a full pipeline architecture is illustrated in Figure 1.3.

Our approach uniquely expands upon the typical supervised learning framework of previous models. We argue that our use of extensive linguistically-aware templates to bootstrap training can provide a comparable breadth of coverage as that of manually collected training data. Furthermore, we argue that our techniques for augmenting the generated data further increase the linguistic robustness and domain adaptability of any model trained wholly or partially on it.

We proceed to show that the generated data serves a variety of purposes beyond serving as sufficient training data for an end-to-end translation model. The data can also be used to augment the training pipeline of other translation models in separate scenarios than an off-the-shelf system that plugs into an input schema and trains a new seq2seq model to translate queries over it. To illustrate this, we inject the data into the pipeline of the work, SyntaxSQLNet [44], a general text-to-SQL approach that is geared towards translating queries over multiple databases with a single model.

SyntaxSQLNet is unique in that it can combine manually-curated training pairs from other, available domains and use them as a universal training set so as to generalize onto new target databases without having to be retrained nor having to even see the target schema before translating questions over them. We can formalize this difference via notation: the initial, off-the-shelf application constructs a new translation model with parameters θ_d using training queries only over given database d, and then during translation it computes $\text{Translate}_{\theta_d}(q)$ to generate the SQL query. This new scenario entails the SyntaxSQLNet model training on queries over many database schemata to acquire parameters θ, and then during translation computing $\text{Translate}_{\theta}(d, q)$.

We demonstrate how adding synthetic data to this second scenario boosts the generalization of the SyntaxSQLNet model onto new databases, as well as how it boosts the ability of the model to grasp the compositionality of SQL in a single domain.

1.3 Related Work

1.3.1 NLIDBs

The task of synthesizing an SQL program from natural language text has been studied extensively within both the NLP and database research communities since the late 1970s [32, 46]. A 1995 study from [1] extensively discusses challenges that need to be address pertaining to NLIDBs; their list includes linguistic coverage and database portability.

In the database community, notable NLIDBs such as NaLIR [24] define a rule-based system over the syntactic structure of the NL utterance (i.e. using part-of-speech tags [35] or dependency relations [24]). The latter also allows users to provide feedback to the system by correcting mappings of utterance tokens to schema elements (table names, attributes or record values). Unfortunately, these approaches are inherently limited in terms of linguistic coverage. The reliance on a finite set of handcrafted rules and the output of off-the-shelf dependency parsers lead the systems to fail at

extrapolating its decision rules beyond a test set of linguistically and syntactically similar queries in a single domain. In the case of [24], as shown in our results in Section 2.2.4, simple lexical or syntactical changes to an otherwise easily translatable utterance are enough to incapacitate their translator.

A different type of approach introduced by [36] leverages domain-specific ontological structures entailing relationships and semantic roles of database elements in order to significantly outperform its predecessors. The system first converts an input NL utterance into an intermediate query of the ontological structure before converting the intermediate form into SQL. While its performance on common benchmarks has yet to be matched, this approach requires extensive manual overhead to craft a new ontology for a given database schema and thus is not realistic for off-the-shelf use.

Within the NLP community, this task is most commonly treated as a semantic parsing problem – one where the goal is to model a mapping of natural language text to corresponding logical form, in this case SQL. Earlier works such as [4, 3, 25, 48] employ variants of CCG parsers [9] to parse a natural language utterance into an executable lambda calculus notation.

Recent success in employing neural network sequence-to-sequence modelling for syntactic constituency parsing by [41] has spurred efforts in adapting the same solution for semantic parsing. That is, they pose logical form synthesis as a neural machine translation task, adapting systems for translating English to Czech or French to instead treat the logical form as the target foreign language. In both settings, mapping to lambda calculus [23, 13] or directly to SQL [20, 15, 6], the sequence-to-sequence architecture has shown highly competitive performance with statistical approaches that rely heavily upon hand-crafted lexical features.

Sequence-to-sequence models consist of a large number of parameters that require vast amounts of training examples. This poses a substantial challenge, as collecting diverse enough training data comprising pairs of utterance and logical form or SQL requires expensive expert supervision. Iyer et al. [20] attempts to deal with this data bottleneck by performing an online learning mechanism in which the model alternates between training and making predictions. Human judges identify incorrect predictions that need to be corrected by a crowdsourced worker with SQL expertise.

Alternatively, a solution more similar to ours is introduced by [42], whose approach produces pairs of canonical utterances aligned with their corresponding logical forms using a seed lexicon. They again use crowdsourcing, in this case to build a more diverse set of NL training examples by paraphrasing the canonical utterances into more fluent sentences using syntactic alterations and context specific predicates. While less efficient than an on-the-fly system, this form of crowd-sourced annotation is much less costly than previous approaches, given worker's SQL expertise is not required.

The main contribution of this work addresses the training data bottleneck from a slightly different angle. We attempt to perform utterance generation in a technique completely detracted from any manual annotation effort by a user who is not well-versed in SQL. Rather, the user need only be familiar with the given new semantic domain in order to sufficiently annotate the new schema's elements with their natural language utterances.

1.3.2 Towards Generalization

Text-to-SQL benchmarks have only very recently been crafted to encourage models to address the problems we enumerated above. A recent analysis of previous benchmarks [7] codifies the problems of generalization and compositionality in NL-to-SQL translation. For each dataset previously used for evaluating recent systems [29, 24, 37, 17], they compiled statistics measuring set redundancy–that is, how many actual NL question patterns the set contains when database literals are removed[4]–, as well as how complex the SQL query patterns covered by the set are. They stress the need to evaluate approaches upon multiple domains with varying levels of redundancy and complexity, citing the diverse methods and motivations of the NLP and databases communities when generating the datasets. They also introduce new train/test splits for existing benchmarks in order to test for compositional SQL generalization; all SQL patterns seen during evaluation are not available in training, thus forcing models to recombine the patterns available to them. In their evaluation, no models performed at all effectively on these new splits, with drops of up to 70% in accuracy on some datasets.

The new benchmark, Spider [45], follows in this trend, expanding the domain of test datasets into the hundreds (leveraging the manual labor of a dozen or so undergraduate annotators). The high number of domains allows them to also introduce a test split *over databases*, thus testing over diverse sets of queries of databases that haven't been seen at all during training.

The curators of Spider also introduce the model, SyntaxSQLNet [44], that currently has state-of-the-art performance on the new benchmark. It differs from previous models in that it encodes information about a database as part of its input, thus allowing it to recognize new domains on the fly. It follows the same encoder-decoder framework that most sequence-to-sequence models employ, but uses a novel, modular decoder inspired by [34] that leverages the grammatical structure of SQL to make explicit the decision process for outputting well-formed queries with attention to database elements. However, this model achieves only 24.8% accuracy on the new benchmark. While this is a massive improvement over previous systems (namely, because no other system is designed to operate over multiple databases as input), there is still obviously extensive room for improvement. We explore this model further in Section 3.1.

Similar to our approach, the training pipeline of [44] also leverages synthetic data generation. Our approach differs from theirs in that they use a far less extensive set of query templates that only cover single-table operations, while ours cover multi-table join queries. Their set of query templates is also not geared towards boosting linguistic robustness and has a very limited variety of NL queries that can be generated for each SQL pattern.

In the field of semantic parsing, progress has also been made towards the induction of extensible representations by training in multiple domains. [19] show that neural semantic parsing models trained on the text-to-λ-calculus task benefit from sharing parameters across multiple semantic domains. Parsing accuracy is boosted in every single source domain, as well as in unseen target

[4]For example, 'show the capital of California' and 'show the capital of Missouri' follow the same question pattern 'Show the capital of STATE'

domains. [14] build upon this multi-domain framework but experiment with a broader form of transfer learning, where they have access to extensive labeled data in a somewhat related task domain–in their case, syntactic parsing– and use it to increase training on the task with less available data.

1.4 Outline

The rest of this work is outlined as follows.

Chapter 2 presents our work on generating a rich, linguistically robust training set over a target database schema. We first describe the generation process (Section 2.1), in which examples covering a wide set of SQL patterns are generated using over 100 template pairs of NL/SQL templates. We then set up a series of evaluation experiments (Section 2.2). We evaluate a trained bidirectional sequence-to-sequence model (described in Section 2.2.1) on a pair of benchmarks, the crafting of the first of which we briefly describe (Section 2.2.2). We compare performance against other recent systems and demonstrate that it can perform competitively with supervised learning models and vastly outperform a well-known rule-based one. Finally, we discuss the strengths and weaknesses of our approach (Section 2.3). This chapter covers work previously published in [39].

Chapter 3 presents our follow-up work on applying synthetic data generation to the pipeline of recent cross-domain text-to-SQL model, SyntaxSQLNet. We first briefly describe the architecture and training pipeline of the SyntaxSQLNet paper (Section 3.1), and then our different approaches to injecting our data to the pipeline (Section 3.3). We then evaluating it on recent benchmarks and demonstrate how our best approach enables both cross-domain generalization (Section 3.4). We proceed with follow-up experiments on this setup to evaluate the impact on domain adaption of training data magnitude (Section 3.5) and number of source domains (Section 3.6). Finally, we evaluate the impact of synthetic data on compositional SQL generalization (Section 3.7).

Chapter 4 concludes with a summary of Chapters 2 and 3 and a general discussion of future work.

Chapter 2

Synthetic Text-to-SQL Training Data Generation

In the following, we first discuss the details of our data generation approach which consists of two steps: data instantiation and data augmentation. Afterwards, we discuss how generation is incorporated into training a model compare the model's performance with recent approaches.

2.1 Generation Process

We describe our generation process by relating it to the approach of 'Building a Semantic Parser Overnight' [42], who address the challenge of automatically curating a semantically and linguistically diverse training set for querying a knowledge-base on the fly. It is currently nearly impossible to follow some set of codified logical steps in order to produce fully-formed natural language, given its infinite richness as discussed above. In order to address this, [42] separate their training set generation into two distinct steps:

1. **Logical Form and Canonical Utterance Generation**, in which a domain-general grammar containing logical form templates is used to repeatedly instantiate possible queries over the knowledge base. For each instantiation, a corresponding 'canonical utterance' is generated using slotfilling with a 'seed lexicon' that maps knowledge base elements to NL words and phrases. The canonical utterance very closely resembles the syntax of the target logical sequence and would thus leave much to be desired were it used to train a semantic parser– i.e. the parser would only ever be able to translate NL that very closely resembles the target logical form.

2. **Augmentation via Crowdsourced Paraphrasing**, in which the canonical natural language is given to a crowdsourced worker to be modified into a natural and more linguistically interesting question. The resulting training set thus contains diverse and fully-formed natural

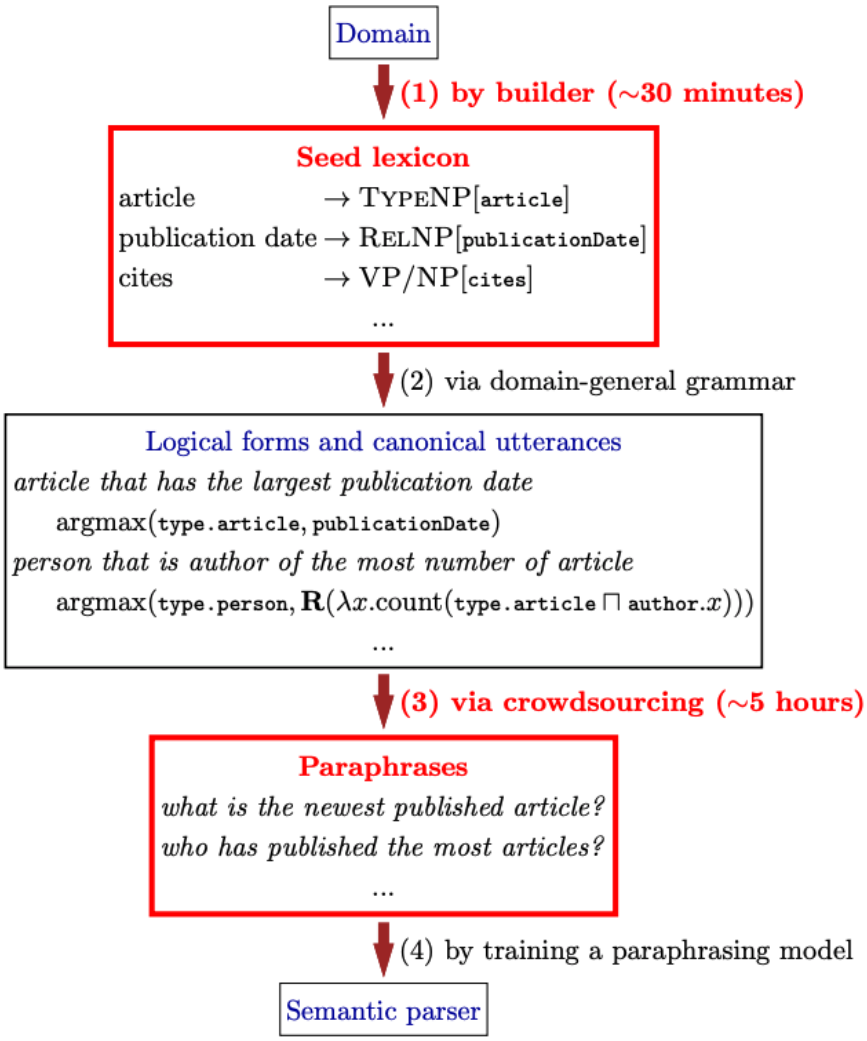


Figure 2.1: Training Set Generation Process from 'Building a Semantic Parser Overnight'

language examples that will yield a much more robust translation model.

This pipeline is summarized in Figure 2.1, borrowed from [42]. This work demonstrates that it is possible to learn to translate queries into logical forms in domains without any manually-curated data, aside from the passive use of crowdsourcing, a hybrid artificial/manual resource. It is functionality-driven, in that the curator defines the set of possible queries that can be produced and then sets about creating robust translation over that set.

2.1.1 Template-based Example Instantiation

The separation between template-based slot-filling and augmentation for natural richness also appears in our generation approach, which is summarized in Figure 2.2. A major difference, however,

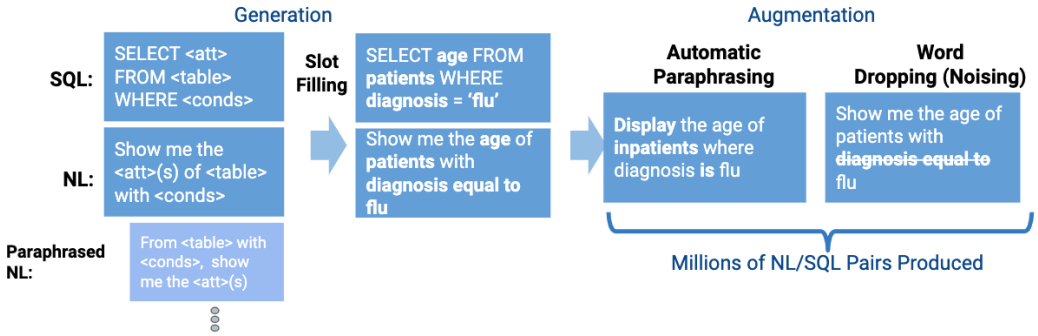


Figure 2.2: Training Set Generation Process in Our Approach

is that we attempt to incorporate some of the natural richness into the first step by generating a much more substantial lexicon and set of templates that extends beyond 'canonical' NL that closely resembles the target SQL.

To exemplify this, consider the following SQL template from our grammar:

```
SELECT {FUNC1} ({TABLE1}.{COL1}) FROM {TABLE1}
        WHERE {TABLE1}.{COL2} {COMP} {TABLE1}.{COL2}.{LITERAL}
```

Each token surrounded by `{}`'s represents a slot to be filled by database entities or, in the case of `{FUNC1}` and `{COMP}`, a sql-specific vocabulary term. The canonical NL translation that most closely resembles this SQL template is

```
{commandToken} the {FUNC1} {TABLE1}.{COL1} {fromToken} {TABLE1}
    {whereToken} {TABLE1}.{COL2} {COMP} {TABLE1}.{COL2}.{LITERAL}
```

where slots such as `{commandToken}` and `{fromToken}` are randomly filled in using an extensive backend lexicon–for example:

```
{commandToken} => show|get|list|find|show me|get me|show me|display|exhibit
```

However, as opposed to the approach of [42], A SQL template is not associated with just a single, canonical NL template. We categorically perform syntactic re-ordering to associate multiple NL templates whose wording doesn't necessarily align so nicely with the SQL. During generation, many training examples are generated from each one of the NL seed templates. Example re-ordered templates and corresponding NL instantiations are displayed in Table 2.1. This variety serves to inject a sense of possible grammatical variation into the model's understanding of NL queries. Note that instantiation can result in noisy, less than grammatically-correct utterances, as well as utterances that are unlikely to be asked by a naive user. However, we argue that perfect adherence to grammatical constraints shouldn't be the goal of a semantic parser– such utterances still have semantic intent that corresponds to the target SQL query, and could easily be understood by a human listener. Moreover, as discussed above, we believe the larger variety of contexts and instances in which a NL word appears will make it less likely to be the object of artificial biases.

SQL Template:	`SELECT {FUNC1} ({TABLE1}.{COL1}) FROM {TABLE1} WHERE {TABLE1}.{COL2} {COMP} {TABLE1}.{COL2}.{LITERAL}`
Instantiated:	`SELECT Max(patients.age) FROM patients WHERE patients.diagnosis = 'flu'`
Canonical NL Template:	`{commandToken} the {FUNC1} {TABLE1}.{COL1} {fromToken} {TABLE1} {whereToken} {TABLE1}.{COL2} {COMP} {TABLE1}.{COL2}.{LITERAL}`
Instantiated:	`Find the max age from patients where diagnosis equals flu`
NL Template:	`{fromToken} {TABLE1} {whereToken} {TABLE1}.{COL2} {COMP} {TABLE1}.{COL2}.{LITERAL}, {questionToken} the {FUNC1} {TABLE1}.{COL1}`
Instantiated:	`Out of patients whose diagnosis is flu, what is the highest age`
NL Template:	`{commandToken} {fromToken} {TABLE1} {whereToken} {TABLE1}.{COL2} {COMP2} {TABLE1}.{COL2}.{LITERAL} the {FUNC1} {TABLE1}.{COL1}`
Instantiated:	`Display out of patients with diagnosis being flu the highest-valued age`

Table 2.1: Example of NL/SQL Templates with Instantiation

2.1.2 SQL Coverage

Like [42] we take a functionality-driven approach to the training pipeline. Our SQL templates define a subset of all possible SQL patterns that can reliably be produced by our system. These include group-by aggregations, column selects, nests, inner joins, and 'macro' operations such as argmax and argmin. Many of these functionalities are templatized by automatically reforming simpler templates. We describe a couple examples of automatic template extension here.

GROUP BY Statements: The SQL template from Table 2.1 is reformed into a group-by template with simple prefixes and suffixes added to the SQL and NL seeds:

```
SELECT {COLagg}, {FUNC1} ({TABLE1}.{COL1}) FROM {TABLE1}
        WHERE {TABLE1}.{COL2} {COMP} {TABLE1}.{COL2}.{LITERAL} GROUP BY {COLagg}

For each {COLagg}, {commandToken} the {FUNC1} {TABLE1}.{COL1} {fromToken} {TABLE1}
    {whereToken} {TABLE1}.{COL2} {COMP} {TABLE1}.{COL2}.{LITERAL}
```

JOIN Statements: Similarly, join query templates can be inferred by letting `{TABLE1}` tokens vary by number, thus allowing the table in the `FROM` clause differ than the table in the `WHERE` clause[1]:

```
SELECT {FUNC1} ({TABLE1}.{COL1}) FROM {TABLE1}, {TABLE2}
        WHERE {TABLE2}.{COL2} {COMP} {TABLE2}.{COL2}.{LITERAL} AND
```

[1]Note that this simple example only allows for joins between tables that have a single-table join path; more complicated join paths require more clauses to be substituted in, but are still templatizable.

```
                {TABLE1}.JOIN_COL{TABLE2} = {TABLE2}.JOIN_COL{TABLE2}

{commandToken} the {FUNC1} {TABLE1}.{COL1} {fromToken} {TABLE1}
    {whereToken} {TABLE2}.{COL2} {COMP} {TABLE2}.{COL2}.{LITERAL}
```

For clarity, this template pair produces instantiations such as:

```
SELECT Min(city.population) FROM city, state
    WHERE city.state_name = state.state_name AND state.country = 'usa'

Display the population from cities where country is usa
```

Nested ARGMIN/MAX Statements: An example of a templatizable nested 'argmax' query is depicted in Figure 1.1. Such a query can be created by replacing or appending any `WHERE` clause with a placeholder sequence representing an argmax; in this case, a macro taking in the tuple `(column to take min/max of $ source table (can be joined) $ any inner WHERE clauses)` to be replaced by complete SQL during a post-processing step. The result of applying this to our initial example template is as follows:

```
SELECT {FUNC1} ({TABLE1}.{COL1}) FROM {TABLE1} WHERE
    ARGMAX({TABLE1}.{COLf} $ {TABLE1} $)

{commandToken} the {FUNC1} {TABLE1}.{COL1} {fromToken} {TABLE1}
    {whereToken} {TABLE1}.{COLf} is the {ARG1}.
```

For clarity, this template pair produces instantiations such as:

```
SELECT Min(patients.age) FROM patients WHERE
    ARGMAX(patients.length_of_stay $ patients $ )
```

Note that upon instantiation this query pattern can be either converted into the fully-formed SQL query (as `ARGMAX` is not a keyword and would obviously cause an interpreter to crash), or can be left as-is so that a seq2seq model learns to output `ARGMAX`'s instead of outputting the longer SQL statement. Our implementation chooses the latter option.

It is also important to note that it is trivial to continue to add these templates to the back-end grammar before generation. While the system is geared towards naive users, were a SQL-aware user to want to increase the breadth of SQL coverage for their translation model, they could simply add a template to cover it.

2.1.3 Training Data Augmentation

In order to make the query translation model further robust to linguistic variation, we apply the following augmentation steps for each instantiated NL-SQL pair, each of which is performed fully

automatically without the use of crowdsourcing:

Automatic Paraphrasing: We augment the training set by duplicating NL-SQL pairs, but randomly selecting words and sub-clauses of the NL query and paraphrasing them using the Paraphrase Database (PPDB) [30] as a lexical resource. A simple example for this is:

Input NL Query:
Show the name of all patients with age 40

PPDB Output:
demonstrate, showcase, display, indicate, lay

Paraphrased NL Query:
display the names of all patients with age 40

PPDB is an automatically extracted database containing millions of paraphrases in 27 different languages. PPDB provides over 220 million paraphrase pairs, consisting of 73 million phrasal and 8 million lexical paraphrases, as well as 140 million paraphrase patterns, which capture many meaning-preserving syntactic transformations[2]. The paraphrases are extracted from bilingual parallel corpora totalling over 100 million sentence pairs and over 2 billion English words.

In our automatic paraphrasing step, we use PPDB as an index that maps words/sub-phrases to paraphrases and replace words/sub-phrases of the input NL query with available paraphrases. For any given input to PPDB, there are often tens or hundreds of possible paraphrases available.

Random Word Removal: A possible challenge of input NL queries is missing or implicit information. For example, a user might ask for `patients with flu` instead of `patients diagnosed with flu` and thus the information about the referenced attribute might be missing in a user query.

Therefore, to make the translation more robust against missing information, we duplicate individual NL-SQL pairs and select individual words/sub-clauses that are removed from the NL query. This augmentation also serves to increase robustness to fragmented queries, an important quality as mentioned above.

The above approaches parallel ones seen in computer vision, in which image training examples are modified by changing pixel values or removing subsections so as to create a new example but retain the essential visual information contained within– in our case, we perform noisy modifications that ideally retain the semantics of the input phrase[3].

Other Augmentations: We also apply additional techniques to increase linguistic variations contained in the training set. One example is randomly sampling from available linguistic dictionaries for comparative and superlative adjectives. That way, we can replace for example the general phrase *greater than* in an input NL query by *older than* if the domain of the schema attribute is set to *age*.

[2]We do not implement the syntactic transformations (yet).

[3]It may be the case that the sementics *are* changed, as in the case where `show patients with age not 40` becomes `show patients with age 40`. However, we argue that this contrived example hinders the model less severely than it helps to make the model less brittle to particular question patterns.

In the future, we plan on extending the augmentation phase; we have explored automated paraphrasing via 'language pivoting' [27], in which an utterance is fed through a neural machine translation model into a foreign language, e.g. German, and then through a second model back into English. We have also explored NL generation via backtranslation from SQL into NL– that is, to generate a SQL instantiation and then feed it through either a rule-based ([22]) or neural model for summarizing SQL queries using natural language.

2.2 Evaluation

2.2.1 Training Our Model

We use the standard architecture of a seq2seq model as described in [38] for machine translation tasks. It comprises two recurrent neural networks (RNNs), an encoder and decoder. We use the bidirectionality proposed by [2]. Both encoder and decoder consist of two layers of gated recurrent units (GRUs) [8]. An overview of this setup is shown in Figure 2.3.

The dimensions of the hidden state vectors are set to 500, and we use GoogleNews word2vec 300-dimension pretrained word embeddings ([28]). We apply a dropout of .5 between the two GRU layers as an attempt to avoid overfitting to the training corpus, which, though to a lesser extent than many manually-curated sets, consists of a relatively small vocabulary (i.e., the backend lexicon, SQL keywords, and schema annotation for the given database).

It is important to note the size of the training sets synthesized by our generator. Even for a simple, single-table database with less than 10 columns, the space of possible NL/SQL pairs that can be produced via recursive slot-filling is in the billions. For multiple-table queries, this size increases a couple orders of magnitude. In order to make training have a realistic duration, we apply probabilistic filtering to the recusrive algorithm. While the unfiltered algorithm fills a slot and creates new recursive branches for every possible lexical item available, at high levels of recursion we prune the branches by selecting fewer items from the candidates available in the lexicon. As a result, we generate 1-2 million queries over a single-table schema and 3-4 million over a multi-table one.

Given a set of synthetic training data generated from an input schema, we first apply stemming and lemmatizing using the NLTK toolbox [26]. We follow a typical train/validation split of the synthetic data, where the validation set is used for hyperparameter tuning. We use Adam [21] with a learning rate of 0.0005 for 15 epochs.

2.2.2 Benchmarks

In order to compare our approach to other baselines, we use the GeoQuery benchmark, a well-known semantic parsing benchmark adapted nl-to-sql task ([33], [18], [20]). For testing different linguistic variants in a principled manner, we also curate a new benchmark that covers different linguistic variations for the user NL input and maps it to an expected SQL output. The benchmark

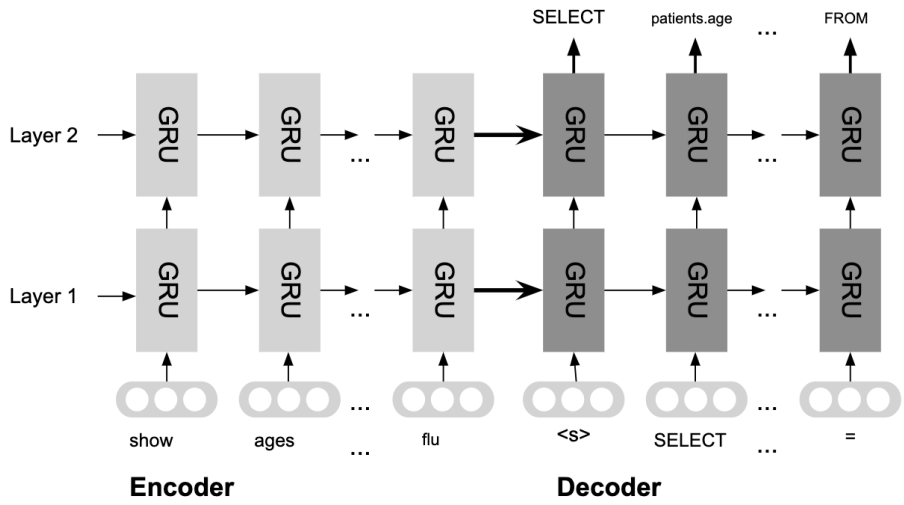


Figure 2.3: Overview of Text-to-SQL Sequence-to-Sequence Architecture

is available online [4].

Here we give an overview of GeoQuery and then present the design of this new benchmark, the Patients dataset.

GeoQuery Benchmark: The GeoQuery benchmark first introduced by Mooney [46] contains 880 queries over a database containing geographic information about the United States. The database consists of 7 tables which represent geographical entities such as US states, cities, rivers and mountains. The schema of this database is shown in Figure 2.4

The target logical form was initially written in the logic programming language Prolog. We use the version converted to SQL ([33], [18], [20]) following the logical form data split from [47].

In addition to queries over individual tables, this benchmark includes more complex queries such as joins, nested queries, and questions that are in general semantically harder than single-table queries with and without aggregation.

For the sake of comparison, our approach was evaluated using the same 280-pair testing set as [20], who used the other 600 for training and validation. To better understand the nature of the test queries, we categorized a large amount of the testing set into various classes of queries:

- 44 queries using the SQL *IN* operator, generally for comparison of values across two or more tables.
- 11 queries requiring aliases and the SQL *as* operator.
- 91 'argmin' or 'argmax' queries such as `get the state with the highest population`. As discussed above, the SQL form generally follows the pattern `SELECT ... FROM {TABLE1} WHERE {COL} = (SELECT max({COL}) FROM {TABLE2})`.

[4] `https://datamanagementlab.github.io/ParaphraseBench/`

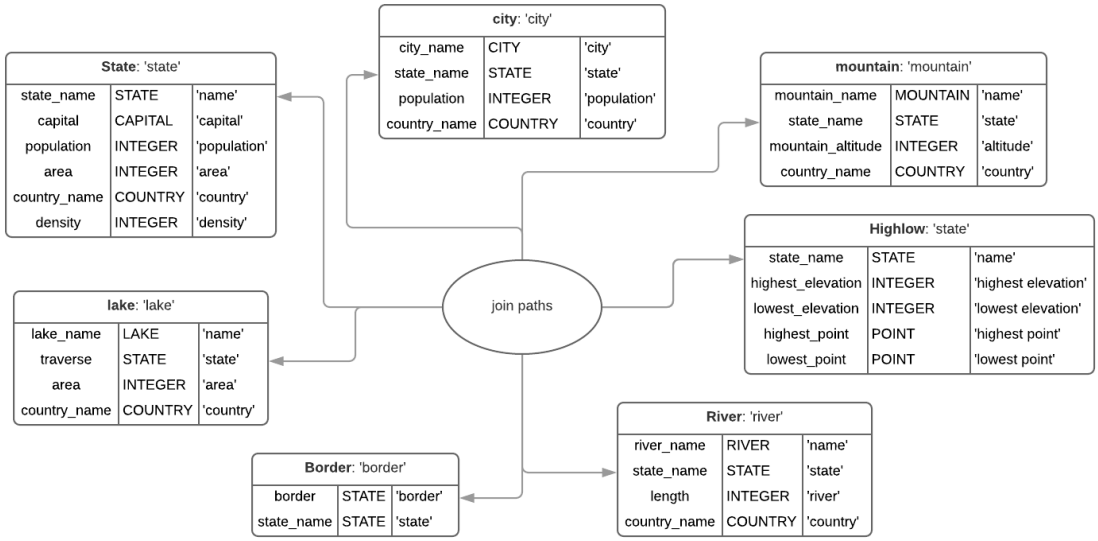


Figure 2.4: Schema of Database from GeoQuery Benchmark

- 11 two-table join requests of the form `SELECT ...FROM {TABLE1},{TABLE2} WHERE ...`
- 29 queries containing the SQL 'GROUP BY' operator.

[7] further provides statistics about the GeoQuery examples. Upon abstracting away database literals from the SQL queries (abstraction level 1 from Figure 1.2), they find the dataset contains 246 unique query patterns, implying an average of 3.6 different NL questions map to the same level-1 abstracted query pattern. They also find 98 unique level-2 SQL patterns; while this implies that on average 8.9 different NL question map to a single pattern, they find that 327 of the 880 NL queries map to a single pattern; that a substantial fraction of the set maps to a single pattern is worrying, as it suggests susceptibility to artificial bias in translation. With respect to SQL complexity, GeoQuery examples contain an average nesting depth of 2.03, though the average number of unique table references is only 1.1; this implies that in most nested queries, the inner and outer queries are over the same table.

We describe these statistics as part of the argument that the GeoQuery train/test split (the training and dev set of which is not used by our system) poses a rather simple challenge to semantic parsers and reflects mcuh of the issues discussed in Section 1.1. Not only is 880 total examples not sufficient to train a zero-shot language understanding model, but the benchmark in general provides many opportunities for overfitting to artifacts. Because the same abstracted query and question patterns appear often in each of the train, dev and test sets, the task is likely to become one of pattern classification rather than true semantic parsing.

Patients Benchmark: The schema of our new benchmark models a medical database which contains only one table comprised of hospital patients with attributes such as name, age, and disease.

We refer this dataset as the Patients benchmark. In total, the benchmark consists of 399 pairs of NL-SQL queries. In an effort to test the linguistic robustness of the given translation model, queries are grouped into one of the following categories depending on the linguistic variation that is used in the NL query: canonical, syntactic paraphrases, morphological paraphrases, semantic paraphrases, and lexical paraphrases as well as a set of queries with missing information. These categories are formulated along the guidelines of paraphrase typologies discussed in [40] and [5].

While the NL queries in the canonical category represent a direct translation of their SQL counterpart, the other categories are more challenging: syntactic paraphrases emphasize structural variances, lexical paraphrases pose challenges such as synonymous words and phrases, semantic paraphrases use changes in lexicalization patterns that maintain the same semantic meaning, morphological paraphrases add affixes, apply stemming, etc., and the NL queries make implicit references to concepts.

We show an example query for each of these paraphrase categories:

- **Canonical**: `What is the average length of stay of patients where age is 80?`
- **Syntactic:** `Where age is 80, what is the average length of stay of patients?`
- **Morphological:** `What is the averaged length of stay of patients where age equaled 80?`
- **Lexical:** `What is the mean length of stay of patients where age is 80 years?`
- **Semantic:** `On average, how long do patients older than 80 stay?`
- **Missing Information:** `What is the average stay of patients who are 80?`

2.2.3 Other Systems Evaluated

We analyze the performance of our approach compared to two other baselines: the rule-based NaLIR [24], and a recent approach [20] that also uses a seq2seq, but in the supervised setting that relies almost fully on manually-curated data.

Supervised seq2seq: As a first baseline we compare against the neural semantic parser introduced by [20], which also uses a deep neural network for translating NL to SQL. However, [20] inherently depends on manually annotated training data on each schema in addition to on-the-fly user feedback. In order to bootstrap the training data creation, [20] provide a limited set of templates to generate some initial NL-SQL pairs. However, their results show that these templatized examples do not contribute much to the overall accuracy of the model. We ran their model on the GeoQuery benchmark only, as the Patients set does not provide a train/test split.

	Patients	**GeoQuery**
NaLIR [24] **(w/o feedback)**	15.60%	7.14%
NaLIR (w feedback)	21.42%	N/A
Supervised seq2seq [20]	N/A	**83.9**%
Our Approach (w/o augmentation)	74.80%	38.60%
Our Approach (full pipeline)	**75.93**%	55.40%

Table 2.2: Accuracy Comparison Between Our Approach and Other Baselines on the Two Benchmark Datasets

NaLIR: Upon its introduction in 2014 and until neural models were first applied to the problem, NaLIR was considered the state-of-the-ART NLIDB. At the core of its database-agnostic framework is a rule-based system which adjusts and refines a dependency parse tree into a query-tree, based on the tree structure and the mapping between its nodes and the RDBMS elements. NaLIR relies heavily on user feedback to refine word to SQL element/function mappings that determine the final SQL string output. For comparison between our system and the other baseline, we run the NaLIR implementation in the non-interactive as well as the interactive setting, where we provide perfect feedback from users (i.e. users always make the correct choices if NaLIR asks for feedback). As seen in our experiments, with interactive feedback we see only a slight increase in accuracy; the system often failed in the initial translation step before giving the user the chance to provide feedback. For example, evaluating NaLIR on the Patients benchmark with user feedback only caused $1 - 2$ more queries to be successfully translated in each paraphrasing category.

Seq2seq Baseline: Finally, in order to show the effectiveness of our augmentation step during data generation, we evaluate each benchmark on two variants of our system: one using a model trained on just the instantiated templates and without the automatic augmentations (this model is referred to as *w/o augmentation*), and one using a model trained on the fully augmented training data (referred to as *full pipeline*).

2.2.4 Results

We evaluated the performance of all NLIDB systems in terms of their accuracy, defined as the number of natural language queries translated correctly over the total number of queries in the test set. Correctness is determined by whether the yielded records from a query's execution in the RDBMS contain the information that is requested by the query intent. The correctness criteria is relaxed by also considering execution results that consist of supersets of the requested columns to be correct. We argue that in practice, users are still able to retrieve the desired information by examining all columns of returned rows. Table 2.2 summarizes the accuracy measures of all NLIDB systems on the two benchmark datasets using this correctness criterion.

It is immediately evident that our approach outperforms the other system that requires no manual

	Naive	Syntactic	Lexical	Morph.	Semantic	Missing	Mixed
NaLIR (w/o feedback)	19.29%	28.07%	14.03%	17.54%	7.01%	5.77%	17.54%
NaLIR (w feedback)	21.05%	38.59%	14.03%	19.29%	7.01%	5.77%	22.80%
Our Approach (full pipeline)	**96.49**%	**94.7**%	**75.43**%	**85.96**%	**57.89**%	**36.84**%	**84.20**%

Table 2.3: Accuracy Breakdown by Paraphrase Category for the Patients Benchmark

effort to support a new database–NaLIR's rule-based approach fails in many cases to produce a direct translation (both with and without user interaction). We only evaluated NaLIR with user involvement on the Patients benchmark due to the high manual evaluation effort of running more than 300 queries and providing feedback for those queries. Furthermore, user feedback did not help to significantly increase the accuracy of NaLIR on Patients.

Meanwhile, our approach is unsurprisingly outperformed on the GeoQuery benchmark by the supervised model. We examine this discrepancy in the discussion.

Results broken down on each paraphrase category of the Patients benchmark are shown in Table 2.3. Regardless of feedback, the rule-based parser generally fails to handle even the relatively simple queries in the benchmark.

2.3 Discussion

It is evident from our results that we can successfully bootstrap a neural text-to-SQL translation model given just an input schema in a new domain, thus doing away with the need for manually-curated examples to achieve at least a baseline performance. We expected that the supervised seq2seq model would outperform our model because it has access to the specific natural language and SQL patterns that appear during evaluation–i.e., the patterns in its train set align nearly perfectly with the patterns in the test, while the patterns in our synthetic data do not align as well. This is mainly the product of our templates not producing queries that are sufficiently domain-specific. The ways in which our approach failed to correctly translate queries in the GeoQuery dataset help to describe this misalignment.

Unseen SQL Pattern: If a SQL pattern appeared in evaluation that did not appear in our synthetic data, our vanilla seq2seq had little to no hope of possibly outputting it. Such queries in the GeoQuery set usually contained one or more level of nests for which we did not create templates. For example, the question

```
what is the length of the river that flows through the most states?
```

corresponds to the query

```
SELECT DISTINCT river.length FROM river WHERE river.river_name = ( SELECT river_name
FROM (SELECT river.river_name, COUNT(1) AS cnt FROM river GROUP BY river.river_name )
AS tmp1 WHERE tmp1.cnt = (SELECT MAX (cnt) FROM (SELECT river.river_name, count(1) AS
cnt FROM river GROUP BY river.river_name ) AS tmp2 ) ).
```

This SQL statement contains aliases (which are currently completely absent from our training data) and multiple nests and is thus very difficult to templatize in a domain-general way with corresponding natural language. That is not to say that a template is not impossible to add, particularly if this type of query is of specific interest to a SQL-aware user.

Domain-Specific Language: If a question contains language that requires ontological inference, it is difficult for our model to make such a decision. For example, the question `How many people live in Austin?` requires the inference that 'people live' corresponds to the city population column rather than a counting operation. The true SQL is

```
select city.population from city where city.city_name='austin';
```

while our model predicts

```
SELECT count(1) FROM city WHERE city.city_name = 'austin'
```

This type of natural language question is not easily covered by templatized training pairs– 'how many people' corresponds to `city.population` because the latter is a numerical column referring to a 'city' that can contain 'people.' Such ontological relationships are very specific and can depend upon database schemata as well as contextual semantics. On the other hand, it is very easy for the supervised seq2seq model to translate this query because the phrase 'how many people' appears 36 different times in the benchmark, and in each instance maps to the population column.

If a test example's SQL pattern was included in our training set but was still translated incorrectly by our model, in a majority of cases it fell under this category. Similar mistranslations occured with the question `how large is Alaska?`, because the model can't connect 'large'-ness with `state.area` versus other numerical columns such as `state.population` or `state.density` without contextual information, and with the question `San Antonio is in what?`, where the model cannot select from all possible entity relationships in the schema the true implied relationship between a `city` entity and its linked `state` entity.

We now revisit the issues discussed in the motivation: we aim to bootstrap generalization into unseen domains without relying on the manually-curated training data bottleneck. We aim to keep models from learning artificial biases in limited data so as to construct 'proper,' cognitively- or linguistically-inspired, and extensible representations of both natural language and SQL. Tied into such extensibility is a notion of compositionality so as to gracefully handle unseen question and query patterns.

Has synthetic training set generation addressed these issues? We believe that it addresses them partially. The domain-agnostic nature of the generator inherently boosts generalization by providing any training items whatsoever when they are otherwise unavailable. Moreover, constructing a training set of size multiple magnitudes larger than the manually-curated sets that is specifically geared towards showing many different ways to ask for the same query patterns ideally encourages a model to not rely on brittle, sequence-level biases.

However, we must consider the scope of information learned by the model. Because the model is

zero-shot– i.e. randomly initialized at the start of training, just like the supervised seq2seq–, the only signals about language and semantic content available to be learned are contained within either the patterns of the templates and lexicon of the generator, or in the noisy word relationships of the PPDB paraphrasing augmentation. In a sense, there is no 'generalization' into the unseen text-to-SQL task domain because the model has no *seen* text-to-SQL task domains from which to draw experience. The generalization of our pipeline lies within the generation step, not the learning step. The model is still isolated to a single training set in a single semantic domain, just as it was in the supervised case. We have therefore not addressed the learning of language itself; we have dealt with the flaws of isolated neural network training by softening their influence–avoiding overfitting to small data and eliminated the overhead of training data curation– rather than reforming the learning process to encourage generalizability. The ways in which our model continues to fail support this conclusion.

In Chapter 3, we deal with a model that has access to many 'seen' domains from which to draw inference, not just a single one. Representations learned by the model are theoretically more general and not only applicable in isolated cases. We explore how we can change the role of our synthetic data; instead of it serving as the entire training set from which the model can learn, it can instead take on the role of truly bootstrapping a more complete learning process.

Chapter 3

Bootstrapping Cross-Domain Learning

In this chapter, we explore how pretraining on synthetic data generated over multiple available databases boosts the performance of SyntaxSQLNet [44], a cross-domain model designed to translate queries over unseen database schemata. We first briefly discuss the model used in their approach. We then proceed with experiments first comparing differing ways to apply synthetic data to the SyntaxSQLNet training pipeline, and then analyzing the best considered approach for its impact on learning and generalization.

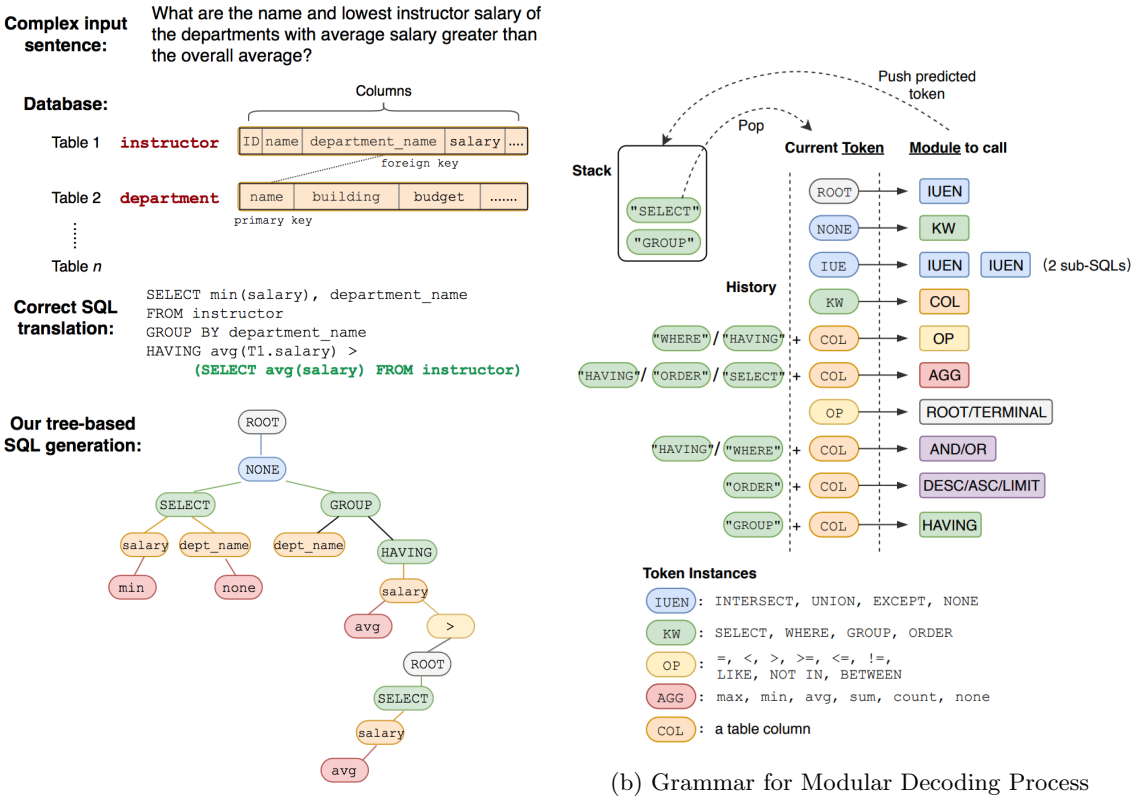
3.1 SyntaxSQLNet Overview

The most notable contributions of the SyntaxSQLNet model are:

1. That it takes as input the schema of the database along with the NL query to be parsed. This allows it to use training data from multiple database domains rather than being consigned to a single one, as was the case with the models of Chapter 2. Relatedly, it also allows the model to generalize to unseen databases after training (which is the scenario on which its authors evaluate it)

2. Its use of a SQL-specific version of syntax tree networks ([43, 34]). This architecture constrains the decoding process to the grammar of SQL and isolates individual grammar-driven decision points in separate modules. This allows the model to handle much more complex queries than vanilla seq2seq models. It also explicitly injects the compositional structure of SQL directly into the model.

SyntaxSQLNet encodes the query and schema using GloVe [31]. It then feeds these encoded states, H_Q and H_{COL} to the syntax-aware modular decoder. The decoder leverages the grammar of SQL to iteratively decode the outputted SQL using different, individually trained decision networks



(a) Example Database and Question Input and Decoding Process

(b) Grammar for Modular Decoding Process

Figure 3.1: Overview of SyntaxSQL Model

for each type of SQL token to be decoded– e.g. what/how many columns, what tables to select from, etc. The decoder decides which modules to use at each step by storing and retrieving previous outputs in a stack. Each module takes in the query, schema, and decoding history and acts as a classifier, determining which of a finite set of SQL or schema tokens to output. The basic pipeline is included in Figure 3.1, borrowed from [44]. Note that each color node corresponds to a different, individually trained network with its own query, schema, and decoding history encoders. Models are trained for up to 600 epochs with early stopping for development accuracy convergence. We refer to the original work for more details.

3.2 Spider Benchmark Overview

The authors of SyntaxSQLNet also introduce a new benchmark, Spider [45], on which their model sets state-of-the art performance. As opposed to previous benchmarks, where queries over a single databases are split into train/dev/test sets, this dataset entails a *database* split, in which the databases themselves are split into train/dev/test. Queries over a particular database only appear in

one set of the split. 20 databases are split into a 146/20/40 train/dev/test, with the test databases and their queries not available publicly. Models are thus evaluated on their ability to generalize across domains, leveraging knowledge from the seen databases in order to translate queries over totally unseen databases without having before seen their schemata nor any NL/SQL queries of them.

In Spider, accuracy is measured by exact match, with partial accuracies available for granular analysis. Test queries are categorized by their degree of translation difficulty, based on the number of SQL components each contains. That is, the more SQL keywords, nested subqueries, column selections, aggregation etc, the higher the difficulty. Because the test set is not available publicly, we treat the development set as the test set from which to draw accuracy metrics.

3.3 Evaluating Strategies for Leveraging Synthetic Data

We first present an initial experiment comparing ways to add our generated data from Chapter 2 to the training pipeline of this model. We use the training set generator to synthesize queries over each database available in the Spider benchmark. In order to create realistic training duration, we apply aggressive recursion filtering to generate only 4-12 thousand examples per database.

3.3.1 Models

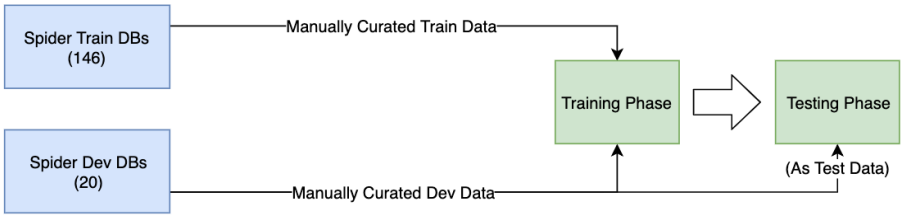
We choose one of two paradigms to apply the data:

PRETRAIN **on Synthetic Data** We first pretrain a randomly-initialized SyntaxSQLNet instance on the fake data (using the same training parameters as the regular pipeline) until development convergence, and then train it normally on the Spider training set (akin to a regular instance of the model pipeline).

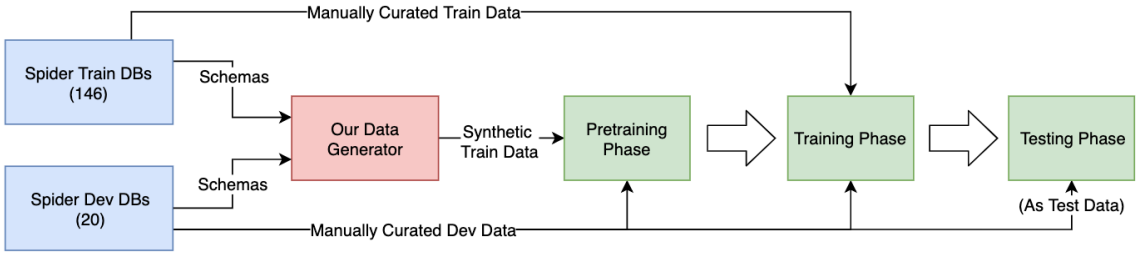
COTRAIN **on both Synthetic and Spider Data** We train a randomly-initialized SyntaxSQL-Net instance on both the fake data and Spider training data simultaneously. In order to keep the synthetic data, whose size dwarfs that of the Spider data, from dominating the training process, we randomly sample a equivalent number of synthetic training pairs as there are Spider pairs.

As a baseline, we compare against the a model–referred to as BASE–trained regularly with only the Spider train data. As a second baseline, we also evaluate the pretrained SyntaxSQLNet model before it is trained regularly on the Spider train set to produce the PRETRAIN model– we refer to this version as SYNTHETIC-ONLY. An overview of each of these pipelines is presented in Figure 3.2.

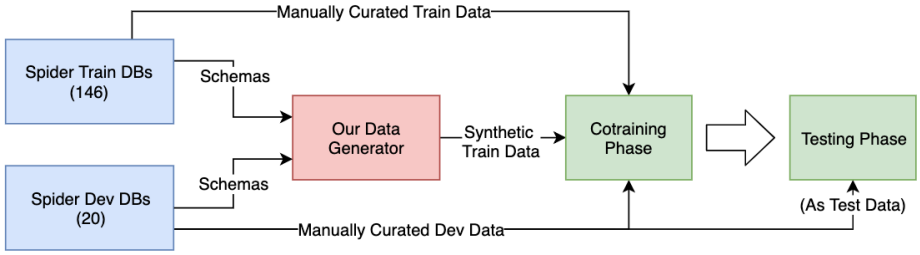
It is important to note that this initial experiment did not directly align with the original scenario in which Spider intends to evaluate a model. While Spider evaluates the capacity to translate queries over totally unseen schemata, this experiment evaluates a model in the same scenario as the experiments from Chapter 2, in which the target schema is available at the time of data generation



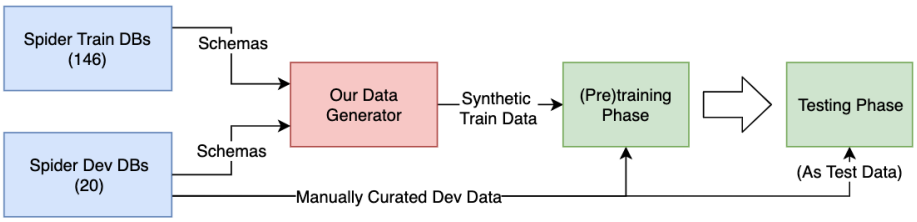
(a) BASE Pipeline



(b) PRETRAIN Pipeline



(c) COTRAIN Pipeline



(d) SYNTHETIC-ONLY PIPELINE

Figure 3.2: Strategies Evaluated in Experiment 3.3

	easy	med	hard	extra	**all**
count	250	440	174	170	**1034**
BASE	.360	.157	.195	.041	**.193**
SYNTHETIC	.137	.075	0	0	**.068**
COTRAIN	.388	.252	.207	.047	**.244**
PRETRAIN	.460	.273	.264	.065	**.282**

Table 3.1: Spider Development Accuracy Comparison for Varying Training Paradigms in 'Off-the-Shelf' Scenario

rather than being totally unseen. Accordingly, we generate synthetic data over all 166 train and target (dev) databases, but only use Spider train data over the 146 train databases.

We refer to this manufactured scenario as the 'off-the-shelf' case, as it mimics the off-the-shelf systems evaluated in Chapter 2. Thus, conclusions drawn from this experiment's results must be taken with a grain of salt, though, as will become evident, the pretraining paradigm seems to perform significantly better.

3.3.2 Results & Discussion

Exact match query accuracy on the Spider development set is displayed in Table 3.1. While both paradigms for leveraging synthetic data provide a performance boost, the PRETRAIN paradigm outperforms all other versions across all query difficulties. Notably, it provides a 7% boost on hard-level queries, of which the model trained on only synthetic data could translate none. This implies that either cotraining on synthetic data makes it harder to learn more complicated SQL queries, or that pretraining allows a model to learn more generalizable representations of harder queries. The former conclusion is somewhat explained away by the marginal boost COTRAIN provides over BASE on hard queries.

We thus hypothesize that the model uses the synthetic data over all domains as a scaffold towards being able to generalize more complicated query patterns seen in the Spider training set. The SQL queries contained within the synthetic data are relatively simple– as shown by the 0% translation accuracy of the SYNTHETIC model for all hard and extra-hard queries, plus its nearly-zero performance on medium queries. Thus it is peculiar that the model improves substantially over each of these harder categories–PRETRAIN outperforms BASE by 12%, 7% and 2.5%, respectively in these categories. Since the model is not seeing ANY difficult queries over development databases, it must be the case that it learns how to identify and translate them by generalizing from the difficult queries in the training set.

The fact that the PRETRAIN paradigm outperforms the COTRAIN most substantially for hard queries might also point to this hypothesis, since it is less useful to be showing fake, simple queries while the model is simultaneously trying to comprehend more difficult, natural queries. We catered

the generator towards performing lingistically robustly only to the simple set of queries that it generates–it seems that it is less useful to be enforcing this goal of robustness during the same period where the goal is to understand more complicated queries from far less data. As a result, for our following experiments, we will use models trained with the PRETRAIN paradigm.

3.4 Augmenting the Full SyntaxSQL Pipeline

Here we discuss our application of the PRETRAIN to the SyntaxSQLNet setup from [44] that achieves state-of-the-art performance on the Spider benchmark[1]. Their best setup involves training on not only the Spider train queries, but also on queries synthesized from their own templatized training data generator. To construct this generator, they gathered the 50 most common abstraction level-2[2] patterns and coverted them into templates. They then generate training examples over the databases from the WikiSQL dataset, a benchmark containing a very large number of single-table 'databases' scraped from Wikipedia. This augmentation serves to show the model a higher order of magnitude amount of possible schemas to encode with its schema encoder, as well as increasing the number of domains from which it learns to translate queries during training.

3.4.1 Models

We apply our own synthetic data generation to the WikiSQL tables as well, producing hundreds of thousands of queries. We then pretrain a SyntaxSQLNet instance on our synthetic data comprising queries over both the WikiSQL and Spider train databases before the regular training phase, in which it mimics [44]'s state-of-the-art setup by training on the manually-curated Spider train queries combined with their synthetic data over the WikiSQL databases.

We compare the resulting model against the original state-of-the-art model's pipeline. These pipelines are summarized in Figure 3.3.

3.4.2 Results and Discussion

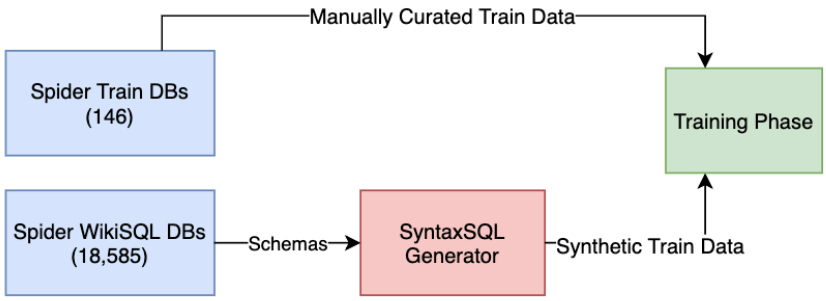
The resulting change in exact query match accuracy is displayed in Table 3.2. Notably, our augmented pipeline doubles the model's accuracy on extra-hard queries.

We have thus demonstrated that pretraining on synthetic data from available domains boosts a model's ability to generalize into unseen domains, even if the target schema is unavailable until translation time.

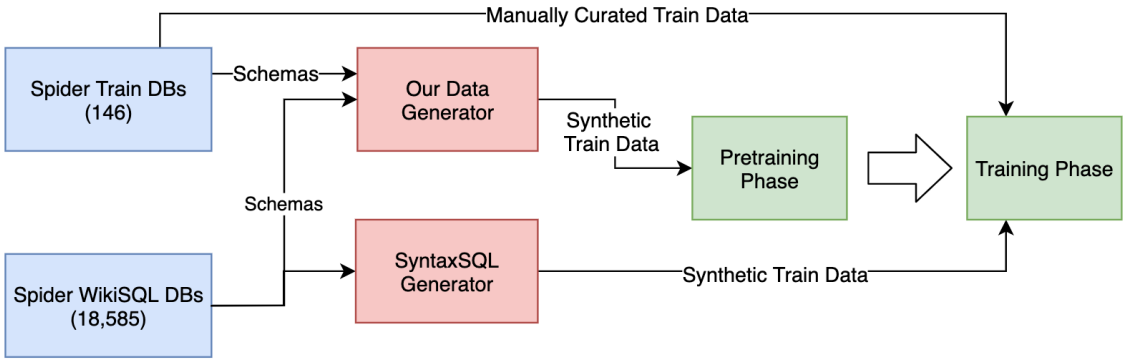
In Table 3.3, we follow the pattern of analysis of [7] to explore the types of errors made by the model with and without the addition of our synthetic data. These error categories are divided by whether the abstracted SQL pattern of an evaluated query has been seen during training. We

[1] Here and in proceeding sections we refer to the original Spider scenario, no longer the 'off-the-shelf' scenario we manufactured for Section 3.3. This means that the Spider development databases are not shown to the model in any training phase.

[2] Levels defined in Table 1.2.



(a) Original Training Pipeline from [44]



(b) Pipeline Augmented with Our Synthetic Data Generator

Figure 3.3: Augmentation of Training Pipeline in Experiment 3.4

	easy	med	hard	extra	**all**
count	250	440	174	170	**1034**
Original SyntaxSQLNet Pipeline	.445	.227	.231	.051	**.248**
Our Augmented Pipeline	.472	.300	.252	.107	**.299**

Table 3.2: Spider Development Accuracy Comparison for Augmented Training Pipeline

	Pattern Category	Original	Augmented	
Correct	gold seen	196	249	
	gold unseen	60	60	
Incorrect	gold seen, pred seen	**402**	**382**	(+10 synthetic)
	gold seen, pred unseen	**224**	**187**	(-4 synthetic)
	gold unseen, pred seen	107	106	
	gold unseen, pred unseen	45	40	(-6 synthetic)

Table 3.3: Frequencies of Error Types by SyntaxSQLNet on Spider Development With/Without Synthetic Augmentation

first anonymized all queries in the training and test sets to their most general level of abstraction; example patterns are displayed below:

```
SELECT COL FROM TABLE WHERE COL <= terminal AND COL = terminal ;

SELECT COUNT( * ) , COL FROM TABLE GROUP BY COL ;

SELECT COL FROM TABLE WHERE COL = ( SELECT MAX( COL ) FROM TABLE ) ;
```

We then categorized each pair by whether the gold pattern and predicted pattern were contained within the training set.

The results in Table 3.3 reveal that the boost in performance provided by the augmented SyntaxSQLNet model are the result of higher performance on pairs where the gold query pattern emphhas been seen during training. On the other hand, no boost in performance is show on pairs where the query pattern *has not* been seen. This suggests that the synthetic data boosts the robustness of the model towards seen query patterns, but does not improve the ability of the model to grasp the compositional nature of SQL so as to recombine seen queries into novel patterns.

3.5 Evaluating Impact of Synthetic Data Magnitude

We briefly explore the impact of the amount of synthetic data available for pretraining on the ability of syntaxSQLNet to learn patterns within the synthetic data and extrapolate them to Spider test

queries before the Spider training phase of our augmented training paradigm.

This experiments serves to answer multiple questions. The first is whether one needs to generate less synthetic data in order to achieve the same rate of performance in the model. If an order of magnitude less training data yields the same performance, then time and storage is saved during training.

The second question is to what extent the synthetic data aligns with those in the more complicated and diverse Spider test queries, or alternatively only facilitates successful performance on Spider as a pretraining step in that it boosts the effectiveness of training on the Spider train queries. In other words, we seek to determine to what extend the synthetic data actually serves as typical 'training data' for the Spider task, or whether it instead prepares the model to more effectively learn from the manually-curated training data that more closely reflects the complexity and nature of task.

In order to answer these questions, we measure the accuracy of the model on the Spider development set, as well as a development set comprising synthetic queries over the Spider development databases. Accuracy on the synthetic development set reveals the extent to which the patterns in the fake data are learned during pretraining. Accuracy on the Spider development set reveals the extent to which the model can translate the test queries before it is exposed to any Spider data, thus revealing whether the synthetic data contains any true 'training' signal for the ultimate task.

Because each module of the SyntaxSQLNet tree-based decoding model is trained individually, distinct training sets are constructed from the NL/SQL training set for each module. For each step in which the module would be called in a correct decoding of each development NL/SQL pair, the NL question and ground truth decoding history up to that step are taken as the input of a training example for the module.

Because of this, it is difficult to evaluate full translation accuracy on a development set for each epoch. Instead, we measure the accuracy of the COL decision module, which is the model responsible for deciding the number and names of columns to be decoded whenever the stack-based decoding process requires such a decision. We refer the reader to section 4 the original work [44] for a detailed overview of this decision mechanism. We choose the COL module in particular because it is called most often of all modules, and moreover it has the highest misprediction rate. Full mis-translations of the NL/SQL examples are most likely to be caused by a mistake of this module.

3.5.1 Models

We take the COL training examples derived from the synthetic data generated over the Spider train databases in Experiment 3.4, and take the COL development examples accordingly from the data generated over the Spider development databases. We randomly subsample from the development set to obtain train and development sets of size 1,445,000 and 2500, respectively.

We then train a model on the training set subsampled by a factor of each of 10, 100, and 1000 (i.e. train sizes of 145 thousand, 14.5 thousand and 1.45 thousand, respectively).

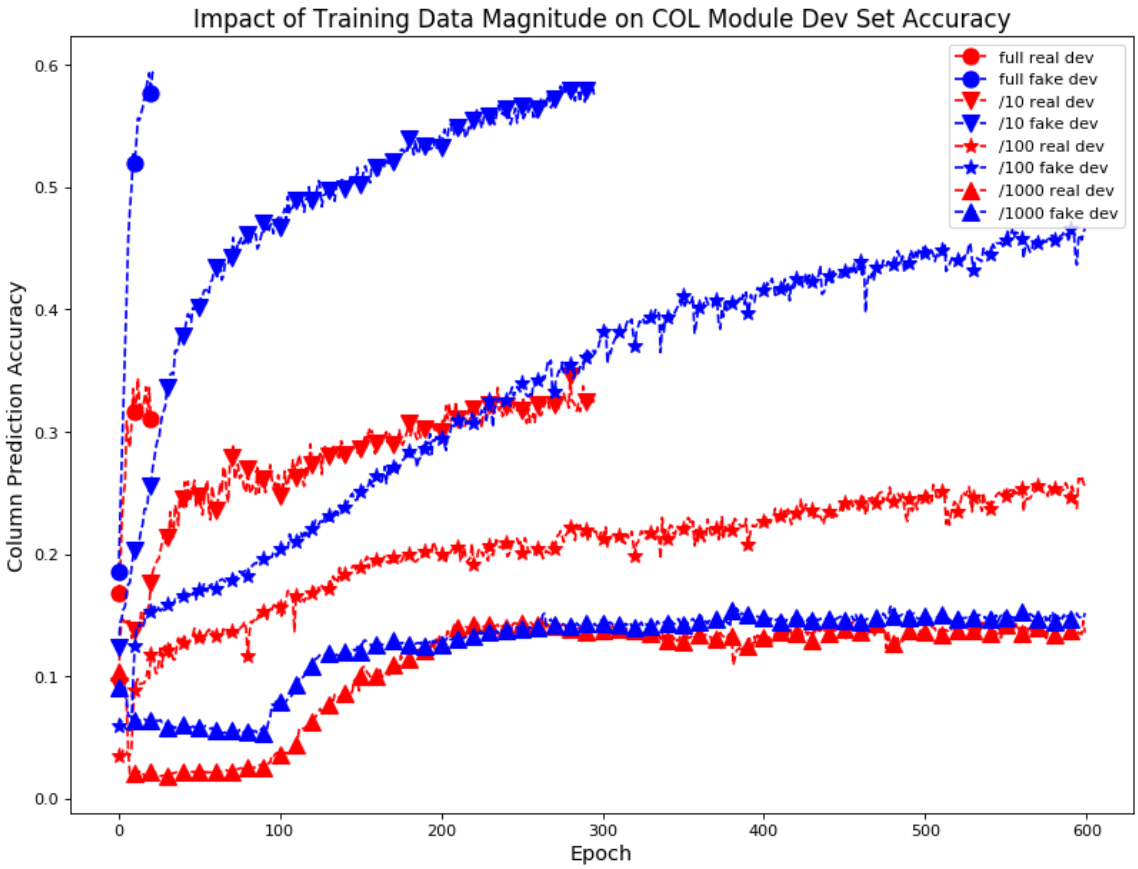


Figure 3.4: Epoch Accuracy of COL Module Trained on Synthetic Data with Varying Magnitudes

3.5.2 Results & Discussion

Preliminary results of evaluating these models on synthetic and Spider dev epoch accuracy are shown in Figure 3.4[3]. To calculate epoch accuracy, an example is marked correct if the model predicts both the ground truth number and names of columns to be decoded.

We find that in the full, 600-epoch training duration, the models trained with .1% and 1% of the synthetic data fail to sufficiently learn the patterns in both the fake, generated data and the real, Spider data. The model trained with 10% of the synthetic data reaches the same level of accuracy as the model trained on all of it, though it takes it almost exactly 10 times longer to do so (300 epochs versus 30). This implies that these two models achieve the same development performance given the same cumulative number of examples trained upon, and thus only 10% of the data is required to approach the likely the limit of accuracy derivable from the synthetic pretraining.

With respect to the above question of whether the fake data facilitates performance on the real test data from Spider, these results imply that it does not, or at least is far less effective than the real Spider training data. It takes 30 epochs training on 1.45 million synthetic examples (i.e. 43.5

[3]Longer train sessions were cut short due to time constraints.

million cumulative examples) to achieve 34% accuracy on the real data, while it takes 86 epochs training on 20 thousand real examples (i.e. 172 thousand cumulative examples) to achieve the same accuracy without any injection of synthetic data during pre- or co-training.

These preliminary results do not yet lead us to conclude that pretraining data magnitude has an impact of the learning rate of the model on the Spider train data. We plan to evaluate the development accuracy of the above models during the regular training phase in order to address this question.

3.6 Evaluating Impact of Source Domain Magnitude

We now address the hypothesis that access to more semantic domains allows a model to build more general and extensible representations of words and query patterns. As discussed in Chapter 1, previous work in Neural Semantic Parsing [19] found that sharing learned parameters from 'seen' semantic domains with a model parsing in an unseen domain is beneficial to model performance. Here we investigate whether the same relationship is found with the synthetic pretraining data. We hypothesize that access to more schemas during pretraining encourages the model to generalize more capably to unseen ones during testing.

3.6.1 Models

To test this hypothesis, we pretrain models on different sets of available databases.

The first model, SpiderOnly, is pretrained on data generated over the 144 Spider train databases. The second, WikiSQLOnly, is pretrained on data generated over the 18,585 WikiSQL databases. Finally, Both is pretrained on data generated over the combined set.

We pretrain each model until synthetic development set convergence (using the same synthetic development set over Spider development databases as in Experiment 3.4).

As with Experiment 3.5, for ease of evaluation we evaluate the epoch accuracy of the COL module only, rather than the entire model. As two baselines, we compare the above models with NoPretrain, a COL module trained without any pretraining on synthetic data, and FullPipeline, the col module in our best model from Experiment 3.4 that is pretrained on both WikiSQL and Spider databases, and then is trained with the extra data generated from WikiSQL databases by the generator of the original SyntaxSQLNet work.

3.6.2 Results & Discussion

Resulting epoch accuracies are displayed in Figure 3.5. As evident by the three models of interest performing essentially indistinguishably, the number of source domains from which the pretraining data is generated has little to no impact on overall performance on the development set.

Notably, this is not the case during the regular training phase; the addition of data from WikiSQL databases via the original work's generator provides a noticable performance boost. This implies

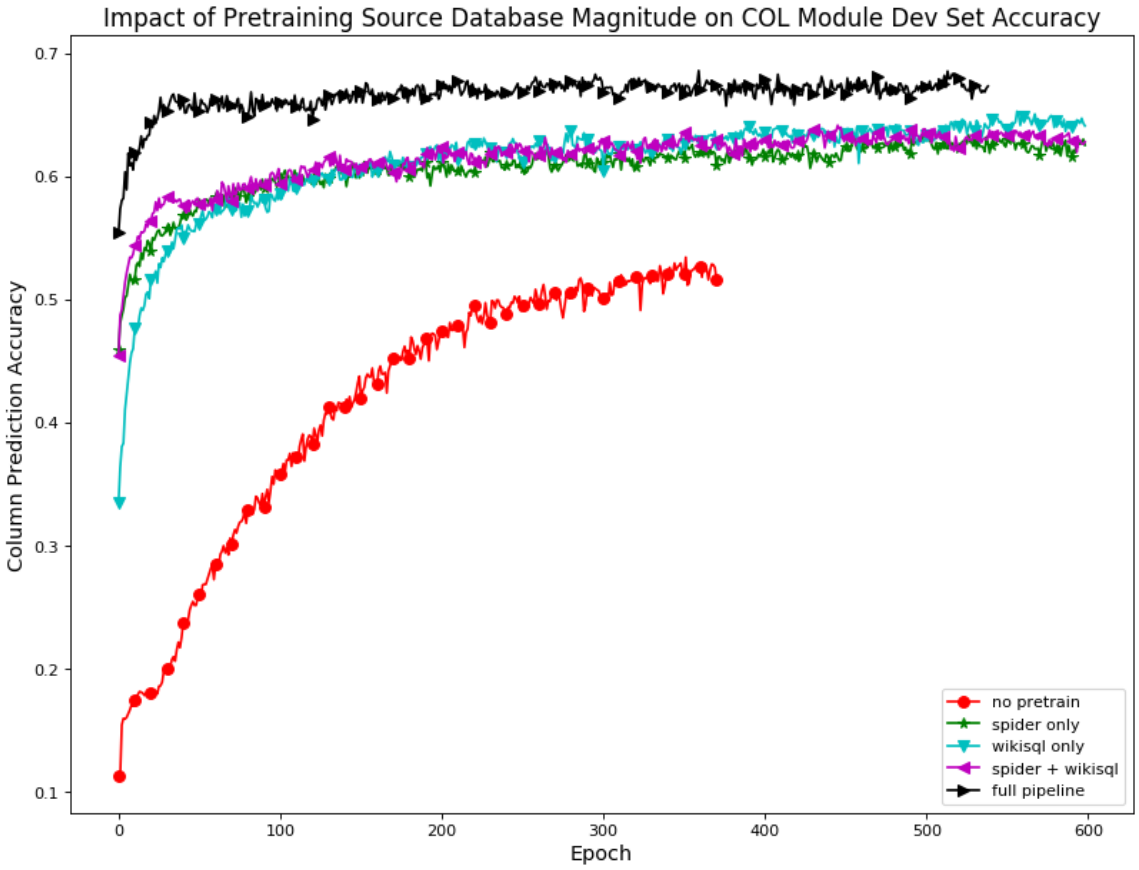


Figure 3.5: Epoch Accuracy of COL Module Trained on Synthetic Data from Varying Source Domains

that domain availability has much more importance during the regular phase of training, and does not serve much of a purpose during the pretraining phase.

This does not support our hypothesis that it would, in fact, benefit the overall performance to apply our generator to more databases for the pretraining step. However, it may support the hypothesis discussed in Section 3.5 that the pretraining step serves a purpose very distinct from the regular training step. This purpose is to *prepare* the model for cross-domain learning rather than to serve as the source of the learning itself.

We plan to test this conclusion by further varying the number of source databases providing queries during the regular training step rather than pretraining. It is important to note here that the Spider train NL/SQL pairs seen during the regular training step are manually curated and therefore more diverse on the NL side than any of the synthetic sets. This might lead to a larger drop in performance if we remove Spider train database availability, if only because the available queries provide the most natural and diverse signals to be learned. This issue again reflects the fundamental difference between the richness entailed by manual curation and the relative simplicity of templatized approaches such as ours. There may be no way to fully replace the role of fully naturally-occurring training examples in bootstrapping these data-driven language understanding models.

3.7 Evaluating Compositional SQL Generalization

Finally, we explore whether pretraining on our synthetic data boosts the ability of SyntaxSQLNet to grasp the compositional structure of SQL and generalize to novel SQL patterns. We return to the work of [7], whose analysis of text-to-SQL benchmarks concluded that training on these small datasets with relatively few, particular SQL patterns that appear in both training and testing sets–a paradigm they term the 'question-based' split– results in the task becoming classification over seen patterns rather than true semantic parsing. Successful performance on these splits, they argue, does not equate to sufficiently generalizable understandings of NL nor SQL. Accordingly, they introduce new 'query-based' splits of the benchmarks that *require* compositional generalization of SQL: queries seen during evaluation are specifically unseen during training. Seq2seq models have thusfar performed very poorly on these splits, with drops in performance of up to 80% between the 'question'- and 'query'-based split of a benchmark.

We seek to test whether the addition of our pretraining step increases the performance of SyntaxSQLNet on a 'query'-based split. We set up a scenario in which the network trains normally with the full pipeline as discussed in Experiment 3.4, but with the training set of either the 'question' or 'query' split of a target domain injected into its regular training step. Thus, the model must generalize to output SQL patterns that are unseen within the target domain, and must rely on either having seen the patterns in other domains, on recombining the patterns that it does see in the target domain, or a combination of both methods.

3.7.1 Advising Benchmark Overview

We use the benchmark introduced by [7] to evaluate our model's compositional generalization. It consists of questions over a database of course information at the University of Michigan. It contains 4570 unique NL queries mapping to 211 unique entity-abstracted (abstraction level 1) SQL queries. The SQL queries contain 174 patterns with literals and entities abstracted (level 2), as compared with the GeoQuery benchmark's total of 98. While it contains fewer nests per query, its patterns are still quite a bit more complex, containing an average of 3.0 unique tables in each query instance (compared to GeoQuery's 1.1). It thus requires substantially more handling of the ontological relationships modeled by the Database schema in order to compute complicated operations involving many joins.

As well as because it is a more generally difficult schema to handle with more complicated queries, we chose this benchmark because of the discrepancy in performance between the split types; no model evaluated by [7] achieves more than a 8% accuracy on the query-based split, while they achieve up to nearly 90% on the question-based split.

Some of the examples in the dataset are not included in the question-based split; we filter them out of the query-based split for fairness of evaluation. Moreover, some of the query patterns in this dataset cause the SQL parser used to train and evaluate SyntaxSQLNet to crash; we filter these examples out as well. We end up with 2555 NL/SQL pairs, divided into a 1326/329/900 query-based train/dev/test split and 1941/176/438 question-based split.

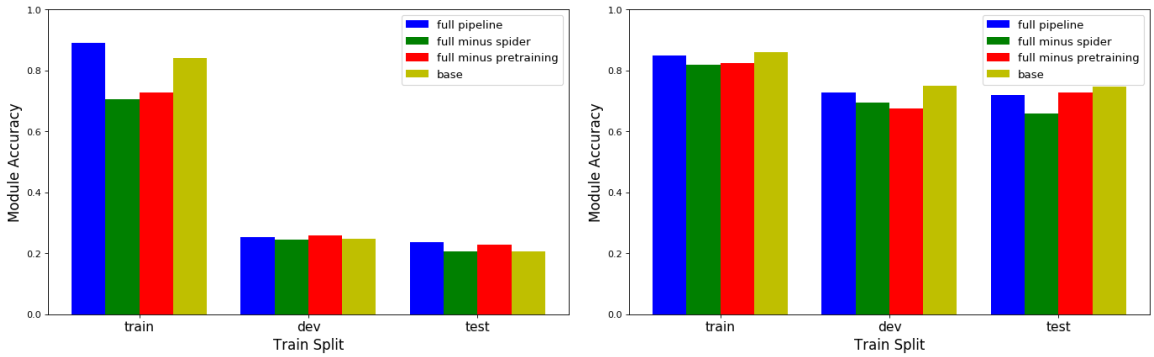
3.7.2 Models

We evaluate 4 different training pipelines, from each of which a first model is trained using the query-based split, and a second is trained using the question-based split.

- First, as a baseline model Base, a randomly-initialized SyntaxSQL model is trained using only the Advising training/dev data.
- Second, to isolate pretraining as a contributing factor, a randomly-initialized model is trained on the Spider training/dev set[4] with the Advising data mixed into it. We refer to this as NoPretrain.
- Third, to isolate the Spider data as a contributing factor, a model is pretrained on our generated data[5] before regular training on only the Advising data (a la Base). We refer to this as NoSpider.
- Finally, a model is pretrained on our generated data and then trained on the combined Spider and Advising data. We refer to this as FullPipeline.

[4]This includes the WikiSQL synthetic queries generated by the SyntaxSQL authors.

[5]This includes data generated over the Spider train as well as WikiSQL databases.



(a) Query-Based Split

(b) Question-Based Split

Figure 3.6: COL Module Prediction Accuracy Across Query and Question Splits on Advising Benchmark

3.7.3 Results & Discussion

Unfortunately, we found that a bug in the SQL parsing script required to evaluate our trained models created an unidentified misalignment between queries and the object representation predicted by the model. As a result, on all experiments, the models achieved 0% translation accuracy on both the query and question splits. Due to time constrains, we could not identify the bug. However, we are still able to examine the module-level accuracies of the models rather than relying on the parsing script to output the full SQL predictions.

Here, we will display only the COL module, given (as discussed above) it is responsible for a majority of translation errors by the model. We present train, development and test accuracy for the COL module in Figure 3.6.

It is notable first that, as expected, there is a sizable drop-off in the query-based split on the development and test sets that is not present in the question-based split. This is clearly a result of the nature of the split; it is far more difficult to traslate to SQL patterns that are unseen. Note that a 20% accuracy rate for a column module that is called multiple times during a decoding process will likely result in incredibly low translation accuracy, even with the aforementioned parsing bug fixed.

Secondly, note that the models are all essentially interchangeable (i.e. within a margin of statistical insignificance) on the question-based split. This implies that on this split, it is not unlikely that training from other domains is not contributing to the successful translation of Advising queries. This is also not incredibly surprising; because the evaluation set highly reflects the language and patterns learned in the training set, there is very little need to leverage cross-domain knowledge in order to correctly perform the 'pattern classification' task that the question-based split entails.

Finally, the models are also essentially interchangeable on the development and test sets of the query-based model. This leads us to conclude that neither pretraining nor cross-domain training boost the compositional generalizability of SyntaxSQLNet. This is not to say that they may not do so for other models. This particular model uses a modular decoder that constrains it to making

isolated classification decisions about next-token decoding. The modules have their own sets of encoding parameters and make decisions independently of each other, apart from the sharing of a common decoding history sequence (which each module still encodes individually). It remains to be seen whether a more connectionist architecture– i.e. one that shares more parameters or does away with the modular decoder, but can still translate queries across domains– might benefit from incorporating synthetic pretraining or cross-domain training. In future experiments, we thus plan to incorporate other flavors of seq2seq models on this task.

Chapter 4

Conclusion

The goal of this thesis has been to explore a novel strategy to address the flaws in the typical training pipeline of data-driven, deep Neural Semantic Parsing models for text-to-SQL translation. We identified how training models on manually-curated queries in an isolated, supervised approach to learning creates brittle, domain-dependent translation models that rely on artificial biases inducted from the limited data and thus cannot in good faith be considered 'language understanding' nor 'semantic parsing' models. We have attempted to craft an approach to semantic parsing that does away with these flaws in manually-curated training sets and attempts to bootstrap a more general model that might be encouraged to perform a more natural process of learning to parse natural language.

4.1 Summary of Contributions

In Chapter 2, we introduced a method for taking target database schema as input and generating a diverse set of millions of synthetic training examples from which to train a text-to-SQL translation model. This generation process can serve at the core of a domain-general, online pipeline for translating queries in a Natural Language Interface for Databases (NLIDB). We discussed the template-based approach to crafting the training set, which is geared toward linguistic robustness over a substantial set of templatized SQL patterns.

We crafted a new benchmark, the Patients dataset, to evaluate a model's linguistic robustness, and showed that our approach using a simple seq2seq architecture yields a model that drastically outperforms other systems on this new benchmark, as well as performing competitively with a supervised seq2seq model on the seminal GeoQuery benchmark.

In Chapter 3, we demonstrated how synthetic data generation can be used not only to construct zero-shot models over a single target schema, but can also serve to boost the performance of cross-domain models that translate queries over many ad-hoc schemata. We discussed different strategies for leveraging synthetic data, and then demonstrated how our best approach, that of pretraining a model on our synthetic data before fine-tuning it on manually-curated examples from available

domains, yields a model that sets state-of-the-art performance on a recent benchmark for cross-domain text-to-SQL translation.

We also explored how both the magnitude of synthetic data available during pretraining and also the number of domains from which the data is generated impact domain generalization. Finally, we tested whether synthetic data pretraining boosts the compositional generalization of the model. While the final experiment yielded negative results, these experiments led us towards the conclusion that synthetic pretraining serves a role in preparing a model to more effectively learn to generalize during subsequent training steps.

4.2 Future Directions

As we have discussed, a key difficulty in leveraging synthetic data is that it fundamentally is not as diverse nor as expressive as its manually-curated counterpart. However, we are exploring various ways to bridge this gap. Possible strategies include neural paraphrase generation using language pivoting ([27]) or backtranslation using SQL-to-NL summarization systems ([22]).

At a higher level of pipeline abstraction, we seek to de-isolate the learning model so that it can more naturally learn representations of language that can generalize more easily into new domains. Such representations can be learned via multi-task learning frameworks. We are exploring ways to incorporate into our text-to-SQL model recent advances in language model pretraining ([12]), an approach which has become the de facto standard for building state-of-the-art natural language understanding models.

Finally, building off of the intuition that synthetic pretraining serves to condition models to 'learn better' from their training phase, we want to explore recent approaches to task meta-learning ([16]), in which models treat entire tasks as training examples for the meta-purpose of generalizing more capably and efficiently to new domains.

Bibliography

[1] Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. Natural language interfaces to databases - an introduction. *Natural Language Engineering*, 1:29–81, 1995.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[3] Islam Beltagy, Katrin Erk, and Raymond J. Mooney. Semantic parsing using distributional semantics and probabilistic logic. In *ACL*, 2014.

[4] Jonathan Berant and Percy Liang. Semantic parsing via paraphrasing. In *ACL*, 2014.

[5] Rahul Bhagat and Eduard H. Hovy. Squibs: What is a paraphrase? In *ACL*, 2013.

[6] Ruichu Cai, Boyan Xu, Zhenjie Zhang, Xiaoyan Yang, Zijian Li, and Zhihao Liang. An encoder-decoder framework translating natural language to database queries. In *IJCAI*, 2018.

[7] Li Zhang Karthik Ramanathan Sesh Sadasivam Rui Zhang Catherine Finegan-Dollak, Jonathan K. Kummerfeld and Dragomir Radev. Improving text-to-sql evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, July 2018.

[8] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014.

[9] Stephen Clark and James R. Curran. Parsing the wsj using ccg and log-linear models. In *ACL*, 2004.

[10] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew McCallum, Tom M. Mitchell, Kamal Nigam, and Seán Slattery. Learning to construct knowledge bases from the world wide web. *Artif. Intell.*, 118(1-2):69–113, 2000.

[11] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W. Bruce Croft. Neural ranking models with weak supervision. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, pages 65–74, 2017.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[13] Li Dong and Mirella Lapata. Language to logical form with neural attention. *CoRR*, abs/1601.01280, 2016.

[14] Xing Fan, Emilio Monti, Lambert Mathias, and Markus Dreyer. Transfer learning for neural semantic parsing. In *Rep4NLP@ACL*, 2017.

[15] Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Xiang Lin, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir R. Radev. Improving text-to-sql evaluation methodology. *CoRR*, abs/1806.09029, 2018.

[16] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017.

[17] Alessandra Giordani and Alessandro Moschitti. Automatic generation and reranking of sql-derived answers to nl questions. In *Proceedings of the Second International Conference on Trustworthy Eternal Systems via Evolving Software, Data and Knowledge*, pages 59–76, 2012.

[18] Alessandra Giordani and Alessandro Moschitti. Translating questions to sql queries with generative parsers discriminatively reranked. In *COLING*, 2012.

[19] Jonathan Herzig and Jonathan Berant. Neural semantic parsing over multiple knowledge-bases. In *ACL*, 2017.

[20] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. Learning a neural semantic parser from user feedback. 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 963–973, 2017.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[22] Andreas Kokkalis, Panagiotis Vagenas, Alexandros Zervakis, Alkis Simitsis, Georgia Koutrika, and Yannis Ioannidis. Logos: a system for translating queries into narratives. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 673–676. ACM, 2012.

[23] Mirella Lapata and Li Dong. Coarse-to-fine decoding for neural semantic parsing. In *ACL*, pages 731–742, 2018.

[24] Fei Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *PVLDB*, 8:73–84, 2014.

[25] Percy Liang, Michael I. Jordan, and Dan Klein. Learning dependency-based compositional semantics. *Computational Linguistics*, 39:389–446, 2011.

[26] Edward Loper and Steven Bird. Nltk: The natural language toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*, 2002.

[27] Jonathan Mallinson, Rico Sennrich, and Mirella Lapata. Paraphrasing revisited with neural machine translation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 881–893, 2017.

[28] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[29] Isil Dillig Navid Yaghmazadeh, Yuepeng Wang and Thomas Dillig. Sqlizer: Query synthesis from natural language. In *International Conference on Object-Oriented Programming, Systems, Languages, and Applications, ACM*, pages 63:1–63:26, October 2017.

[30] Ellie Pavlick and Chris Callison-Burch. Simple PPDB: A paraphrase database for simplification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*, 2016.

[31] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

[32] 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.

[33] Ana-Maria Popescu, Oren Etzioni, and Henry A. Kautz. Towards a theory of natural language interfaces to databases. In *IUI*, 2003.

[34] Maxim Rabinovich, Mitchell Stern, and Dan Klein. Abstract syntax networks for code generation and semantic parsing. In *ACL*, 2017.

[35] Rodolfo A. Pazos Rangel, Joaquín Pérez Ortega, Juan Javier González Barbosa, Alexander F. Gelbukh, Grigori Sidorov, and M. MyriamJ.Rodríguez. A domain independent natural language interface to databases capable of processing complex queries. In *MICAI*, 2005.

[36] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R. Mittal, and Fatma Özcan. Athena: An ontology-driven system for natural language querying over relational data stores. *Proc. VLDB Endow.*, 9(12):1209–1220, August 2016.

[37] Alvin Cheung Jayant Krishnamurthy Srinivasan Iyer, Ioannis Konstas and Luke Zettlemoyer. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973, 2017.

[38] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pages 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

[39] Prasetya Utama, Nathaniel Weir, Fuat Basik, Carsten Binnig, Ugur etintemel, Benjamin Hättasch, Amir Ilkhechi, Shekar Ramaswamy, and Arif Usta. An end-to-end neural natural language interface for databases. *CoRR*, abs/1804.00401, 2018.

[40] Marta Vila, Maria Antònia Martí, and Horacio Rodríguez. Paraphrase concept and typology. A linguistically based and computationally oriented approach. *Procesamiento del Lenguaje Natural*, 46:83–90, 2011.

[41] Oriol Vinyals, Lukasz Kaiser, Terry K Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. Grammar as a foreign language. In *NIPS*, 2015.

[42] Yushi Wang, Jonathan Berant, and Percy Liang. Building a semantic parser overnight. In *ACL*, 2015.

[43] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. *CoRR*, abs/1704.01696, 2017.

[44] Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir R. Radev. Syntaxsqlnet: Syntax tree networks for complex and cross-domaintext-to-sql task. In *EMNLP*, 2018.

[45] 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 *EMNLP*, 2018.

[46] John M. Zelle and Raymond J. Mooney. Learning to parse database queries using inductive logic programming. In *AAAI/IAAI, Vol. 2*, 1996.

[47] Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI*, 2005.

[48] Luke S. Zettlemoyer and Michael Collins. Online learning of relaxed ccg grammars for parsing to logical form. In *EMNLP-CoNLL*, 2007.