Generalization Challenges in Semantic Parsing

Bailin Wang

Doctor of Philosophy

Institute for Language, Cognition and Computation
School of Informatics
The University of Edinburgh

2022

Abstract

Semantic parsing is the task of translating natural language utterances onto machine-interpretable programs, which can be executed against a real-world environment to obtain desired responses (e.g., a SQL query against a relational database). It is an established paradigm for building natural language interfaces. However, most existing semantic parsing systems are built only for the conventional yet limited setting, i.e., in the context of a single fixed database. Moreover, they are typically data-hungry, i.e., they require a large number of examples for training. This thesis focuses on extending the limited setting to a diverse spectrum of settings inspired by real-life scenarios, and addressing the generalization challenges that arise during such extensions.

We consider three aspects of semantic parsing along which the conventional setting deviates based on some real-life scenarios. Firstly, we consider *transferability*, a property indicating whether a semantic parser is applicable on unseen domains (e.g., unseen databases), leading to cross-domain setting. Secondly, we consider *three forms of supervision*. Apart from standard, yet expensive, utterance-program pairs, we investigate settings where cheap supervision in the form of utterance-response pairs is given, or even no supervision is at all. Thirdly, we study *linguistic coverage* i.e., the extent to which a semantic parser can cover the space of utterances. In practice, it is operationalized by investigating the degree to which a parser can generalize to unseen utterances that are combinations of known fragments (e.g., phrases), which reflects the generative nature of natural languages. For example, we ask how well a parser can generalize to long utterances if exposed in training to only short ones.

From the machine learning perspective, the practical settings introduced above can be formulated as two kinds of generalization challenges: settings driven by transferability and linguistic coverage require *out-of-distribution* generalization as test data in such settings diverge from the observed training data in certain aspects (e.g., domains, length of utterances); settings based on alternative forms of supervision require learning from *weak learning signals*. The main contribution of this thesis is to address the generalization challenges via: 1) injecting alignment-based *model biases* into semantic parsers and; 2) designing specialized objectives with model-agnostic *learning biases* to train semantic parsers, in the hope that the resulting parsers can be more robust (e.g., to domain shifts) and take better advantage of weak learning signals. Empirical results on a wide range of semantic parsing benchmarks show that our methodologies are effective in handling the generalization challenges of interest, thus pushing existing natural language interfaces one step further towards real usage.

Lay summary

Natural language underpins how humans communicate with each other; it enables the rich expression and exchange of information. As computational systems become staples in more and more parts of our daily lives, how we communicate with *them* becomes increasingly important too. A smart speaker (e.g., Amazon Echo, Google Nest), which is typically not equipped with a graphical interface, uses natural language (via speech recognition) as the primary - and often only - means for communicating with users. This thesis looks at systems like these that enable communication with computer systems via natural language. Of the computational systems users could communicate with databases are of particular interest. They store huge volumes of structured information, and a user who does not know how to query a database should ideally still be able to access its contents without employing a technician. Being able to ask a database a question in natural language could allow users to straightforwardly get an answer whenever a question comes to mind.

Imagine you have a database about Scotland's geography, and our aim is to build you an interface where you can ask questions like '*what's the longest river in Scotland?*'. This could be done by collecting a huge number of user questions and hiring technicians to annotate them with the corresponding database queries, queries which computers can interpret and use to obtain an answer. Then we can build an intelligent system that learns from these question-query pairs so that the next time you ask a novel question the system hasn't seen before, it will still be able to return the answer by intelligently figuring out the corresponding computer queries itself, without the aid of a technician. In building a system like this, several challenging problems arise: like when you have other databases, e.g., one about Scottish farms, or Canadian geography - which our intelligent system hasn't seen before but you would still like to be able to ask questions about. Another challenge in building our system is that hiring technicians is expensive, so it would not be economical to request expert annotation for every database you could want to ask about. In this thesis, we ask specific questions that originate from these challenges, like whether or not we can build a universal natural language interface that works for all databases – or whether an intelligent system can learn well enough from just question-answer pairs (rather than question-query pairs) that we do not need an expert technician to annotate. This thesis focuses on developing new techniques that address these questions so that natural language interfaces can be built more efficiently and effectively, ultimately bringing us one step closer to the goal of effortlessly interacting with computational systems.

Acknowledgements

First, I would like to thank my supervisors Ivan and Mirella. I still remember around four years ago, after the first time I met both of them virtually during an interview, they offered me the long-desired chance to research semantic parsing, the topic I am really excited about. I feel grateful to have this timely opportunity that allows me to transform my research passion from that time to the thesis that you are reading now. I am also lucky to be advised by both of them, and they have shaped my academic thinking in different ways. Ivan taught me how to make sense of research work in a more theoretical and fundamental way; Mirella taught me to think about research problems from a broader perspective and guide me through research on semantic parsing. Their influences on me are complementary in a very interesting and valuable way.

Many thanks to Mark Steedman and Matt Gardner for examining the thesis. Some of the ideas in this thesis are greatly inspired by their work, and I am really happy to have a chance to interact with them via this thesis. I thank Alex and Matt for a really wonderful internship at Microsoft (before the pandemic!); Wenpeng and Victoria for a nice virtual internship at Salesforce. I would like to thank Adam Lopez and Shay Cohen to review my annual progress.

More than half of my PhD is spent during the pandemic, making me feel more grateful to all the colleagues and friends that I have interacted during this tough time. Many thanks to Henry and Tom for proofreading this thesis and for many constructive feedbacks. Thanks to Henry and Matthias for the reading group on grammar/structures, where I have a lot of fun talking with them. Thanks to members from Ivan's group, namely Caio, Diego, Michael, David, Serhii, Lena, Arthur, Nicola, Matthias, Verna, Antonio, Xinnuo, from whom I learned many interesting things outside my research area. Thanks to many friends with whom I share my spare time: Hao for practicing and playing badminton with me; Yanpeng for cycling together around Edinburgh; Biao, Bowen, Kai, Chunchuan, Shangmin, Shuping, Xinchi, Yang, Yao, Yumo, Zhijiang, Zhifeng for many random walks, meals, and chats; my office mate Shuzhuang for unlocking the door for me occasionally as I sometimes forgot to take the keys; Tao for many fun chats on research and PhD life.

Last but not least, I would like to give my special thanks to Bonnie Lun, who had always been there during every work that I have done and presented in this thesis. The thesis would be impossible without her support during the frustrating pandemic.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Bailin Wang)*

To my beloved parents.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

In recent decades, computational systems have significantly reshaped how users learn, think, and communicate with the world. In many scenarios, it is highly desirable for users to interact with computational systems using natural language (e.g., English). Representative scenarios include interactions with smart home automation devices, and interfaces to databases. A smart speaker (e.g., Amazon Echo), which is typically not equipped with graphical interface, resorts to natural language (via speech recognition) as the primary vehicle for receiving commands from users. For databases, which store structured information, a user who does not know how to query a database should ideally still be able to access its contents without employing a technician, whenever a question comes to mind. Being able to ask a database a question in natural language could straightforwardly enable this.

In general, a *natural language interface* is a transparent language layer that bridges humans and machines. Machine-interpretable special-purpose languages (aka *programs*) are typically designed so that computational systems can be controlled programmatically by humans. For smart devices, such a language can be a set of Application Programming Interface (API) calls; for databases, formal query languages are the primary mode of access and management. However, just like any foreign language, these machine languages require time-consuming special training to acquire and master. Moreover, the lack of a unified machine language implies that monolingualism is far from enough when it comes to conversing with machines in multiple applications. Natural language interfaces bridge natural and machine languages, offering ease of use and a unified mode of interaction of computational systems to end-users.

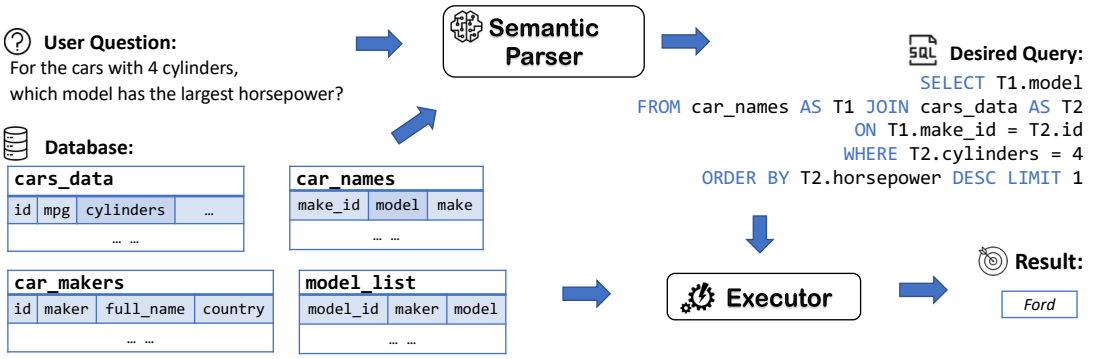


Figure 1.1: An example of natural language interface to a database about cars. Questions from a user are translated via a semantic parser to the desired SQL queries, which will be executed against the database to obtain results returned to users.

An established paradigm to building natural language interfaces is *semantic parsing*, which is an active research area within natural language processing (NLP) aiming to perform translation from natural to machine languages. For example, a natural language interface to databases can be built by a semantic parser illustrated in Figure 1.1. To successfully handle the mapping from natural language to Structured Query Languages (SQL), a semantic parser needs to not only obtain the semantics of the user utterances, but also be able to ground them with respect to the database. Concretely, such a model may have to handle mentions of relevant columns like 'cylinders', 'model' and 'horsepower', and table relations (e.g., table 'cars_data' and 'car_names' need to be joined). The grounding process and usage of special-purpose languages make semantic parsing in the context of natural language interfaces distinct from broad-coverage semantic parsing where the main goal is to represent the semantics of natural languages in general-purpose formalisms such as Combinatory Categorial Grammar (Steedman, 2000, CCG), and Abstract Meaning Representation (Banarescu et al., 2013, AMR), where no grounding is required.

This thesis primarily focuses on a more general setting of natural language interface to databases. Specifically, the concept of a database is generalized to *structured queryable data* so that it can cover many other common data sources like knowledge bases and Web tables, beyond relational databases. Accordingly, the conventional NLP problems of question answering over knowledge bases and Web tables are instantiations of this general setting. Other scenarios of natural language interface, for instance the interface to smart devices mentioned above, can reasonably be viewed as instantiations as well, where the concept of structured data refers to the ontology of API calls.

Semantic parsing's direct mapping to readily applicable programs is very appealing from the application perspective. However, such direct mapping results in additional computational complexity when developing an effective semantic parser. In the current deep learning era, state-of-the-art methods - which are neural models learning from examples of utterance-program pairs (Zettlemoyer and Collins, 2005) - are only successful in limited settings (e.g., semantic parsing in the context of a fixed database) and are typically quite data-hungry (e.g., they require many labeled examples for the target database). This thesis focuses on addressing the generalization challenges that arise for semantic parsers when deviating from the conventional training setting and data requirements in some real-life scenarios.

1.2 Challenges from Diverse Settings

We consider three aspects of semantic parsing tasks, inspired by real-life scenarios we will explain below. Each aspect establishes an axis along which the conventional setting deviates, thereby defining one or more new task settings. We walk-through each of these aspects and their resulting settings below. Most importantly, we will describe the challenges that arise in each setting, which are the main problems this thesis aims to address.

Transferability: from In-Domain to Cross-Domain Typically, semantic parsing tasks are defined in the *in-domain setting* where the goal is to develop a semantic parser for a particular domain (e.g., a relational database). For example, two conventional semantic parsing tasks GeoQuery (Zelle and Mooney, 1996) and ATIS (Dahl et al., 1994) are designed for the domain of US geography and airline travel, respectively. However, systems that are successful in such tasks cannot be transferred to other domains directly. Systems developed in such in-domain settings do not have to explicitly take into account the process of grounding user utterances to particular domains. Let us revisit the example shown in Figure 1.1. If the goal is to develop a parser for a particular database about cars, it is not necessary for a semantic parser to model the grounding process via linking relevant columns or tables such as 'horsepower', because it is very likely that the column name 'horsepower' is observed during training by the parser. This can then develop a simple - yet effective - strategy to exploit co-occurrences of natural language phrases and column names for generating (instead of linking) the column during prediction. Explicitly modeling the grounding process is crucial for

transferability because column names from a new database are probably not observed during training. Intuitively, the grounding process requires a parser comprehending both user utterances and databases so that the parser can link (instead of generate) the desired column when necessary. The *cross-domain setting* arises to test whether a system can be readily applied to other domains. Concretely the training and test domains in this setting are disjoint so that a system is expected to be able to ground utterances onto unseen domains during evaluation.

Annotation: Three Different Forms Semantic parsers are typically trained with *utterance-program pairs*, e.g., annotators need to write the corresponding SQL for the collected user question in Figure 1.1. Such annotation is labor-intensive: annotators are required to have expert knowledge of query languages like SQL, and be familiar with the structured data, e.g. which columns to use and which tables to join in a relational database. In the case where structured data is in the form of simple Web tables, it is possible to reduce cost by only annotating the answer (Liang et al., 2011; Berant et al., 2013), i.e., execution result (aka denotation) of the corresponding program, as such annotations do not require expert knowledge of programs. To learn from *utterance-answer* pairs, a parser is expected to search over all possible programs that can lead to the desired answer.

In the extreme setting where only *unannotated user utterances* are given, and no programs or answers are provided, can we still utilise them to improve a semantic parser? Such a setting resembles the common real-life scenario where massive numbers of user utterances can be collected after deploying an initial (reasonably good) natural language interface (Iyer et al., 2017). Effectively utilizing the unannotated utterances makes it possible for a semantic parser to improve over time without human involvement. To benefit from unannotated data weak learning signals need to be designed which the semantic parser can use.

Linguistic Coverage: Compositionality of Utterances In semantic parsing, target programs for machines to interpret and execute are intrinsically compositional in the sense that they are defined and restricted according to some underlying grammar (e.g., the grammar of SQL). Compositionality allows the expression of exponentially many meanings with a limited set of grammar rules by effectively abstracting and reusing certain sub-expressions. Natural language also exhibits compositionality which enables compact and efficient representations of the same underlying meanings as

Natural Language Utterances	Programs
what is the length of the colorado river ?	`len( river( riverid ( 'colorado' ) ) )`
what is the longest river ?	`longest( river( all ) ) )`
what is the length of the longest river ?	`len( longest( river( all ) ) )`

Table 1.1: Three utterance-program pairs from the GeoQuery dataset. To express a compound intention in the last example, the program is effectively a combination of fragments of other programs; interestingly, natural language also exhibits a similar phenomenon where the utterance reuses and combines phrases (e.g., 'the length' and 'the longest river') from other utterances.

programs. Table 1.1 gives an example of compositionality from the GeoQuery dataset elucidating this phenomenon. Though natural language has sequential surface forms and no explicit structure (e.g., tree structure), it has long been believed that natural language is compositional as a result of the (potentially innate) process that governs the syntax of natural language, e.g., generative grammar (Chomsky, 1965). To develop a semantic parser that covers a wide spectrum of user intents (i.e., high linguistic coverage) using a limited number of examples, developing strategies that can capture compositionality seems to be necessary for generalization. Whether or not neural models can acquire an underlying compositional process, often called *compositional generalization*, is also interesting from the perspective of general artificial intelligence. Human learners can interpret the test example in Table 1.1 - to a great extent - as compositionality allows them to break novel examples down into known atoms (Cann, 1993). Whether neural models can generalize in this way, or how to develop systems capable of the kinds of compositional reasoning characteristic of the human mind is a long-standing research question (Fodor and Pylyshyn, 1988; Lake and Baroni, 2018).

1.3 Tackling Generalization Challenges

From the machine learning perspective, the three challenges introduced above can be intuitively framed as the following problem: we need to develop a semantic parser based on data from a certain training distribution $\mathcal{T}_{train}$, and expect the resulting parser to perform well on data from a certain test distribution $\mathcal{T}_{test}$. Each setting effectively specifies a pair of $<\mathcal{T}_{train}, \mathcal{T}_{test}>$, and our goal is to tackle the *generalization challenges* that arise in each pair. For example, transferability can be framed as a generalization

	Out-of-Distribution	Weak Supervision
Domain Transferability	✓	
Cheap Forms of Supervision		✓
Linguistic Coverage	✓	

Table 1.2: From the machine learning perspective, domain transferability and linguistic coverage require out-of-distribution generalization of a semantic parser; different forms of supervision requires generalization from cheap but weak supervision.

challenge where $\mathcal{T}_{train}$, $\mathcal{T}_{test}$ are from some observed source (i.e., training) domains and unseen target (i.e., test) domains.

By characterizing the pair $<\mathcal{T}_{train}, \mathcal{T}_{test}>$ of each setting, we categorize the three challenges into two groups, namely out-of-distribution generalization and learning from weak supervision, where the problems of domain transferability and linguistic coverage are grouped together. These correspondences are summarized in Table 1.2. We elaborate on each group of generalization challenges below. Moreover, as the main contribution of this thesis, we will present a shared set of methodologies, including modeling alignments and specialized training objectives, to address both kinds of challenges.

1.3.1 Out-of-Distribution Generalization

In a standard supervised learning paradigm, the training and test examples are sampled independently and identically distributed (i.i.d.) from the same distribution, i.e., $\mathcal{T}_{train}$ and $\mathcal{T}_{test}$ are identical. However, this paradigm cannot cover cases of transferability and linguistic coverage, where $\mathcal{T}_{train}$ and $\mathcal{T}_{test}$ are divergent. Such settings require *out-of-distribution* generalization.

- In instances of domain transferability (i.e. cross-domain semantic parsing) the divergence between $\mathcal{T}_{train}$ and $\mathcal{T}_{test}$ is obvious as they are distributions based on two disjoint sets of domains. A typical sequence-to-sequence model trained with standard supervised learning would presumably suffer from overfitting to the source domains. For example, a parser could memorize the mentions of tables/columns (e.g., 'cars' to 'cars_data' in Figure 1.1) which cannot help generalization (e.g., 'cars' to 'cars_name' in a new database). Encouraging a parser to condition on the environment appropriately, we expect it to rely on some

generalizable functions or features (e.g., semantic matching between 'cars' and 'cars_data').

- In the linguistic coverage setting, i.e. whether a system can interpret all possible programs combinatorially composed of known segments, the divergence between $\mathcal{T}_{train}$ and $\mathcal{T}_{test}$ is less obvious. It is specifically created to challenge a parser's generalizability to new examples with known segments in an effort to test whether a parser can acquire the underlying grammar-like compositional regularity of natural language. The basic assumption in this setting is that there are some underlying rules that dictate the mapping from natural language utterances to programs. To reflect this assumption, $\mathcal{T}_{train}$ is typically assumed to cover all the basic rules whereas $\mathcal{T}_{test}$ contains novel compositions of these rules. In this sense, the divergence between $\mathcal{T}_{train}$ and $\mathcal{T}_{test}$ lies in different distribution of grammatical structures [1]. Since the true underlying rules are unknown, as proxies, $\mathcal{T}_{train}$ and $\mathcal{T}_{test}$ are created based on some measurable properties that correlate with the underlying compositional rules. The most understandable setting is based on splitting according to length of examples (Lake and Baroni, 2018): $\mathcal{T}_{train}$ only cover utterances (or programs) with length smaller than a threshold, while $\mathcal{T}_{test}$ covers those with length greater than the threshold. Another example is shown in Table 1.1, where a parser is expected to generalize the last example after being trained on the first two examples. The divergence lies in that the grammatical structure of 'length of the longest river' is novel, while the grammatical structures of its segments 'the length' and 'the longest river' are not. There are other properties (e.g. Finn et al., 2017; Keysers et al., 2019) that can be used to create divergent distributions.

Out-of-distribution generalization in general is a difficult machine learning problem, and has gained soaring interest in recent years (Arjovsky, 2020). In the context of semantic parsing, the basic way of handling this problem is incorporating prior knowledge into the modeling process, e.g., universal regularities of mapping utterances to programs, and the way $\mathcal{T}_{train}$ diverges from $\mathcal{T}_{test}$. Specifically, we propose new semantic parsers that take into account structured alignments invariant across $\mathcal{T}_{train}$ and $\mathcal{T}_{test}$, and specialized training objectives to learn a parser based on our assumptions about the

[1] More concretely, grammatical rules can include both lexical and structural rules. Intuitively, a divergent distribution of lexical rules means that a word or phrase occurs in a novel position at test time, while a divergent distribution of structural rules means that the test set contains sentences with novel syntactic structures. Kim and Linzen (2020) provide detailed explanation on this.

divergence. We elaborate on each of these methodologies below.

Modeling Alignments Our first methodology is based on the intuition that although $\mathcal{T}_{train}$ and $\mathcal{T}_{test}$ differ in certain ways (e.g., domains), they may share key common features that can be leveraged for robust semantic parsing. The main commonality investigated in this thesis is the correspondence (aka alignments) between utterances and programs: how natural language fragments are linked with program fragments. For example, in Figure 1.1, '4 cylinders' should be aligned with the condition `cylinders = 4`, 'largest' with `ORDER BY ... DESC LIMIT 1`. Intuitively, modeling alignments encourages a parser to find fine-grained correspondences between fragments; this would hopefully prevent it from memorizing large spans of training examples. In the naive case, a parser could use a simple strategy to keep an utterance-to-program dictionary, memorizing data at the example level, so that it can perfectly fit the data from $\mathcal{T}_{train}$. But such a strategy would not generalize to unseen utterances. In practice, although neural models seem not exhibit the naive strategy they are found to memorize large spans of training utterances (Hupkes et al., 2019). That is, current models seem to rely on coarse-grained large-span level alignments to fit $\mathcal{T}_{train}$. To address this we explicitly model the underlying fine-grained alignments to favour better generalization.

Alignments between natural language and program fragments are not annotated and thus unknown. Typically in neural models the attention mechanism (Bahdanau et al., 2015) is believed to be responsible for handling the correspondences between input and output. In this thesis, we propose to either upgrade or replace the attention mechanism: 1) we inject some discrete relations into the standard attention mechanism so that attention weights can be guided; 2) instead of relying on the standard attention, we introduce *latent-alignment models* that explicitly accommodate the underlying alignments via treating them as discrete latent variables.

Specialized Training Objectives Our second approach promotes out-of-distribution generalization via specialized training objectives that augment conventional training. The basic idea is to mimic the divergence between $\mathcal{T}_{train}$ and $\mathcal{T}_{test}$ within the training data to enable a learning algorithm to learn a strategy robust to that divergence. To do this we create a set of virtual tasks – each containing a virtual train-test split. For example, if the divergence is based on domain, where training and test data may be from disjoint domains, we can mimic it by subsampling two disjoint training domains as virtual train and virtual test respectively. To take advantage of this mimicked divergence,

we employ a meta-learning algorithm operating on the created virtual tasks. The basic intuition behind the algorithm is that, during the course of training, it encourages gradient steps to not only reduce the loss on virtual train split (i.e., examples from virtual train domains), but also be beneficial on the corresponding virtual test split, (i.e., examples from virtual test domains).

1.3.2 Learning from Weak Supervision

The generalization challenges that arise from cheap forms of annotation are straightforward: a parser needs to learn from weak signals in the form of denotations or unlabeled data. A bit more formally, typical training data in semantic parsing comes in the form of triples $\{(x_i, e_i, y_i)\}_{i=1}^{n}$, where x, e, y denotes a natural language utterance (or question), the context of the utterance (e.g., a relational database) and the corresponding program. We consider two settings with cheaper data: 1) learning from denotations, where training examples are in the form of $\{(x_i, e_i, d_i)\}_{i=1}^{n}$, where d_i is the answer (or denotation) of question x_i ; 2) learning from unlabeled utterances, where apart from typical $\{(x_i, e_i, y_i)\}_{i=1}^{n}$, a large amount of unlabeled data in the form of $\{(x_i, e_i)\}_{i=1}^{m}$ is also given. To handle these two settings, we employ the same methodologies of modeling alignments and specialized training objectives discussed earlier.

Modeling Alignments When learning from denotations the correct program for each question is not provided, meaning a parser needs to explore a large search space of possible programs to find those whose execution can result in the given denotation. One major challenge in this setting is that a parser should be robust to spurious programs which accidentally execute to a given denotation, but do not reflect the semantics of the question. Our intuition is that correct programs would likely respect certain constraints were they to be aligned to the question text, while spurious programs would not. We capitalize on this intuition and capture structural constraints by modeling latent alignments between programs and questions.

Specialized Training Objectives Learning from unlabeled utterances is effectively a semi-supervised learning problem. Our key observation for unsupervised learning on unlabeled utterances is that not all candidate programs for an utterance will be semantically valid. If we were to try and execute all candidate programs, only some of them could be executed (i.e., without triggering errors from the executor) and obtain non-empty execution results. Based on the assumption that correct programs should

also be executable, we encourage a parser to only focus on executable programs via specialized training objectives. Such objectives are also incorporated with certain structural biases. For example, sparsity, which further encourages a parser to only focus on a subset of executable programs, can be injected into training objectives.

1.3.3 Summary of Methodology

Though motivated by two different generalization challenges, we arrive at a similar set of methods. At the core of both modeling alignments and specialized training objectives are injecting *inductive biases*, or prior knowledge, into parsers. We make certain assumptions in each setting about the way a semantic parser should generalize, like how natural language utterances and programs should be aligned, or what out-of-distribution test data will look like. This knowledge is not explicitly imported into the model (e.g., in terms of grammars extracted from external tools). Instead it is specified via computational components with certain regularities, like an alignment network constraining the way natural language utterances can be aligned with programs or a training objective encouraging certain behaviors of models.

In comparison, modeling alignments and specialized training objectives incorporate inductive biases in quite different manners. The former can be viewed as a *model bias*, where the architecture of a semantic parser is designed to accommodate alignments. The latter can be viewed as a *learning bias*, where instead of upgrading a semantic parser, the training objectives of the parser are manipulated. Injecting learning biases is based on the implicit assumption that a parser is powerful enough to solve a task, but too flexible and lacking sufficient guidance to arrive at a desired strategy. These methodologies enjoy distinct properties were they to be extended to other scenarios: 1) specialized training objectives are model-agnostic and can be applied to any semantic parser, implying their more straightforward application to existing models than modeling alignments; 2) Modeling alignments between input and output sequences is not unique to semantic parsing. Accordingly this approach can be applied to a wider array of tasks, including machine translation, than specialized training objectives.

1.3.4 Generalization to Other Types of Semantics

This thesis explores semantic parsing as a paradigm for natural language interfaces to structured queryable data, such as relational databases and knowledge bases. Hence, the formal languages that we primarily focus on are mainly developed from the perspective

of data querying and management, e.g., SQL for relational databases and domain-specific languages for querying knowledge bases. This is in contrast to semantic parsing for other types of semantics, such as broad-coverage natural language semantics, e.g., CCG (Steedman, 2000) and lambda calculus (Liang et al., 2013), which are designed from the linguistic perspective; general-purpose language (e.g., Python, C++), which are designed for modelling general computations. The crucial distinction is that these formalisms are not grounded in the sense that they cannot be readily used to generate desired outcomes for natural language interfaces. As a result, semantic parsing with them does not directly exhibit generalization challenges such as transferability and learning from weak supervision.

However, our two methodologies of injecting model and learning biases can still be used for other semantic formalisms. Specifically, some instantiations of the two methodologies, which we will elaborate in this thesis, do not depend on the grounding aspect of formalisms. Specifically, our first methodology of injecting structural alignment biases into semantic parsers can be used across semantics because alignments can be defined only at the surface level where no grounding is required. Second, some specialized training objectives are based on surface-level features of programs and can be adapted to facilitate semantic parsing for other formalisms. For example, in our recent exploration (Conklin et al., 2021), we adapt the training algorithm for transferability (Chapter 3) to boost the compositional generalization of semantic parsing for natural language semantics.

1.4 Thesis Outline

In the remainder of this thesis, we first provide some essential background on semantic parsing. Then each remaining chapter details our methods to address generalization challenges that arise from the settings laid out above. A brief map of the main chapters can be found in Table 1.3. We provide a summary of each chapter below.

Chapter 2 details the semantic parsing framework we will use throughout this thesis, including task definition, basic components of our parser, datasets and evaluation metrics for semantic parsing. We also situate our methodologies by relating them to previous work on semantic parsing.

	Model Alignments	Specialized Training Objectives
Domain Transferability	Chapter 3	Chapter 3
Cheap Forms of Supervision	Chapter 4	Chapter 5
Linguistic Coverage	Chapter 6	Chapter 6 [2]

Table 1.3: A brief map of main chapters of the thesis from the modeling perspective. The three challenges are approached from modeling alignments and specialized training objectives.

Chapter 3 studies the setting of cross-domain semantic parsing. We first present a new parser framework called RAT-SQL. It is designed to accommodate the alignments and relations required for cross-domain text-to-SQL parsing, based on the relation-aware self-attention mechanism. We further present a model-agnostic meta-learning algorithm, called DG-MAML, that can further encourage RAT-SQL to generalize to unseen domains.

Chapter 4 presents our latent-alignment parser for learning from denotations. To facilitate tractable alignments, we first decompose the parsing task into predicting a partial abstract program, then refining it while modeling structured alignments with differential dynamic programming.

Chapter 5 presents our work on semi-supervised semantic parsing. To effectively utilize unlabeled data, we use executability as a weak yet free learning signal. Due to the large search space of executable programs conventional methods that use approximations based on beam-search, such as self-training, do not perform as well. To address this, we view the problem of learning from executions from the perspective of posterior regularization and propose a set of new training objectives.

Chapter 6 introduces our general latent-alignment sequence-to-sequence model for boosting linguistic coverage of a semantic parsing by improving its systematic generalization. We draw inspiration from traditional grammar formalisms which would excel at systematic generalization given suitable rules. We assume that their generalizability comes from implicitly encoding alignments between input and output segments using

[2] This cell is based on a collaboration work Conklin et al. (2021), which is not a part of the thesis, we place it here for the completeness of the map. We will only briefly describe it in Chapter 6.

rules. Based on this assumption we directly model segment-to-segment alignments as discrete structured latent variables within a neural sequence-to-sequence (seq2seq) model. To efficiently explore the large space of alignments we introduce a reorder-first align-later framework whose central component is a neural reordering module producing separable permutations. The resulting model can be efficiently trained in an end-to-end manner via dynamic programming.

Chapter 7 summarizes the findings of the thesis. We also discuss how our methodologies can be potentially extended to other related settings in semantic parsing.

Related Publications The main chapters are based on the following published work.

- Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*. (**Chapter 3**)

- Bailin Wang, Mirella Lapata, and Ivan Titov. 2021b. Meta-learning for domain generalization in semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*. (**Chapter 3**)

- Bailin Wang, Ivan Titov, and Mirella Lapata. 2019. Learning semantic parsers from denotations with latent structured alignments and abstract programs. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. (**Chapter 4**)

- Bailin Wang, Mirella Lapata, and Ivan Titov. 2021a. Learning from executions for semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*. (**Chapter 5**)

- Bailin Wang, Mirella Lapata, and Ivan Titov. 2021c. Structured reordering for modeling latent alignments in sequence transduction. In *Advances in Neural Information Processing Systems (NeurIPS)*. (**Chapter 6**)

Reproducibility Unless stated otherwise, code and data are available at https://github.com/berlino/tensor2struct-public.

Chapter 2

Background

Semantic parsing provides a unified framework for building natural language interfaces. In this chapter, we will first lay out the problem definition, the basic components of a semantic parsing framework along with evaluation metrics we use throughout the remaining thesis. We then introduce three representative semantic parsing tasks. To position our methodologies, we discuss some related systems that have previously been proposed for semantic parsing. Finally, we position semantic parsing with respect to related areas from a modeling perspective.

2.1 Semantic Parsing Framework

Task Definition Given a natural language question x in the context of an environment e (e.g., a relational database), semantic parsing aims to map x to a corresponding program y, which can be executed against e to obtain an answer or denotation d, denoted as $[[y]]_e = d$. Given a set of examples, e.g., $\{(x_i, e_i, y_i)\}_{i=1}^n$, we would like to build a system that can perform the mapping for new questions and new environments (e.g., new relational databases). Such generalization is possible as the typical assumptions made for this task are that new questions consist of known fragments (e.g., phrases) seen from the training examples, and elements of new environments (e.g., column/table names of new relational databases) are expressed in natural languages so that a system can interpret them by "reading" them.

Four Components The semantic parsing framework used in this thesis has four basic components listed below [1].

[1] They are based on and effectively equivalent to the five components defined by Liang (2016).

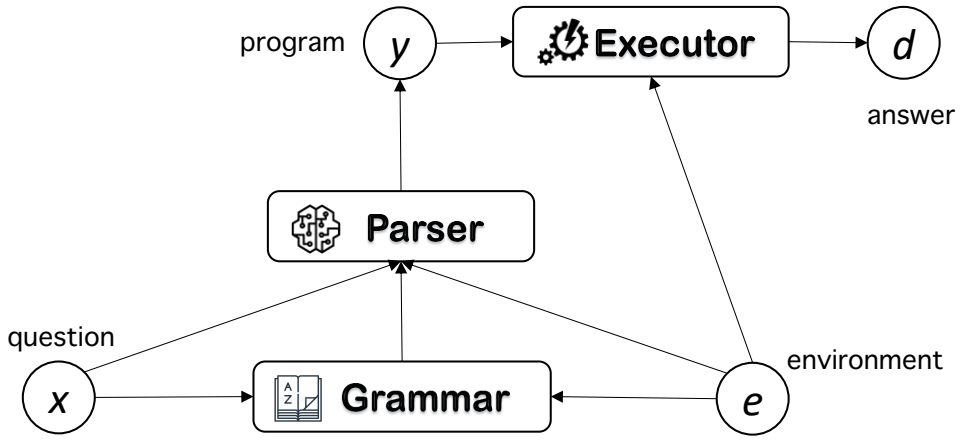


Figure 2.1: Components for semantic parsing. Given question x and its environment e, a parser assigns a score (i.e., probability) to programs that are licensed by a environment-specific grammar. To obtain the desired parser component that properly assigns scores to correct programs, a trainer component is also required to search for optimal configurations of the parser.

- **Grammar**: a set of grammar rules that constrains the set of valid programs given question x and environment e.
- **Executor**: executes programs against environment e to obtain an answer d.
- **Parser**: $p_\theta(y|x,\theta)$ parameterized by θ assigns probability to possible programs.
- **Trainer**: searches for optimal parameters for θ given training examples.

The components are depicted in Figure 2.1. In the next section, these components will be instantiated in three representative tasks for building natural language interfaces.

Evaluation To evaluate our semantic parser, we apply it to some new test questions (which may also be in new environments). Two metrics are typically used:

- *Program Accuracy* where a predicted program is correct if it has the exact same surface form as the gold program;
- *Execution Accuracy* where a predicted program is deemed correct if it obtains the same execution results as the gold program.

Unfortunately neither metric is perfect: there might exist semantically equivalent programs that look different (false negatives) or semantically distinct programs that

Question	What restaurant can you eat lunch outside at?
Lambda-DCS	`type.restaurant ⊓ hasOutdoorSeating ⊓ serveslunch`
Question	what is the length of the colorado river ?
FunQL	`len( river( riverid ( 'colorado' ) ) )`
Question	What is the number of cars with more than 4 cylinders?
SQL	`SELECT COUNT(*) FROM cars_data WHERE cylinders` > 4

Table 2.1: Examples of query languages from Overnight (Wang et al., 2015), GeoQuery (Kate et al., 2005) and Spider (Yu et al., 2018c), respectively.

happen to obtain the same execution results (false positives). But from a end user's perspective, evaluating execution results are desired as they are the ultimate return from natural language interfaces, rather than programs. In this thesis, we will report both performance in both metrics whenever possible. In Chapter 3, we also use use a metric, called exact set match, that determines a program to be good if it is partially matched with the gold program. This metric is primarily used during the early stage of development, where program accuracy and execution accuracy are too low to measure the progress being made.

2.2 Representative Tasks

The main goal of the thesis is to build natural language interfaces to structured data. We consider three representative tasks, each with a different form of structured data. For each task, we identify the formal language used to query the structured data, and some datasets that have been commonly used to benchmark research progress. We also provide some examples from these tasks in Table 2.1.

Interfaces for Knowledge Bases A knowledge base contains two parts: an ontology and a set of facts. In the simplest form, it can be viewed as a collection of assertions in the form of triples (e_1, p, e_2), where e_1, e_2 are entities (e.g., journal_1, 1993) and p is a property (e.g., publicationDate). To query a knowledge base, it is typical to use lambda-DCS (Liang et al., 2013), which is a compact variant of lambda calculus which eliminates variables and makes existential quantification implicit. It can be viewed as a specialized version of lambda calculus adapted for notational convenience for building natural language interfaces for knowledge bases.

Based on the availability of large-scale open-domain knowledge bases such as Freebase (Bollacker et al., 2008), early work (Berant et al., 2013; Yih et al., 2015) use semantic parsing as a means for (open-domain) question answering over such knowledge bases. Despite the appealing scale, datasets for such tasks only contain relatively simple questions. To match practical scenarios of having specialized domains and domain-specific jargon, the Overnight dataset (Wang et al., 2015) provides knowledge bases from eight domains along with complex questions. The first row of Table 2.1 shows an example from the Overnight dataset.

Interfaces for Relational Databases Relational databases [2] are the primary vehicle for storing information in many applications such as financial markets, and medical records. A standard way to query relational databases is using SQL (Structured Query Language), which is a specialized language based on relational algebra. Less popular query languages are Prolog (Zelle and Mooney, 1996) and a simple functional query language FunQL (Kate et al., 2005) which are mostly used in the traditional benchmark GeoQuery (Zelle and Mooney, 1996). Compared with lambda-DCS used for querying knowledge bases, these query languages are very convenient for end applications. Although they tend to be less suitable for semantic parsing as they are not designed to represent the semantics of natural language, which should make learning the mapping between natural language and query language more challenging.

Early datasets like GeoQuery (Zelle and Mooney, 1996) and ATIS (Dahl et al., 1994) only consider designing interfaces for a single database. As a result, a semantic parser is evaluated on utterances in the same database as the one used during training. Different from this single-domain setting, recent datasets such as Spider (Yu et al., 2018c) explore the cross-database setting where the sets of databases used for training and evaluation are disjoint. This setting is more practical as generalization to unseen databases entails a universal natural language interface applicable to any target application and a particular database.

Interfaces for Web Tables Another rich source of structured data is HTML tables from Web pages. Specifically, tables from Wikipedia pages can be extracted and serve as an update-to-date repository of open-ended information. Compared with knowledge bases and relational databases extracted Web tables are simpler in structure - as a table

[2] One major difference between knowledge bases and relational bases is that they are built upon ontologies and relational schemas, respectively. This difference also contributes to the divergence of their query languages.

schema is simply a short list of headers. Moreover, Web tables are semi-structured in that cell values are usually not normalized (e.g.,'100 cm' needs to be converted to the number 100 with the unit 'cm'). Hence data normalization is a key additional step required for semantic parsing over Web tables. Query languages for Web tables are quite flexible: lambda-DCS (Pasupat and Liang, 2015), and SQL (Zhong et al., 2017) have both been used in previous work.

Due to the availability and domain coverage of Wikipedia pages, datasets of Web tables are usually large-scale and have expanded the scope of semantic parsing from limited to open domains. WikiTableQuestions (Pasupat and Liang, 2015) and WikiSQL (Zhong et al., 2017) are two representative datasets. WikiTableQuestions contains around 22k examples with around 2k tables; WikiSQL contains around 81k examples with 24 tables. In contrast to WikiTableQuestions, WikiSQL is larger in size but lacks complexity in the semantics of its questions, i.e., questions from this datasets are much simpler and do not require complex logic operators such as `argmax`. Similar to cross-database semantic parsing, the Web tables for training and test in WikiTableQuestion and WikiSQL are disjoint, implying that a semantic parser is required to generalize to novel Web tables. It is worth noting that semantic parsing is different from a related open-domain question answering setting; where the desired Web table to answer a particular question is not given and needs to be retrieved. In our case the corresponding Web table for a question is already provided so the research emphasis is placed on resolving the semantics of questions rather than table retrieval.

Remarks on Components for Interfaces In summary, each kind of task requires a different query language for programmatical interaction. To incorporate the query language of a task, the grammar and executor components of a semantic parser vary depending on the environment in the task. But these components are typically readily available, e.g., they can be existing SQL grammars and SQL engines for execution for building interfaces to relational databases. Hence, with existing grammar and executor components at hand, the focus of this thesis is on the top two components, namely the parser and the trainer component. The parser component is also affected by the form of a query language, especially in our case alignments between natural language utterances and query languages are explicitly modeled in this component. In general, the choice of query language can have a significant impact on performance (Guo et al., 2020), and choosing an appropriate query language for modeling alignments is also an important but less acknowledged step when building semantic parsers. It is worth

noting that finding a suitable formal language for semantic parsing requires further effort (Liang et al., 2013). In contrast, the trainer component is task-agnostic and it is typically based on the standard training objective of maximizing the likelihood of correct programs. The specialized training algorithms we will introduce in Chapter 3 and Chapter 5 improve the standard trainer component, and they can be applied both across tasks and across parsers.

2.3 Symbolic vs Neural Semantic Parsers

Like many other NLP areas, systems for semantic parsing have shifted from early rule-based parsers (e.g., Woods, 1973; Winograd, 1971; Hendrix, 1982) to statistical parsers (e.g., Wong and Mooney, 2006, 2007a; Lu, 2014; Zettlemoyer and Collins, 2005; Liang et al., 2013) to recent neural models (e.g., Dong and Lapata, 2016; Jia and Liang, 2016). The transition has been heavily influenced by progress in machine learning and deep learning. See Kamath and Das (2018) for a comprehensive survey of the history of semantic parsing. Here we review systems that are very related to our work from the perspective of *symbolic vs neural construction*, which will be explained below.

One key ingredient of rule-based and statistical systems is symbolic construction mechanism, i.e., the process where programs are constructed according to some rules or grammars. Symbolic construction is linguistically appealing as it explicitly models the coupling (i.e., alignment) of natural language and programs; accomodating certain language phenomena (e.g., anaphor) can be achieved specifically via rule or grammar engineering. The seminal work Zelle and Mooney (1996) proposes CHILL which uses inductive logic programming to learn control rules for parsing. Zettlemoyer and Collins (2005) relied on rules designed in Categorical Combinatorial Grammar (CCG, Steedman, 2000), which construct lambda calculus grounded to natural language in a bottom-up compositional manner using only a handful of combinators (e.g., type raising and function composition). Later, Liang et al. (2013) proposed another grammar formalism lambda-DCS to simplify the construction of lambda calculus based on a generalization of dependency trees. Hybrid-tree from Lu (2014) is another grammar formalism for semantic parsing inspired by synchronous grammars. In contrast, the standard neural sequence-to-sequence models construct programs sequentially via the attention mechanism. The critical difference lies in the concept of alignment: symbolic construction explicitly model the correspondences between natural language

words/phrases and program fragments whereas neural construction relies on soft alignments (i.e., attention).

The transition from symbolic construction to neural construction is mainly due to efficiency of the attention mechanism, which allows efficient building and training of large neural systems. However, symbolic construction becomes appealing again when confronting challenges such as learning from weak supervision and generalization to longer utterances as symbolic rules, when correctly identified, can generalize much robustly than neural construction (Herzig and Berant, 2021; Shaw et al., 2020; Akyurek and Andreas, 2021). The parsers we will present in Chapter 3, 4 and 6 are the attempts to combine the best of both symbolic and neural constructions, and a step towards neural-symbolic systems. Intuitively, we would like neural systems with soft rule-like structured alignments which enjoy efficient training and better generalization.

2.4 Related Areas

Relation to Machine Translation Semantic parsing can be regarded as a special instance of machine translation where the source language is a natural language (e.g., English) whereas the target language is a machine-interpretable formal language. It is also evident by the resemblance between some successful semantic parsers and machine translation systems in both statistical and neural eras. To name a few statistical systems, Wong and Mooney (2006) employ synchronous context-free grammar (Aho and Ullman, 1973), which forms the basis of many successful syntax-based machine translation systems (e.g., Chiang (2005)); Andreas et al. (2013) linearize target programs into sequences and rely on the phrase-based machine translation system (Koehn et al., 2003); Jones et al. (2012) employ tree-transducers for semantic parsing inpired by their usage in machine translation (e.g., Yamada and Knight (2001)). The recent shift to neural semantic parsers is influenced by the success of sequence-to-sequence models in neural machine translation (Bahdanau et al., 2015).

Relation to Program Synthesis Semantic parsing is also very connected with program synthesis (Gulwani et al., 2017); it can be even viewed as a special instance in program synthesis where program specification is given in the form of natural language (as opposed to other forms such as input-output examples). The subtle difference lies in the motivation behind the two tasks: semantic parsing is usually coupled with a practical application such as building a natural language interface. Hence, apart from

synthesizing a program, semantic parsing in general also needs to cover other human interaction problems, e.g., what if users need to ask a sequence of thematically related questions to fullfil a complex goal (Yu et al., 2019b). In contrast, program synthesis focus more on the methodology of searching for the desired program based on any kind of manual specification, including natural language. Hence, from the modeling perspective, two areas share some similarities. The problem we address in Chapter 5 is partially inspired by related work in program synthesis. Systematic generalization, which is the topic of Chapter 6, has also motivated much work (e.g., Nye et al., 2020; Chen et al., 2020) in program synthesis.

Chapter 3

Domain Generalization

In the context of semantic parsing, the notion of domain is realized by the *environment* in which user utterances are issued. For example, in text-to-SQL parsing, a domain refers to a relational database; in question answering against knowledge bases, a domain trivially refers to a particular knowledge base [1]. Conventional semantic parsing tasks are typically defined as *in-domain* settings where model development and evaluation are restricted to a particular domain. For example, a parser developed for the GeoQuery (Zelle and Mooney, 1996) dataset only needs to handle questions asking about US geography. This is desirable in the case where a natural language interface is created for a fixed domain. But this is undesirable in a common scenario: to add a new domain or apply a parser to a new application with new domains, we have to start from scratch, e.g., collecting training data for new domains, without exploiting common knowledge shared across domains so that the required human effort can be reduced.

Motivated by the practical scenario, the *out-of-domain* (aka cross-domain) setting has been gaining interest in the NLP community. In this setting, semantic parsers that are transferable to new domains need to be developed. Figure 3.1 shows a simple example of cross-domain semantic parsing. From the modeling standpoint, building semantic parsers in the out-of-domain setting is hard due to new challenges that are not encountered in the conventional in-domain setting. In this chapter, we aim to address these new challenges to obtain better transferable semantic parsers. To probe transferability in a cross-domain semantic parsing task, a parser is trained on some observed source domains, but evaluated on some new target domains. It is obvious that tailoring a semantic parser for training domains, which worked in in-domain settings,

[1] In the remainder of the thesis, the notion of domain, environment are interchangeable with their instantiations such as databases, knowledge bases or Web tables in the context of the respective task.

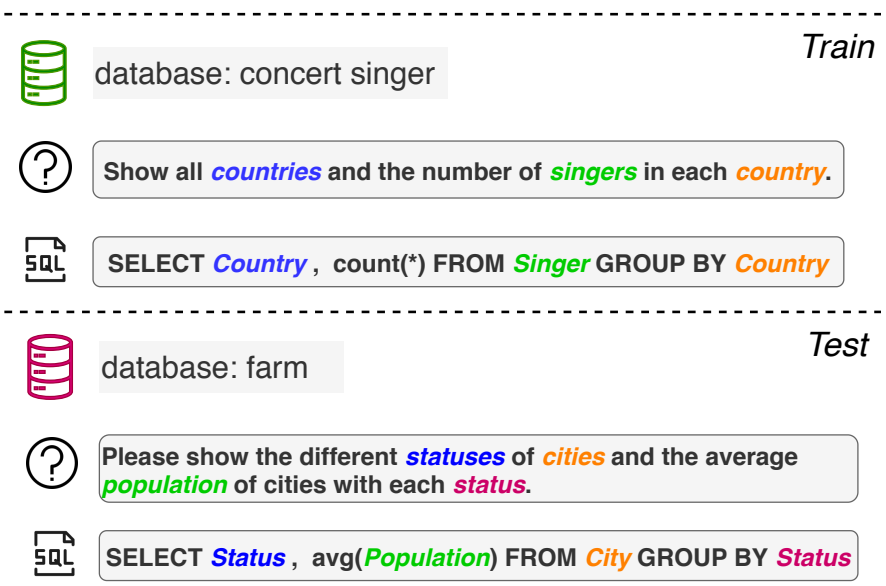


Figure 3.1: Cross-domain semantic parsing: at training time, a parser observes instances for the database *concert singer*. At test time, it needs to generate SQL for questions pertaining to the unseen database *farm*. Examples are taken from the Spider dataset (Yu et al., 2018c).

would not be a good strategy to handle out-of-domain settings. Instead, a cross-domain semantic parser is expected to capture some general functionality or features that are shared across domains.

To succeed in the cross-domain setting, a parser needs to achieve *domain generalization*, i.e., the ability to generalize to domains that are unseen during training. Achieving this goal would, in principle, lead to a universal natural language interface that allows users to interact with data in arbitrary domains. This is also very intriguing from the perspective of building semantic parsers: the ideal domain generalization entails that a parser trained on a fixed amount of data from a limited number of domains are applicable to any other new domains, i.e., eliminating the effort required for new domains at all. Though in practice this assumption is hardly true, the extent to which domain generalization is achieved has a significant impact on the additional effort (e.g., data annotation) required to build a desirable parser for a new domain, as intuitively we can start from a reasonably good parser trained on other domains instead of starting from scratch. This chapter aims at identifying challenges of achieving domain generalization and presents our methodologies for improving domain generalization.

Firstly, we identify two new challenges that arise in the setting of cross-domain semantic parsing.

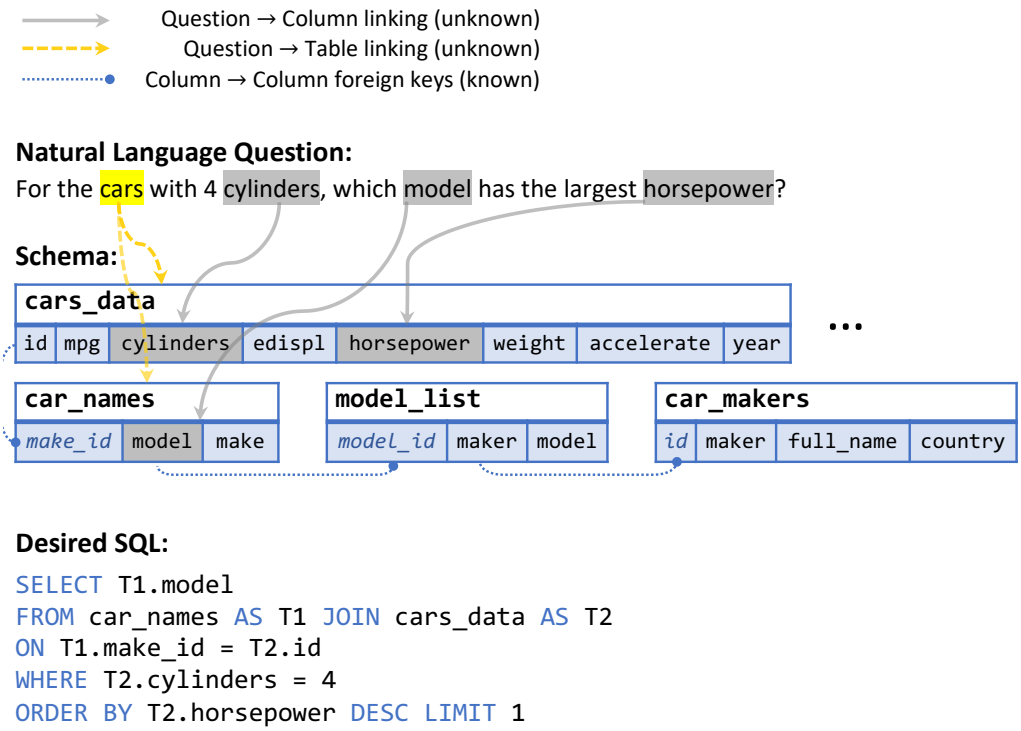


Figure 3.2: A challenging text-to-SQL task from the Spider dataset (Yu et al., 2018c).

- *Environment Conditioning*: Typical in-domain semantic parsers cannot be trivially adapted for cross-domain parsing as they lack a mechanism of conditioning on (potentially new) environments.
- *Optimization Mismatch*: Optimization in the conventional supervised learning paradigm does not encourage generalization to unseen domains. In contrast, it probably leads to overfitting observed training domains.

That is, conventional models and the training strategy, which are developed for in-domain settings (Finegan-Dollak et al., 2018), do not directly transfer to the challenging cross-domain settings. In this chapter, we propose to upgrade semantic parsers and their training strategy to address the two challenges respectively. We will use text-to-SQL parsing as a running example of cross-domain semantic parsing, but the methodology we will present can be adapted to environments other than relational databases [2].

Environment Conditioning In the context of cross-domain text-to-SQL parsing, domain generalization requires generalization to unseen database schemas. Schema

[2]For example, we use the proposed parser framework for semantic parsing against knowledge bases in Chapter 5.

generalization is challenging for three interconnected reasons. First, any text-to-SQL parsing model must encode the schema into representations suitable for decoding a SQL query that might involve generating references to columns or tables. Secondly, these representations should encode all the information about the schema such as its column types, foreign key relations, and primary keys used for database joins. Finally, the model must recognize natural language phrases used to refer to columns and tables, which might differ from the referential language seen in training. The former two are known as *schema encoding* whereas the latter challenge is known as *schema linking* – aligning entity references in the question to the intended schema columns or tables.

While the question of *schema encoding* has been studied in recent literature (Bogin et al., 2019a), *schema linking* has been relatively under-explored. Consider the example in Figure 3.2. It illustrates the challenge of ambiguity in linking: while *"model"* in the question refers to `car_names.model` rather than `model_list.model`, *"cars"* actually refers to both `cars_data` and `car_names` (but not `car_makers`) for the purpose of table joining. To resolve the column/table references properly, the semantic parser must take into account both the known schema relations (e.g., foreign keys) and the question context.

Prior work (Bogin et al., 2019a) addressed the schema representation problem by encoding the directed graph of foreign key relations in the schema with a graph neural network (GNN). While effective, this approach has two important shortcomings. Firstly, it does not contextualize schema encoding with the question, thus making reasoning about schema linking difficult after both the column representations and question word representations are built. Secondly, it limits information propagation during schema encoding to the predefined graph of foreign key relations. The advent of self-attentional mechanisms in NLP (Vaswani et al., 2017) shows that global reasoning is crucial to effective representations of relational structures. However, we would like any global reasoning to still take into account the aforementioned schema relations.

In this chapter, we present a unified framework, called RAT-SQL,[3] for encoding relational structure in a database schema and the given question. It uses *relation-aware self-attention* to combine global reasoning over the schema entities and question words with structured reasoning over predefined schema relations. We then apply RAT-SQL to the problems of schema encoding and schema linking. As a result, we obtain 57.2% exact match accuracy on the challenging Spider (Yu et al., 2018c) benchmark. At the time of releasing the work, this result was the state of the art among models

[3] **R**elation-**A**ware **T**ransformer.

unaugmented with pretrained BERT (Devlin et al., 2019) embeddings – and further reaches the overall state of the art (65.6%) when RAT-SQL is augmented with BERT. In addition, we experimentally demonstrate that RAT-SQL enables the model to build more accurate internal representations of the question's true alignment with schema columns and tables.

Optimization Mismatch To what extent is RAT-SQL close to achieving ideal domain generalization? Or more directly is it good enough? We answer this question by inspecting the gap between in-domain and out-of-domain performance on the Spider dataset. The in-domain performance is obtained by a new split that lets RAT-SQL observe some training examples of test domains, and it serves as an upper bound of the out-of-domain performance. Thus, the gap reveals the extent to which domain generalization is achieved for a parser. Surprisingly, we still see a gap of more than 25% in accuracy using RAT-SQL, despite it being a state-of-the-art cross-domain parser.

As an orthogonal direction to improving the architecture of semantic parsers, we focus on another important, yet under-explored aspect of building semantic parsers – training strategy. Conventional supervised learning simply assumes that source- and target-domain data originate from the same distribution, and as a result struggle to capture this notion of domain generalization for cross-domain semantic parsing. We draw inspiration from meta-learning (Finn et al., 2017; Li et al., 2018a) and use an objective that optimizes for domain generalization. That is, we simulate a set of tasks, where each task is a cross-domain semantic parsing task with its own source and target domains. By optimizing towards better target-domain performance on each task, we encourage a parser to extrapolate from source-domain data and achieve better domain generalization.

During training, to simulate the scenario where a parser needs to process questions to a new database at test time, we synthesize a set of virtual cross-domain parsing tasks by sampling disjoint source and target domains for each task from the training domains. The desideratum is that gradient steps computed towards better source-domain performance would also be beneficial to target-domain performance. One can think of the objective as consisting of both the loss on the source domain (as in standard supervised learning) and a regularizer, equal to the dot product between gradients computed on source- and target-domain data. Maximizing this regularizer intuitively encourages gradient updates on source and target domains to agree with each other, which presumably leads to gradient updates that are generally helpful for most domains

and alleviate overfitting to source or target domains. The objective is adapted from Li et al. (2018a) who adapt the Model-Agnostic Meta-Learning (MAML; Finn et al. 2017) algorithm for domain generalization in computer vision. In this chapter, we study the effectiveness of this objective in the context of semantic parsing. This objective is model-agnostic, simple to incorporate, and does not require any changes in parsing models. Moreover, it does not introduce new parameters for meta-learning.

Contributions We focus on transferability of semantic parsing, and aim to obtain a semantic parser that can be applied to any new domain. To this end, we promote cross-domain generalization of semantic parsing via two kinds of complementry methods listed below.

- *Model with structural inductive biases*: we propose RAT-SQL, a unified framework, based on the relation-aware self-attention mechanism, to handle the process of environment conditioning, comprising schema encoding and schema linking for cross-domain text-to-SQL parsing.
- *Training objectives*: in addition to the new semantic parser, we present a meta-learning objective (DG-MAML) that alleviates the optimization mismatch, and directly optimizes for domain generalization.

Empirical experiments on standard benchmarks, such as Spider, verify that our methodologies, which result to a new model architecture coupled with a new training strategy, lead to significantly better domain generalization.

Section Structure In Section 3.1 and 3.2, we first present our new semantic parser RAT-SQL. Then in Section 3.3 and 3.4, we show how RAT-SQL can be further augmented with our new training algorithm for better domain generalization.

3.1 Relation-Aware Semantic Parsing

We now describe RAT-SQL, a new semantic parser to handle conditioning on relational databases in text-to-SQL parsing. First, we explain relation-aware self-attention, which is the basic building block of RAT-SQL. Prior to presenting the components of RAT-SQL, we first formally define the text-to-SQL semantic parsing problem in detail. Finally, we present our main contribution – relation-aware input encoding module of RAT-SQL, along with a decoder module based on existing work.

Type of x	Type of y	Edge label	Description
Column	Column	SAME-TABLE	x and y belong to the same table.
		FOREIGN-KEY-COL-F	x is a foreign key for y.
		FOREIGN-KEY-COL-R	y is a foreign key for x.
Column	Table	PRIMARY-KEY-F	x is the primary key of y.
		BELONGS-TO-F	x is a column of y (but not the primary key).
Table	Column	PRIMARY-KEY-R	y is the primary key of x.
		BELONGS-TO-R	y is a column of x (but not the primary key).
Table	Table	FOREIGN-KEY-TAB-F	Table x has a foreign key column in y.
		FOREIGN-KEY-TAB-R	Same as above, but x and y are reversed.
		FOREIGN-KEY-TAB-B	x and y have foreign keys in both directions.

Table 3.1: Description of edge types present in the directed graph $\mathcal{G}$ created to represent the schema. An edge exists from source node $x \in \mathcal{S}$ to target node $y \in \mathcal{S}$ if the pair fulfills one of the descriptions listed in the table, with the corresponding label. Otherwise, no edge exists from x to y.

3.1.1 Preliminary: Relation-Aware Self-Attention

First, we introduce *relation-aware self-attention*, a model for embedding semi-structured input sequences in a way that jointly encodes pre-existing relational structure in the input as well as induces "soft" relations between sequence elements in the same embedding. Our solutions to schema embedding and linking naturally arise as features implemented in this framework.

Consider a set of inputs $X = \{x_i\}_{i=1}^{n}$ where $x_i \in \mathbb{R}^{d_x}$. In general, we consider it an unordered set, although x_i may be imbued with positional embeddings to add explicit ordering information. A *self-attention* encoder, or *Transformer*, introduced by Vaswani et al. (2017), is a stack of *self-attention layers* where each layer (consisting of H *heads*)

transforms each x_i into $y_i \in \mathbb{R}^{d_x}$ as follows:

$$
\begin{aligned}
e_{ij}^{(h)} &= \frac{x_i W_Q^{(h)} (x_j W_K^{(h)})^\top}{\sqrt{d_z/H}}; \quad \alpha_{ij}^{(h)} = \text{softmax}_j\{e_{ij}^{(h)}\} \\
z_i^{(h)} &= \sum_{j=1}^{n} \alpha_{ij}^{(h)} (x_j W_V^{(h)}); \quad z_i = \text{Concat}\left(z_i^{(1)}, \cdots, z_i^{(H)}\right) \\
\tilde{y}_i &= \text{LayerNorm}(x_i + z_i) \\
y_i &= \text{LayerNorm}(\tilde{y}_i + \text{FC}(\text{ReLU}(\text{FC}(\tilde{y}_i))))
\end{aligned}
\tag{3.1}
$$

where FC is a fully-connected layer, LayerNorm is *layer normalization* (Ba et al., 2016), $1 \le h \le H$, and $W_Q^{(h)}, W_K^{(h)}, W_V^{(h)} \in \mathbb{R}^{d_x \times (d_x/H)}$.

One interpretation of the embeddings computed by a Transformer is that each head of each layer computes a *learned relation* between all the input elements x_i, and the strength of this relation is encoded in the attention weights $\alpha_{ij}^{(h)}$. However, in many applications (including text-to-SQL parsing) we are aware of some pre-existing relational features between the inputs, and would like to bias our encoder model toward them. This is straightforward for non-relational features (represented directly in each x_i). We could limit the attention computation only to the "hard" edges where the preexisting relations are known to hold. This would make the model similar to a *graph attention network* (Veličković et al., 2018), and would also impede the Transformer's ability to learn *new* relations. Instead, RAT provides a way to communicate known relations to the encoder by adding their representations to the attention mechanism.

Shaw et al. (2018) describe a way to represent *relative position information* in a self-attention layer by changing Equation (3.1) as follows:

$$
\begin{aligned}
e_{ij}^{(h)} &= \frac{x_i W_Q^{(h)} (x_j W_K^{(h)} + r_{ij}^K)^\top}{\sqrt{d_z/H}} \\
z_i^{(h)} &= \sum_{j=1}^{n} \alpha_{ij}^{(h)} (x_j W_V^{(h)} + r_{ij}^V).
\end{aligned}
\tag{3.2}
$$

Here the r_{ij} terms encode the known relationship between the two elements x_i and x_j in the input. While Shaw et al. used it exclusively for relative position representation, we show how to use the same framework to effectively bias the Transformer toward arbitrary relational information.

Consider R relational features, each a binary relation $\mathcal{R}^{(s)} \subseteq X \times X$ $(1 \le s \le R)$. The RAT framework represents all the pre-existing features for each edge (i, j) as $r_{ij}^K = r_{ij}^V = \text{Concat}\left(\rho_{ij}^{(1)}, \ldots, \rho_{ij}^{(R)}\right)$ where each $\rho_{ij}^{(s)}$ is either a *learned embedding* for the relation $\mathcal{R}^{(s)}$ if the relation holds for the corresponding edge (i.e., if $(i, j) \in \mathcal{R}^{(s)}$), or

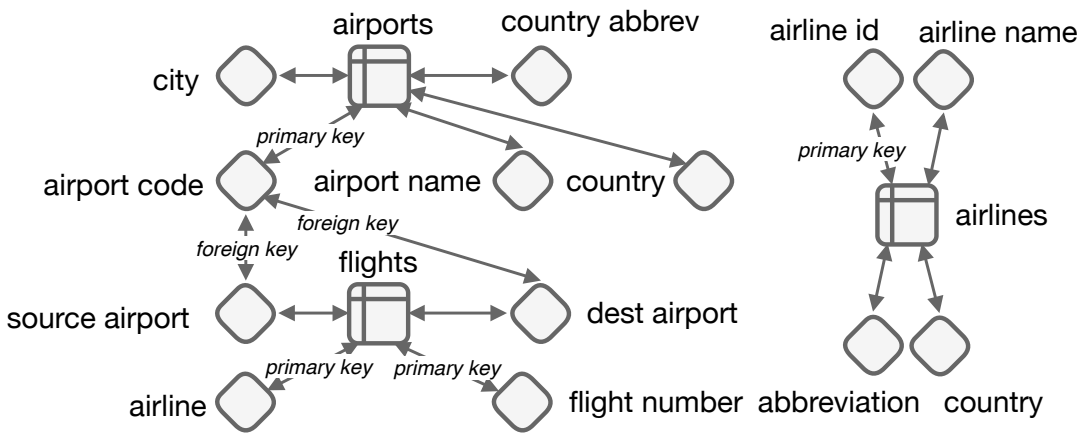


Figure 3.3: An illustration of an example schema as a graph $\mathcal{G}$. We do not depict all the edges and label types of Table 3.1 to reduce clutter. $\diamondsuit$ and $\square$ denote a column and a table, respectively.

a zero vector of appropriate size. In the following section, we will describe the set of relations our RAT-SQL model uses to encode a given database schema.

3.1.2 Problem Definition of Text-to-SQL Parsing

Given a natural language question Q and a schema $\mathcal{S} = \langle \mathcal{C}, \mathcal{T} \rangle$ for a relational database, our goal is to generate the corresponding SQL program P. Here question $Q = q_1 \dots q_{|Q|}$ is a sequence of words, and the schema consists of columns $\mathcal{C} = \{c_1, \dots, c_{|\mathcal{C}|}\}$ and tables $\mathcal{T} = \{t_1, \dots, t_{|\mathcal{T}|}\}$. Each column name c_i contains words $c_{i,1}, \dots, c_{i,|c_i|}$ and each table name t_i contains words $t_{i,1}, \dots, t_{i,|t_i|}$. The desired program P is represented as an *abstract syntax tree* T in the context-free grammar of SQL.

Some columns in the schema are *primary keys*, used for uniquely indexing the corresponding table, and some are *foreign keys*, used to reference a primary key column in a different table. In addition, each column has a *type* $\tau \in \{\texttt{number}, \texttt{text}\}$.

Formally, we represent the database schema as a directed graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$. Its nodes $\mathcal{V} = \mathcal{C} \cup \mathcal{T}$ are the columns and tables of the schema, each labeled with the words in its name (for columns, we prepend their type τ to the label). Its edges $\mathcal{E}$ are defined by the pre-existing database relations, described in Table 3.1. Figure 3.3 provides an example graph (with a subset of actual edges and labels).

While $\mathcal{G}$ holds all the known information about the schema, it is insufficient for appropriately encoding a previously unseen schema *in the context of the question* Q. We would like our representations of the schema $\mathcal{S}$ and the question Q to be

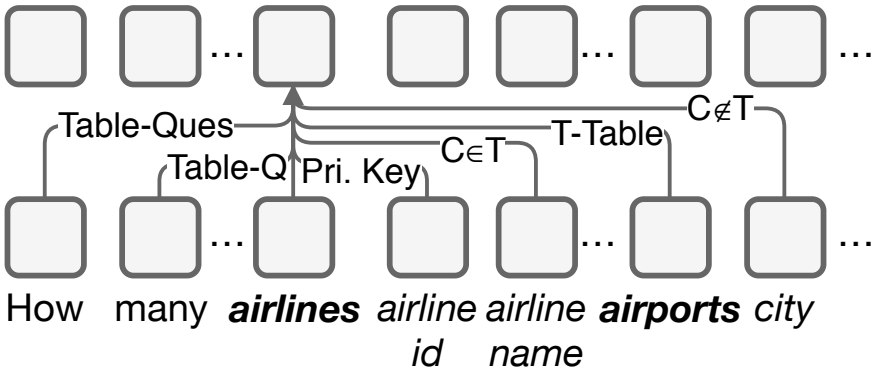


Figure 3.4: One RAT layer in the schema encoder.

joint, in particular for modeling the alignment between them. Thus, we also define the *question-contextualized schema graph* $\mathcal{G}_Q = \langle \mathcal{V}_Q, \mathcal{E}_Q \rangle$ where $\mathcal{V}_Q = \mathcal{V} \cup Q = \mathcal{C} \cup \mathcal{T} \cup Q$ includes nodes for the question words (each labeled with a corresponding word), and $\mathcal{E}_Q = \mathcal{E} \cup \mathcal{E}_{Q \leftrightarrow \mathcal{S}}$ are the schema edges $\mathcal{E}$ extended with additional special relations between the question words and schema members, detailed in the rest of this section.

For modeling text-to-SQL generation, we adopt the *encoder-decoder framework*. Given the input as a graph $\mathcal{G}_Q$, the *encoder* f_{enc} embeds it into joint representations c_i, t_i, q_i for each column $c_i \in \mathcal{C}$, table $t_i \in \mathcal{T}$, and question word $q \in Q$ respectively. The decoder f_{dec} then uses them to compute a distribution $\Pr(P \mid \mathcal{G}_Q)$ over the SQL programs.

3.1.3 Relation-Aware Input Encoding

Following the state-of-the-art NLP literature, our encoder first obtains the *initial* representations c_i^{init}, t_i^{init} for every node of $\mathcal{G}$ by **(a)** retrieving a pre-trained Glove embedding (Pennington et al., 2014) for each word, and **(b)** processing the embeddings in each multi-word label with a bidirectional LSTM (BiLSTM) (Hochreiter and Schmidhuber, 1997). It also runs a separate BiLSTM over the question Q to obtain initial word representations q_i^{init}.

The initial representations c_i^{init}, t_i^{init}, and q_i^{init} are independent of each other and devoid of any relational information known to hold in $\mathcal{E}_Q$. To produce joint representations for the entire input graph $\mathcal{G}_Q$, we use the relation-aware self-attention mechanism (Section 3.1.1). Its input X is the set of all the node representations in $\mathcal{G}_Q$:

$$X = (c_1^{\text{init}}, \cdots, c_{|\mathcal{C}|}^{\text{init}}, t_1^{\text{init}}, \cdots, t_{|\mathcal{T}|}^{\text{init}}, q_1^{\text{init}}, \cdots, q_{|Q|}^{\text{init}}).$$

The encoder f_{enc} applies a stack of N relation-aware self-attention layers to X, with

separate weight matrices in each layer. The final representations c_i, t_i, q_i produced by the N^{th} layer constitute the output of the whole encoder.

Alternatively, we also consider pre-trained BERT (Devlin et al., 2019) embeddings to obtain the initial representations. Following Huang et al. (2019); Zhang et al. (2019), we feed X to the BERT and use the last hidden states as the initial representations before proceeding with the RAT layers.[4]

Importantly, as detailed in Section 3.1.1, every RAT layer uses self-attention between *all elements of the input graph* $\mathcal{G}_Q$ to compute new contextual representations of question words and schema members. However, this self-attention is *biased* toward some pre-defined relations using the edge vectors r^K_{ij}, r^V_{ij} in each layer. We define the set of used relation types in a way that directly addresses the challenges of schema embedding and linking. Occurrences of these relations between the question and the schema constitute the edges $\mathcal{E}_{Q\leftrightarrow\mathcal{S}}$. Most of these relation types address schema linking (Section 3.1.4); we also add some auxiliary edges to aid schema encoding.

3.1.4 Schema Linking

Schema linking relations in $\mathcal{E}_{Q\leftrightarrow\mathcal{S}}$ aid the model with aligning column/table references in the question to the corresponding schema columns/tables. This alignment is implicitly defined by two kinds of information in the input: *matching names* and *matching values*, which we detail in order below.

Name-Based Linking Name-based linking refers to identifying *exact* or *partial* occurrences of the column/table names in the question, such as the occurrences of *"cylinders"* and *"cars"* in the question in Figure 3.2. Textual matches are the most explicit evidence of question-schema alignment and as such, one might expect them to be directly beneficial to the encoder. However, in all our experiments the representations produced by vanilla self-attention were insensitive to textual matches even though their initial representations were identical. Brunner et al. (2020) suggest that representations produced by Transformers mix the information from different positions and cease to be directly interpretable after 2+ layers, which might explain our observations. Thus, to remedy this phenomenon, we explicitly encode name-based linking using RAT relations.

Specifically, for all n-grams of length 1 to 5 in the question, we determine (1) whether it exactly matches the name of a column/table (*exact match*); or (2) whether

[4] In this case, the initial representations c_i^{init}, t_i^{init}, q_i^{init} are not strictly independent although still yet uninfluenced by $\mathcal{E}$.

the n-gram is a subsequence of the name of a column/table (*partial match*).[5] Then, for every (i, j) where $x_i \in Q$, $x_j \in \mathcal{S}$ (or vice versa), we set $r_{ij} \in \mathcal{E}_{Q \leftrightarrow \mathcal{S}}$ to QUESTION-COLUMN-M, QUESTION-TABLE-M, COLUMN-QUESTION-M or TABLE-QUESTION-M depending on the type of x_i and x_j. Here M is one of EXACTMATCH, PARTIALMATCH, or NOMATCH.

Value-Based Linking Question-schema alignment also occurs when the question mentions any *values* that occur in the database and consequently participate in the desired SQL, such as *"4"* in Figure 3.2. While this example makes the alignment explicit by mentioning the column name *"cylinders"*, many real-world questions do not. Thus, linking a value to the corresponding column requires background knowledge.

The database itself is the most comprehensive and readily available source of knowledge about possible values, but also the most challenging to process in an end-to-end model because of the privacy and speed impact. However, the RAT framework allows us to outsource this processing to the database engine to *augment* $\mathcal{G}_Q$ with potential value-based linking without exposing the model itself to the data. Specifically, we add a new COLUMN-VALUE relation between any word q_i and column name c_j s.t. q_i occurs as a value (or a full word within a value) of c_j. This simple approach drastically improves the performance of RAT-SQL (see Section 3.2). It also directly addresses the aforementioned DB challenges: **(a)** the model is never exposed to database content that does not occur in the question, **(b)** word matches are retrieved quickly via DB indices and textual search.

Memory-Schema Alignment Matrix Our intuition suggests that the columns and tables which occur in the SQL P will generally have a corresponding reference in the natural language question. To capture this intuition in the model, we apply relation-aware attention as a pointer mechanism between every memory element in y and all the columns/tables to compute explicit *alignment matrices* $L^{\text{col}} \in \mathbb{R}^{|y| \times |C|}$ and $L^{\text{tab}} \in$

[5] This procedure matches that of Guo et al. (2019b), but we use the matching information differently in RAT.

$\mathbb{R}^{|y|\times|\mathcal{T}|}$:

$$\begin{aligned}
\tilde{L}^{\text{col}}_{i,j} &= \frac{y_i W^{\text{col}}_Q (c^{\text{final}}_j W^{\text{col}}_K + r^K_{ij})^\top}{\sqrt{d_x}} \\
\tilde{L}^{\text{tab}}_{i,j} &= \frac{y_i W^{\text{tab}}_Q (t^{\text{final}}_j W^{\text{tab}}_K + r^K_{ij})^\top}{\sqrt{d_x}} \\
L^{\text{col}}_{i,j} &= \text{softmax}_j\{\tilde{L}^{\text{col}}_{i,j}\} \qquad L^{\text{tab}}_{i,j} = \text{softmax}_j\{\tilde{L}^{\text{tab}}_{i,j}\}
\end{aligned} \tag{3.3}$$

Intuitively, the alignment matrices in Eq. (3.3) should resemble the real discrete alignments and therefore should respect certain constraints like sparsity. When the encoder is sufficiently parameterized, sparsity tends to arise with learning, but we can also encourage it with an explicit objective.

3.1.5 Decoder

The decoder f_{dec} of RAT-SQL follows the tree-structured architecture of Yin and Neubig (2017). It generates the SQL P as an abstract syntax tree in depth-first traversal order, by using an LSTM to output a sequence of *decoder actions* that either (i) expand the last generated node into a grammar rule, called ApplyRule; or when completing a leaf node, (ii) choose a column/table from the schema, called SelectColumn and SelectTable.

Formally, $\Pr(P \mid \mathcal{Y}) = \prod_t \Pr(a_t \mid a_{<t}, \mathcal{Y})$ where $\mathcal{Y} = f_{\text{enc}}(\mathcal{G}_Q)$ is the final encoding of the question and schema, and $a_{<t}$ are all the previous actions. In a tree-structured decoder, the LSTM state is updated as $m_t, h_t = f_{\text{LSTM}}\left([a_{t-1} \,\|\, z_t \,\|\, h_{p_t} \,\|\, a_{p_t} \,\|\, n_{f_t}],\ m_{t-1}, h_{t-1}\right)$ where m_t is the LSTM cell state, h_t is the LSTM output at step t, a_{t-1} is the embedding of the previous action, p_t is the step corresponding to expanding the parent AST node of the current node, and n_{f_t} is the embedding of the current node type. Finally, z_t is the context representation, computed using multi-head attention (with 8 heads) on h_{t-1} over $\mathcal{Y}$.

For ApplyRule$[R]$, we compute $\Pr(a_t = \text{ApplyRule}[R] \mid a_{<t}, y) = \text{softmax}_R(g(h_t))$ where $g(\cdot)$ is a 2-layer MLP with a tanh non-linearity. For SelectColumn, we compute

$$\tilde{\lambda}_i = \frac{h_t W^{\text{sc}}_Q (y_i W^{\text{sc}}_K)^T}{\sqrt{d_x}} \qquad \lambda_i = \text{softmax}_i\{\tilde{\lambda}_i\}$$

$$\Pr(a_t = \text{SelectColumn}[i] \mid a_{<t}, y) = \sum_{j=1}^{|y|} \lambda_j L^{\text{col}}_{j,i}$$

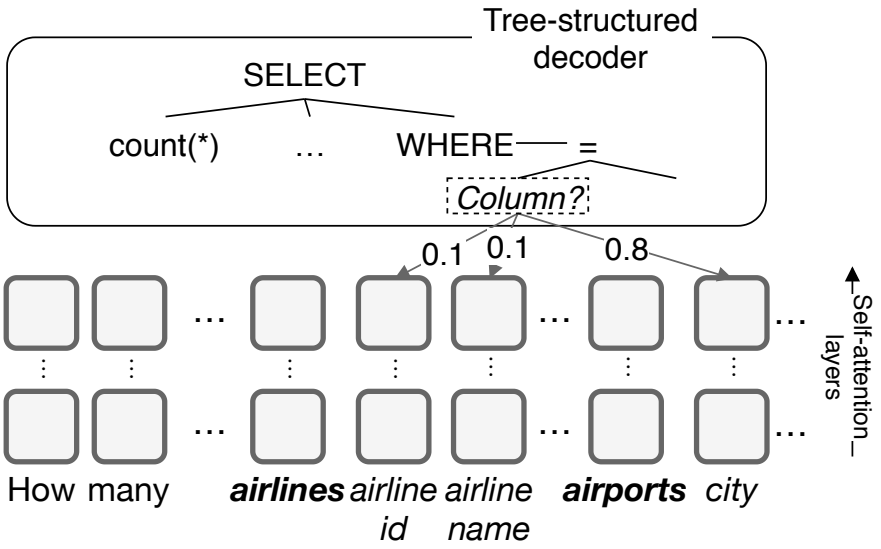


Figure 3.5: Choosing a column in a tree decoder.

and similarly for SELECTTABLE. Figure 3.5 illustrates SELECTCOLUMN of the tree decoder.

3.2 Experiments

We implemented RAT-SQL in PyTorch (Paszke et al., 2017). During preprocessing, the input of questions, column names and table names are tokenized and lemmatized with the StandfordNLP toolkit (Manning et al., 2014). Within the encoder, we use GloVe (Pennington et al., 2014) word embeddings, held fixed in training except for the 50 most common words in the training set. For RAT-SQL BERT, we use the WordPiece tokenization. All word embeddings have dimension 300. The bidirectional LSTMs have hidden size 128 per direction, and use the recurrent dropout method of Gal and Ghahramani (2016) with rate 0.2. We stack 8 relation-aware self-attention layers on top of the bidirectional LSTMs. Within them, we set $d_x = d_z = 256$, $H = 8$, and use dropout with rate 0.1. The position-wise feed-forward network has inner layer dimension 1024. Inside the decoder, we use rule embeddings of size 128, node type embeddings of size 64, and a hidden size of 512 inside the LSTM with dropout of 0.21.

We used the Adam optimizer (Kingma and Ba, 2015) with the default hyperparameters. During the first $warmup\_steps = max\_steps/20$ steps of training, the learning rate linearly increases from 0 to 7.4×10^{-4}. Afterwards, it is annealed to 0 with $7.4 \times 10^{-4}(1 - \frac{step - warmup\_steps}{max\_steps - warmup\_steps})^{0.5}$. We use a batch size of 20 and train for up to 40,000 steps. For RAT-SQL + BERT, we use a separate learning rate of 3×10^{-6} to

Model	Dev	Test
IRNet (Guo et al., 2019b)	53.2	46.7
Global-GNN (Bogin et al., 2019b)	52.7	47.4
IRNet V2 (Guo et al., 2019b)	55.4	48.5
RAT-SQL (ours)	**62.7**	**57.2**
With BERT:		
EditSQL + BERT (Zhang et al., 2019)	57.6	53.4
GNN + Bertrand-DR (Kelkar et al., 2020)	57.9	54.6
IRNet V2 + BERT (Guo et al., 2019b)	63.9	55.0
RYANSQL V2 + BERT (Choi et al., 2020)	**70.6**	60.6
RAT-SQL + BERT (ours)	69.7	**65.6**

Table 3.2: Accuracy on the Spider development and test sets, compared to the other approaches at the top of the dataset leaderboard as of May 1st, 2020. The test set results were scored using the Spider evaluation server.

fine-tune BERT, a batch size of 24 and train for up to 90,000 steps.

Hyperparameter Search We tuned the batch size (20, 50, 80), number of RAT layers (4, 6, 8), dropout (uniformly sampled from $[0.1, 0.3]$), hidden size of decoder RNN (256, 512), max learning rate (log-uniformly sampled from $[5 \times 10^{-4}, 2 \times 10^{-3}]$). We randomly sampled 100 configurations and optimized on the dev set. RAT-SQL + BERT reuses most hyperparameters of RAT-SQL, only tuning the BERT learning rate (1×10^{-6}, 3×10^{-6}, 5×10^{-6}), number of RAT layers (6, 8, 10), number of training steps (4×10^4, 6×10^4, 9×10^4).

Datasets We use the Spider dataset (Yu et al., 2018c) for most of our experiments, and also conduct preliminary experiments on WikiSQL (Zhong et al., 2017) to confirm generalization to other datasets. As described by Yu et al., Spider contains 8,659 examples (questions and SQL queries, with the accompanying schemas), including 1,659 examples lifted from the Restaurants (Popescu et al., 2003; Tang and Mooney, 2000), GeoQuery (Zelle and Mooney, 1996), Scholar (Iyer et al., 2017), Academic (Li and Jagadish, 2014), Yelp and IMDB (Yaghmazadeh et al., 2017) datasets. We do **not** use the data augmentation scheme of Yu et al. (2018b).

As Yu et al. (2018c) make the test set accessible only through an evaluation server,

we perform most evaluations (other than the final accuracy measurement) using the development set. It contains examples with databases and schemas distinct from those in the training set. For Spider, we report results using exact set match accuracy Yu et al. (2018c) [6], as well as divided by difficulty levels. This metric considers SQL query as a set of components and measures the extent to which the set of a predicted query matches the set of a gold query in terms of F1. For WikiSQL, we report exact match (i.e., program) accuracy and execution accuracy.

3.2.1 Spider Results

In Table 3.2 we show accuracy on the (hidden) Spider test set for RAT-SQL and compare to all other approaches at or near state-of-the-art (according to the official leaderboard). RAT-SQL outperforms all other methods that are not augmented with BERT embeddings by a large margin of 8.7%. Surprisingly, it even beats other BERT-augmented models. When RAT-SQL is further augmented with BERT, it achieves the new state-of-the-art performance. Compared with other BERT-argumented models, our RAT-SQL + BERT has smaller generalization gap between development and test set.

We also provide a breakdown of the accuracy by difficulty in Table 3.3. As expected, performance drops with increasing difficulty. The overall generalization gap between development and test of RAT-SQL was strongly affected by the significant drop in accuracy (9%) on the extra hard questions. When RAT-SQL is augmented with BERT, the generalization gaps of most difficulties are reduced.

Ablation Study Table 3.4 shows an ablation study over different RAT-based relations. The ablations are run on RAT-SQL without value-based linking to avoid interference with information from the database. Schema linking and graph relations make statistically significant improvements (p¡0.001). The full model accuracy here slightly differs from Table 3.2 because the latter shows the best model from a hyper-parameter sweep (used for test evaluation) and the former gives the mean over five runs where we only change the random seeds.

Split	Easy	Medium	Hard	Extra Hard	All
RAT-SQL					
Dev	80.4	63.9	55.7	40.6	62.7
Test	74.8	60.7	53.6	31.5	57.2
RAT-SQL + BERT					
Dev	86.4	73.6	62.1	42.9	69.7
Test	83.0	71.3	58.3	38.4	65.6

Table 3.3: Accuracy on the Spider development and test sets, by difficulty as defined by Yu et al. (2018c).

Model	Accuracy (%)
RAT-SQL + value-based linking	**60.54 ± 0.80**
RAT-SQL	55.13 ± 0.84
w/o schema linking relations	40.37 ± 2.32
w/o schema graph relations	35.59 ± 0.85

Table 3.4: Accuracy (and ±95% confidence interval) of RAT-SQL ablations on the dev set.

	Dev		Test	
Model	**LF Acc%**	**Ex. Acc%**	**LF Acc%**	**Ex. Acc%**
IncSQL (Shi et al., 2018)	49.9	84.0	49.9	83.7
MQAN (McCann et al., 2018)	76.1	82.0	75.4	81.4
RAT-SQL (ours)	73.6	79.5	73.3	78.8
Coarse2Fine (Dong and Lapata, 2018)	72.5	79.0	71.7	78.5
PT-MAML (Huang et al., 2018)	63.1	68.3	62.8	68.0

Table 3.5: RAT-SQL accuracy on WikiSQL, trained without BERT augmentation or execution-guided decoding (EG). Compared to other approaches without EG. “LF Acc” = Logical Form Accuracy; “Ex. Acc” = Execution Accuracy.

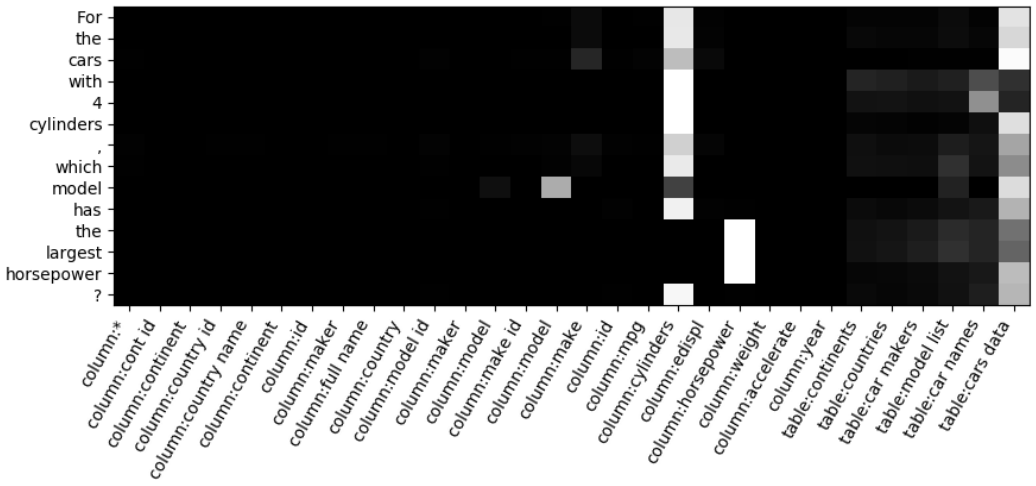


Figure 3.6: Alignment between the question *"For the cars with 4 cylinders, which model has the largest horsepower"* and the database `car_1` schema (columns and tables) depicted in Figure 3.2.

3.2.2 WikiSQL Results

We also conducted preliminary experiments on WikiSQL (Zhong et al., 2017) to test generalization of RAT-SQL to new datasets. Although WikiSQL lacks multi-table schemas (and thus, schema encoding is not as challenging), it still presents the challenges of schema linking and generalization to new schemas. For simplicity of experiments, we did not implement either BERT augmentation or execution-guided decoding (EG) (Wang et al., 2018), both of which are common in state-of-the-art WikiSQL models. We thus only compare to the models that also lack these two enhancements.

While not reaching state of the art, RAT-SQL still achieves competitive performance on WikiSQL as shown in Table 3.5. Most of the gap between its accuracy and state of the art is due to the simplified implementation of *value decoding*, which is required for WikiSQL evaluation but not in Spider. Our value decoding for these experiments is a simple token-based pointer mechanism, which often fails to retrieve multi-token value constants accurately.

[6] At the time of this work, exact set match accuracy is used as the primary metric mainly because early systems are too weak to be measured by program or execution accuracy. For direct comparison with other systems, we use exact set match accuracy. In Section 3.4, we report execution accuracy in addition to exact set match accuracy.

3.2.3 Discussion

Alignment Recall from Section 3.1 that we explicitly model the alignment matrix between question words and table columns, used during decoding for column and table selection. The alignment matrix provides a mechanism for the model to align words to columns. An accurate alignment representation has other benefits such as identifying question words to copy to emit a constant value in SQL.

In Figure 3.6 we show the alignment generated by our model on the example from Figure 3.2.[7] For the three words that reference columns (*"cylinders"*, *"model"*, *"horsepower"*), the alignment matrix correctly identifies their corresponding columns. The alignments of other words are strongly affected by these three keywords, resulting in a sparse span-to-column like alignment, e.g., *"largest horsepower"* to `horsepower`. The tables `cars_data` and `cars_names` are implicitly mentioned by the word *"cars"*. The alignment matrix successfully infers to use these two tables instead of `car_makers` using the evidence that they contain the three mentioned columns.

The Need for Schema Linking One natural question is how often does the decoder fail to select the correct column, even with the schema encoding and linking improvements we have made. To answer this, we conducted an oracle experiment (see Table 3.6). For "oracle sketch", at every grammar nonterminal the decoder is forced to choose the correct production so the final SQL sketch exactly matches that of the ground truth. The rest of the decoding proceeds conditioned on that choice. Likewise, "oracle columns" forces the decoder to emit the correct column/table at terminal nodes.

With both oracles, we see an accuracy of 99.4% which verifies that our grammar is sufficient to answer nearly every question in the data set. With just "oracle sketch", the accuracy is only 73.0%, which means 72.4% of the questions, that RAT-SQL gets wrong and could get right, have incorrect column or table selection. Similarly, with just "oracle columns", the accuracy is 69.8%, which means that 81.0% of the questions, that RAT-SQL gets wrong, have incorrect structure. In other words, most questions have both column and structure wrong, so there is room for improvement.

Error Analysis An analysis of mispredicted SQL queries in the Spider dev set showed three main causes of evaluation errors. **(I)** 18% of the mispredicted queries are in fact *equivalent* implementations of the NL intent with a different SQL syntax (e.g., `ORDER`

[7] The full alignment also maps from column and table names, but those end up simply aligning to themselves or the table they belong to, so we omit them for brevity.

Model	Acc.
RAT-SQL	62.7
RAT-SQL + Oracle columns	69.8
RAT-SQL + Oracle sketch	73.0
RAT-SQL + Oracle sketch + Oracle columns	99.4

Table 3.6: Accuracy (exact match %) on the development set given an oracle providing correct columns and tables ("Oracle columns") and/or the AST sketch structure ("Oracle sketch").

`BY` C `LIMIT 1` vs. `SELECT MIN(`C`)`). Measuring execution accuracy rather than exact match would detect them as valid. **(II)** 39% of errors involve a wrong, missing, or extraneous column in the `SELECT` clause. This is a limitation of our schema linking mechanism, which, while substantially improving column resolution, still struggles with some ambiguous references. Some of them are unavoidable as Spider questions do not always specify which columns should be returned by the desired SQL. Finally, **(III)** 29% of errors are missing a `WHERE` clause, which is a common error class in text-to-SQL models as reported by prior works. One common example is domain-specific phrasing such as *"older than 21"*, which requires background knowledge to map it to `age > 21` rather than `age < 21`. Such errors disappear after in-domain fine-tuning.

3.3 Meta-Learning for Domain Generalization

We now investigate another dimension of building cross-domain semantic parsers, which is how to effectively train them to handle domain generalization, rather than rely on standard maximum likelihood training.

Conventional Supervised Learning: a Single Task Assuming that question-SQL pairs from source domains and target domains are sampled i.i.d from the same distribution, the typical training objective of supervised learning, which is the one we used in the previous section, is to minimize the loss function of the negative log-likelihood of the gold SQL query:

$$\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(P|Q,\mathcal{D}) \tag{3.4}$$

where N is the size of mini-batch $\mathcal{B}$. Since a mini-batch is randomly sampled from all training source domains $\mathcal{D}_s$, it usually contains question-SQL pairs from a mixture of different domains.

Meta Learning: Distribution of Tasks Instead of treating semantic parsing as a conventional supervised learning problem, we take an alternative view based on meta-learning. Basically, we are interested in a learning algorithm that can benefit from a distribution of choices of source and target domains, denoted by $p(\tau)$, where τ refers to an instance of a zero-shot semantic parsing task in the sense that each has its own disjoint source and target domains.

In practice, we usually have a fixed set of training source domains $\mathcal{D}_s$. We construct a set of virtual tasks τ by randomly sampling disjoint source and target domains from the training domains. Intuitively, we assume that divergences between the test and training domains during the learning phase are representative of differences between training and actual test domains. This is still an assumption, but considerably weaker compared to the i.i.d. assumption used in conventional supervised learning. Next, we introduce the training algorithm called DG-MAML motivated by this assumption.

3.3.1 Learning to Generalize with DG-MAML

Having simulated source and target domains for each virtual task, we now need a training algorithm that encourages generalization to unseen target domains in each task. For this, we turn to optimization-based meta-learning algorithms Finn et al. (2017); Nichol et al. (2018); Li et al. (2018a) and apply DG-MAML (Domain Generalization with Model-Agnostic Meta-Learning), a variant of MAML (Finn et al., 2017) for this purpose. Intuitively, DG-MAML encourages the optimization in the source domain to have a positive effect on the target domain as well.

During each learning episode of DG-MAML, we randomly sample a task τ which has its own source domain $\mathcal{D}_s^\tau$ and target domain $\mathcal{D}_t^\tau$. For the sake of efficiency, we randomly sample mini-batch question-SQL pairs $\mathcal{B}_s$ and $\mathcal{B}_t$ from $\mathcal{D}_s^\tau$ and $\mathcal{D}_t^\tau$, respectively, for learning in each task. DG-MAML conducts optimization in two steps, namely *meta-train* and *meta-test*.

Meta-Train DG-MAML first optimizes parameters towards better performance in the virtual source domain $\mathcal{D}_s^\tau$ by taking one step of stochastic gradient descent (SGD) from

the loss under $\mathcal{B}_s$.

$$\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}_s}(\boldsymbol{\theta}) \tag{3.5}$$

where α is a scalar denoting the learning rate of meta-train. This step resembles conventional supervised learning where we use stochastic gradient descent to optimize the parameters.

Meta-Test We then evaluate the resulting parameters $\boldsymbol{\theta}'$ in the virtual target domain $\mathcal{D}_t$ by computing the loss under $\mathcal{B}_t$, which is denoted as $\mathcal{L}_{\mathcal{B}_t}(\boldsymbol{\theta}')$.

Our final objective for a task τ is to minimize the joint loss on $\mathcal{D}_s$ and $\mathcal{D}_t$:

$$\begin{aligned} \mathcal{L}_\tau(\boldsymbol{\theta}) &= \mathcal{L}_{\mathcal{B}_s}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{B}_t}(\boldsymbol{\theta}') \\ &= \mathcal{L}_{\mathcal{B}_s}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{B}_t}(\boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}_s}(\boldsymbol{\theta})) \end{aligned} \tag{3.6}$$

where we optimize towards the better source *and* target domain performance simultaneously. Intuitively, the objective requires that the gradient step conducted in the source domains in Equation (3.5) is beneficial to the performance of the target domain as well. In comparison, conventional supervised learning, whose objective would be equivalent to $\mathcal{L}_{\mathcal{B}_s}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{B}_t}(\boldsymbol{\theta})$, does not pose any constraint on the gradient updates. As we will elaborate shortly, DG-MAML can be viewed as a regularization of gradient updates in addition to the objective of conventional supervised learning.

We summarize our DG-MAML training process in Algorithm 1. Basically, it requires two steps of gradient update (Step 5 and Step 7). Note that $\boldsymbol{\theta}'$ is a function of $\boldsymbol{\theta}$ after the meta-train update. Hence, optimizing $\mathcal{L}_\tau(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ involves optimizing through the gradient update in Equation (3.5) as well. That is, when we update the parameters $\boldsymbol{\theta}$ in the final update of Step 7, the gradients need to back-propagate through the meta-train updates in Step 5. The update function in Step 7 could be based on any gradient descent algorithm. We choose the frequently-used optimizer Adam (Kingma and Ba, 2015). The process to compute the DG-MAML objective is also illustrated in Figure 3.7.

Note that DG-MAML is different from the original MAML (Finn et al., 2017) which is typically used in the context of few-shot learning. In our case, it encourages domain generalization during training, and does not require an adaptation phase.

3.3.2 Analysis of DG-MAML

To give an intuition of the objective in Equation (3.6), we follow previous work (Nichol et al., 2018; Li et al., 2018a) and use the first-order Taylor series expansion to approxi-

Algorithm 1 DG-MAML Training Algorithm

Require: Training databases $\mathcal{D}$

Require: Learning rate α

1: **for** step $\leftarrow 1$ **to** T **do**
2: Sample a task τ of $(\mathcal{D}_s^\tau, \mathcal{D}_t^\tau)$ from $\mathcal{D}$
3: Sample mini-batch $\mathcal{B}_s^\tau$ from $\mathcal{D}_s^\tau$
4: Sample mini-batch $\mathcal{B}_t^\tau$ from $\mathcal{D}_t^\tau$
5: Meta-train update: $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}_s^\tau}(\boldsymbol{\theta})$
6: Compute meta-test objective: $\mathcal{L}_\tau(\boldsymbol{\theta}) = \mathcal{L}_{\mathcal{B}_s}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{B}_t}(\boldsymbol{\theta}')$
7: Final Update: $\boldsymbol{\theta} \leftarrow \text{Update}(\boldsymbol{\theta}, \nabla_{\boldsymbol{\theta}} \mathcal{L}_\tau(\boldsymbol{\theta}))$
8: **end for**

mate it:

$$\begin{aligned}\mathcal{L}_\tau(\boldsymbol{\theta}) =& \mathcal{L}_{\mathcal{B}_s}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{B}_t}(\boldsymbol{\theta}') \\ =& \mathcal{L}_{\mathcal{B}_s}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{B}_t}(\boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}_s}(\boldsymbol{\theta})) \\ \approx& \mathcal{L}_{\mathcal{B}_s}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{B}_t}(\boldsymbol{\theta}) - \alpha(\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}_s}(\boldsymbol{\theta}) \cdot \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}_t}(\boldsymbol{\theta}))\end{aligned} \tag{3.7}$$

where in the last step we expand the function $\mathcal{L}_{\mathcal{B}_s}$ at $\boldsymbol{\theta}$. The approximated objective sheds light on what DG-MAML optimizes. In addition to minimizing the losses from both source and target domains, which are $\mathcal{L}_{\mathcal{B}_s}(\boldsymbol{\theta}) + \mathcal{L}_{\mathcal{B}_t}(\boldsymbol{\theta})$, DG-MAML tries to maximize $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}_s}(\boldsymbol{\theta}) \cdot \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}_t}(\boldsymbol{\theta})$, the dot product between the gradients of source and target domains. That is, it encourages gradients to generalize between source and target domain within each task τ.

3.3.3 First-Order Approximation

The final update in Step 7 of Algorithm 1 requires second-order derivatives, which may be problematic, inefficient or non-stable with certain classes of models (Mensch and Blondel, 2018). Hence, we propose an approximation that only requires computing first-order derivatives.

First, the gradient of the objective in Equation (3.6) can be computed as:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \mathcal{L}_\tau(\boldsymbol{\theta}) =& \nabla_{\boldsymbol{\theta}} \boldsymbol{\theta}' \nabla_{\boldsymbol{\theta}'} \mathcal{L}_{\mathcal{B}_t}(\boldsymbol{\theta}') + \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}_s}(\boldsymbol{\theta}) \\ =& \big(\boldsymbol{I} - \alpha \nabla^2_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}_s}(\boldsymbol{\theta})\big) \nabla_{\boldsymbol{\theta}'} \mathcal{L}_{\mathcal{B}_t}(\boldsymbol{\theta}') + \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}_s}(\boldsymbol{\theta})\end{aligned} \tag{3.8}$$

where $\boldsymbol{I}$ is an identity matrix and $\nabla^2_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}_s}(\boldsymbol{\theta})$ is the Hessian of $\mathcal{L}_{\mathcal{B}_s}$ at $\boldsymbol{\theta}$. We consider the alternative of ignoring this second-order term and simply assume that $\nabla_{\boldsymbol{\theta}} \boldsymbol{\theta}' = \boldsymbol{I}$. In

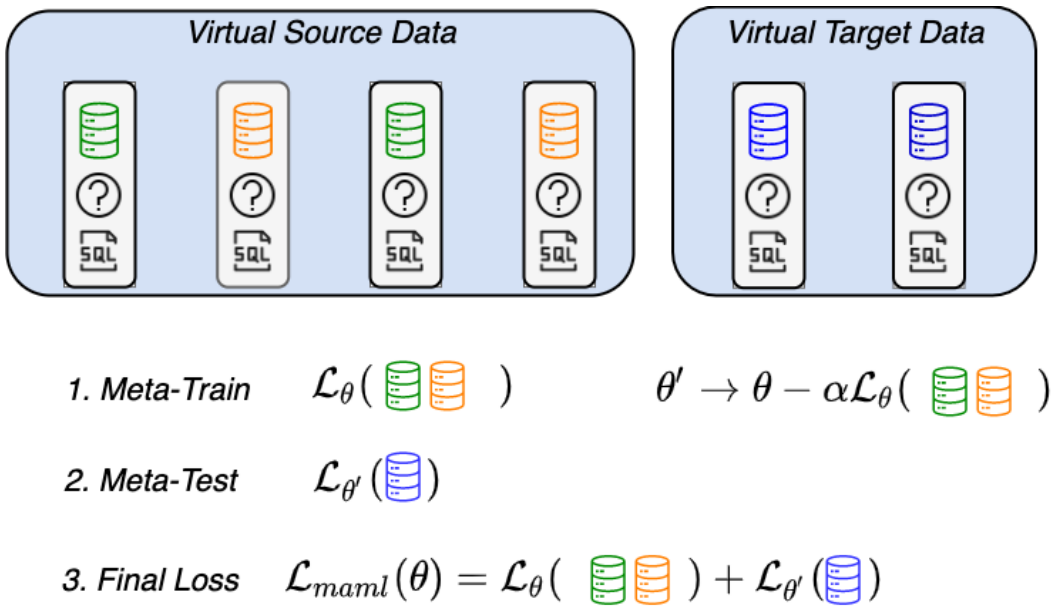


Figure 3.7: Three steps to compute the DG-MAML objective. In the first meta-train step, we perform one step gradient update based on the loss $\mathcal{L}_\theta$ from the virtual source databases shown in green and orange. In the second meta-test step, we evaluate the updated parameter θ' on the virtual target database shown in blue and get a new loss $\mathcal{L}_{\theta'}$. Finally, the DG-MAML objective $\mathcal{L}_{maml}$ is the sum of both objectives from the meta-train and meta-test steps. Loss on a database is computed over a batch of examples sampled from the database, and the grey block represents an example with a database (denoted with different colors), a natural language query (denoted as ? symbol), and a SQL query.

this variant, we simply combine gradients from source and target domains. But this objective can still be viewed as maximizing the dot product of gradients from source and target domain (for details see Nichol et al. (2018)).

The resulting first-order training objective, which we refer to as DG-FMAML, is inspired by Reptile, a first-order meta-learning algorithm (Nichol et al., 2018) for few-shot learning. A two-step Reptile would compute SGD on the same batch twice while DG-FMAML computes SGD on two different batches, $\mathcal{B}_s$ and $\mathcal{B}_t$, once. To put it differently, DG-FMAML tries to encourage *cross-domain* generalization while Reptile encourages *in-domain* generalization.

3.4 Experiments

Base Parsers In general, DG-MAML is model-agnostic implying that it can be coupled with any semantic parser to improve its domain generalization. In this chapter, our base parsers are based on RAT-SQL. We made few improvements based on the original RAT-SQL, such as adding a component for value prediction so that our base parsers can be evaluated by execution accuracy. We call this parser RAT-SQL V2, and we will compare it with the original one. By designing an in-domain benchmark, we also show that the out-of-domain improvement does not come at the cost of in-domain performance. We also present some analysis to show how DG-MAML affects domain generalization.

Datasets We evaluate DG-MAML on two text-to-SQL benchmarks, namely, (English) Spider (Yu et al., 2018c) and Chinese Spider (Min et al., 2019a). Chinese Spider is a Chinese version of Spider that translates all NL questions from English to Chinese and keeps the original English database. It introduces the additional challenge of encoding cross-lingual correspondences between Chinese and English. In both datasets, we report exact set match accuracy and execution accuracy [8].

Baselines Two kinds of features are widely used in recent semantic parsers to boost domain generalization: schema-linking features (as mentioned in Section 3.1) and pre-trained emebddings such as BERT. To show that our method can still achieve additional improvements, we compare with strong baselines that are integrated with schema-linking features and pre-trained embeddings. In the analysis (Section 3.4.4), we will also show the effect of our method when both features are absent in the base parsers.

We also compare with other leading text-to-SQL parsers such as IRNet (Guo et al., 2019b), BRIDGE (Lin et al., 2020) and RYANSQL (Choi et al., 2020). These parsers differ from RAT-SQL in architecture, but the comparison aims to show that our base parser RAT-SQL V2 is comparable, or better than these leading parsers, and DG-MAML can still have a significant effect on this strong base parser, as we will show soon.

Implementation and Hyperparameters For English questions and schemas, we use GloVe (Pennington et al., 2014) and BERT-base (Devlin et al., 2019) as the pre-trained

[8] In the leaderboard, execution accuracy is also called 'execution with values'.

embeddings for encoding. For Chinese questions, we use Tencent embeddings (Song et al., 2018) and Multilingual-BERT (Devlin et al., 2019). In all experiments, we use a batch size of $\mathcal{B}_s = \mathcal{B}_t = 12$ and train for up to 20,000 steps.

3.4.1 Main Results

Our main results on Spider and Chinese Spider are listed in Table 3.7 and 3.8, respectively.

Non-BERT Models DG-MAML boosts the performance of non-BERT base parsers on Spider and Chinese Spider by 2.1% and 4.5% respectively, showing its effectiveness in promoting domain generalization. In comparison, the performance margin for DG-MAML is more significant in the cross-lingual setting of Chinese Spider. This is presumably due to the fact that heuristic schema-linking features, which help promote domain generalization for Spider, are not applicable in Chinese Spider. We will present more analysis on this in Section 3.4.4.

BERT Models Most importantly, improvements on both datasets are not cancelled out when the base parsers are augmented with pre-trained representations. On Spider, the improvements brought by DG-MAML remain roughly the same when the base parser is integrated with BERT-base. As a result, our base parser augmented with BERT-base and DG-MAML achieves the best execution accuracy compared with previous models. On Chinese Spider, DG-MAML helps the base parser with multilingual BERT achieve a substantial improvement. Overall, DG-MAML consistently boosts the performance of the base parser, and is complementary to using pre-trained representations.

3.4.2 In-Domain vs. Out-of-Domain

To confirm that the base parser struggles when applied out-of-domain, we construct an in-domain setting and measure the gap in performance. This setting also helps us address a natural question: does using DG-MAML hurt in-domain performance? This would not have been surprising as the parser is explicitly optimized towards better performance on unseen target domains.

To answer these questions, we create a new split of Spider. Specifically, for each database from the training and development set of Spider, we include 80% of its question-SQL pairs in the new training set and assign the remaining 20% to the new

Model	Dev	Test
Set Match Accuracy		
SyntaxSQLNet (Yu et al., 2018b)	18.9	19.7
Global-GNN (Bogin et al., 2019b)	52.7	47.4
IRNet (Guo et al., 2019b)	55.4	48.5
RAT-SQL (Wang et al., 2020)	**62.7**	57.2
Our Models		
RAT-SQL V2	56.4	-
RAT-SQL V2 + DG-MAML	58.5	-
With BERT-base:		
SyntaxSQLNet + BERT-base (Guo et al., 2019b)	25.0	25.4
IRNet + BERT-base (Guo et al., 2019b)	61.9	54.7
BRIDGE + BERT-base (Lin et al., 2020)	65.5	58.2
RAT-SQL + BERT-base	66.0	-
Our Models		
RAT-SQL V2 + BERT-base	66.8	63.3
RAT-SQL V2 + BERT-base + DG-MAML	**68.9**	**65.2**
With BERT-large:		
RYANSQL + BERT-large (Choi et al., 2020)	**70.6**	60.6
RAT-SQL + BERT-large (Wang et al., 2020)	69.7	**65.6**
Execution Accuracy		
GAZP + Distil-BERT (Zhong et al., 2020)	59.2	53.5
BRIDGE + BERT-base (Lin et al., 2020)	65.3	59.9
Our Models		
RAT-SQL V2 + BERT-base	66.8	64.1
RAT-SQL V2 + BERT-base + DG-MAML	**69.3**	**66.1**

Table 3.7: Accuracy (%) on the development and test sets of Spider. The first half shows set match accuracy for both non-BERT and BERT models; the second half shows execution accuracy of BERT models. Due to the number of model submissions constraint enforced by the Spider team, we only evaluate our BERT models on the test set.

test set. As a result, the new split consists of 7702 training examples and 1991 test examples. When using this split, the parser is tested on databases that all have been

seen during training. We evaluate the non-BERT parsers with the same metric of set match for evaluation.

Does the parser struggle out-of-domain? As in-domain and out-of-domain setting have different splits, and thus do not use the same test set, the direct comparison between them only serves as a proxy to illustrate the effect of domain shift. We show that, despite the original split of out-of-domain setting containing a larger number of training examples (8659 vs 7702), the base parser tested in-domain achieves a much better performance (78.2%) than its counterpart tested out-of-domain (56.4%). This suggests that the domain shift genuinely hurts the base parser.

Does DG-MAML hurt in-domain performance? We study DG-MAML in the in-domain setting to see if it hurts in-domain performance. Somewhat surprisingly, we instead observe a modest improvement (+1.1%) over the base parser. This suggests that DG-MAML, despite optimizing the model towards domain generalization, captures, to a certain degree, a more general notion of generalization or robustness, which appears beneficial even in the in-domain setting.

Model	**Dev**	**Test**
SyntaxSQLNet (Yu et al., 2018b)	16.4	13.3
Our Models		
RAT-SQL V2	31.0	23.0
RAT-SQL V2 + DG-MAML	**35.5**	**26.8**
With Multilingual BERT (M-BERT):		
RAT-SQL + M-BERT	41.4	37.3
RYANSQL + M-BERT (Choi et al., 2020)	41.3	34.7
Our Models		
RAT-SQL V2 + M-BERT	47.0	44.3
RAT-SQL V2 + M-BERT + DG-MAML	**50.1**	**46.9**

Table 3.8: Set match accuracy (%) on the development and test sets of Chinese Spider.

3.4.3 Linking Features and Limitations of Spider

Previous work addressed domain generalization by focusing on the sub-task of schema linking. For Spider, where questions and schemas are both in English, we leverage n-gram matching features in Section 3.1 which improve schema linking and significantly boost parsing performance. However, in Chinese Spider, it is not easy and obvious how to design such linking heuristics. Moreover, as pointed out by Suhr et al. (2020), the assumption that columns/tables are explicitly mentioned is not general enough, implying that exploiting matching features would not be a good general solution to domain generalization. Hence, we would like to see *whether DG-MAML can be beneficial when those features are not present*.

Specifically, we consider a variant of the base parser that does not use this feature, and train it with conventional supervised learning and with DG-MAML for Spider. As shown[9] in Table 3.9, we confirm that those features have a big impact on the base parser. More importantly, in the absence of those features, DG-MAML boosts the performance of the base parser by a larger margin. This is consistent with the observation that DG-MAML is more beneficial for Chinese Spider than Spider, in the sense that the parser would need to rely more on DG-MAML when these heuristics are not integrated or not available for domain generalization.

3.4.4 Additional Experiments and Analysis

We present analysis on DG-FMAML and probing how DG-MAML works. As the test sets for both datasets are not publicly available, we will use the development sets.

Effect of DG-FMAML We investigate the effect of the first-order approximation in DG-FMAML to see if it would provide a reasonable performance compared with DG-MAML. We evaluate it on the development sets of the two datasets, see Table 3.9. DG-FMAML consistently boosts the performance of the base parser, although it lags behind DG-MAML. For a fair comparison, we use the same batch size for DG-MAML and DG-FMAML. However, because DG-FMAML uses less memory, it could potentially benefit from a larger batch size. In practice, DG-FMAML is *twice as fast* to train than DG-MAML.

[9] Some results in Table 3.9 differ from Table 3.7. The former reports dev set performance over three runs, while the latter shows the best model, selected based on dev set performance.

Model	Dev (%)
Spider	
Base Parser	55.6 ± 0.5
+ DG-FMAML	56.8 ± 1.2
+ DG-MAML	$\mathbf{58.0} \pm \mathbf{0.8}$
Base Parser without Features	38.2 ± 1.0
+ DG-FMAML	41.8 ± 1.5
+ DG-MAML	$\mathbf{43.5} \pm \mathbf{0.9}$
Chinese Spider	
Base Parser	29.7 ± 1.1
+ DG-FMAML	32.5 ± 1.3
+ DG-MAML	$\mathbf{34.3} \pm \mathbf{0.9}$

Table 3.9: Accuracy (and $\pm 95\%$ confidence interval) on the development sets of Spider and Chinese Spider.

Probing Domain Generalization Schema linking has been the focus of previous work on zero-shot semantic parsing. We take the opposite direction and use this task to probe the parser to see if it, at least to a certain degree, achieves domain generalization due to improving schema linking. We hypothesize that *improving linking is the mechanism which prevents the parser from being trapped in overfitting the source domains*.

We propose to use 'relevant column recognition' as a probing task. Specifically, relevant columns refer to the columns that are mentioned in SQL queries. For example, the SQL query "*Select Status, avg(Population) From City Groupby Status*" in Figure

Model	Precision	Recall	F1
Spider			
Base Parser	70.0	70.4	70.2
Base Parser + DG-MAML	73.8	70.6	**72.1**
Chinese Spider			
Base Parser	61.5	60.4	61.0
Base Parser + DG-MAML	66.8	61.2	**63.9**

Table 3.10: Performance (%) of column prediction on the development sets of Spider and Chinese Spider.

3.1 contains two relevant columns: 'Status' and 'Population'. We formalize this task as a binary classification problem. Given a NL question and a column from the corresponding database, a classifier should predict whether the column is mentioned in the gold SQL query. We hypothesize that representations from the DG-MAML parser will be more predictive of relevance than those of the baseline, and the probing classifier will detect this difference in the quality of the representations.

We first obtain the representations for NL questions and schemas from the parsers and keep them fixed. The binary classifier is then trained based only on these representations. For classifier training we use the same split as the Spider dataset, i.e., the classifier is evaluated on unseen databases. The results are shown in Table 3.10. The classifier trained on the parser with DG-MAML achieves better performance. This confirms our hypothesis that using DG-MAML makes the parser produce better encodings of NL questions and database schemas and that this is one of the mechanisms the parsing model uses to ensure generalization.

3.5 Related Work

Cross-Domain Text-to-SQL Parsing Semantic parsing of NL to SQL recently surged in popularity thanks to the creation of two new multi-table datasets with the challenge of schema generalization – WikiSQL (Zhong et al., 2017) and Spider (Yu et al., 2018c). Schema encoding is not as challenging in WikiSQL as in Spider because it lacks multi-table relations. Schema linking is relevant for both tasks but also more challenging in Spider due to the richer NL expressiveness and less restricted SQL grammar observed in it. The state of the art semantic parser on WikiSQL (He et al., 2019) achieves a test set accuracy of 91.8%, significantly higher than the state of the art on Spider.

The recent state-of-the-art models evaluated on Spider use various attentional architectures for question/schema encoding and AST-based structural architectures for query decoding. IRNet (Guo et al., 2019b) encodes the question and schema separately with LSTM and self-attention respectively, augmenting them with custom type vectors for schema linking. They further use the AST-based decoder of Yin and Neubig (2017) to decode a query in an intermediate representation (IR) that exhibits higher-level abstractions than SQL. Bogin et al. (2019a) encode the schema with a GNN and a similar grammar-based decoder. Both works emphasize schema encoding and schema linking, but design separate featurization techniques to augment *word vectors* (as opposed to *relations between words and columns*) to resolve it. In contrast, the RAT-SQL framework

provides a unified way to encode arbitrary relational information among inputs.

Concurrently with RAT-SQL, Bogin et al. (2019b) published Global-GNN, a different approach to schema linking for Spider, which applies global reasoning between question words and schema columns/tables. Global reasoning is implemented by *gating* the GNN that encodes the schema using the question token representations. This differs from RAT-SQL in two important ways: (a) question word representations influence the schema representations but not vice versa, and (b) like in other GNN-based encoders, message propagation is limited to the schema-induced edges such as foreign key relations. In contrast, our relation-aware transformer mechanism allows encoding arbitrary relations between question words and schema elements explicitly, and these representations are computed jointly over all inputs using self-attention.

We use the same formulation of relation-aware self-attention as Shaw et al. (2018). However, they only apply it to sequences of words in the context of machine translation, and as such, their relation types only encode the relative distance between two words. We extend their work and show that relation-aware self-attention can effectively encode more complex relationships within an unordered set of elements (in our case, columns and tables within a database schema as well as relations between the schema and the question). To the best of our knowledge, this is the first application of relation-aware self-attention to joint representation learning with both predefined and softly induced relations in the input structure. Hellendoorn et al. (2020) develop a similar model concurrently with this work, where they use relation-aware self-attention to encode data flow structure in source code embeddings. Sun et al. (2018a) use a heterogeneous graph of KB facts and relevant documents for open-domain question answering. The nodes of their graph are analogous to the database schema nodes in RAT-SQL, but RAT-SQL also incorporates the question in the same formalism to enable joint representation learning between the question and the schema.

Our meta-learning inspired training algorithm is similar in spirit to Givoli and Reichart (2019), who also attempts to simulate source and target domains during learning. However, their optimization updates on virtual source and target domains are loosely connected by a two-step training procedure where a parser is first pre-trained on virtual source domains and then fine-tuned on virtual target domains. As we will show in Section 3.3, our training procedure does not fine-tune on virtual target domains but rather, uses them to evaluate a gradient step (for every batch) on source domains. This is better aligned with what is expected of the parser at test time: there will be no fine-tuning on *real* target domains at test time so there should not be any fine-tuning

on *simulated* ones at train time either. Moreover, Givoli and Reichart (2019) treat the division of training domains to virtual train and test domains as a hyper-parameter, which is possible for a handful of domains, but problematic when dealing with hundreds of domains as is the case for text-to-SQL parsing.

Meta-Learning for NLP Meta-learning has been receiving soaring interest in the machine learning community. Unlike conventional supervised learning, meta-learning operates on tasks, instead of data points. Most previous work (Vinyals et al., 2016; Ravi and Larochelle, 2016; Finn et al., 2017) has focused on few-shot learning where meta-learning helps address the problem of learning to learn fast for adaptation to a new task or domain. Applications of meta-learning in NLP are cast in a similar vein and include machine translation (Gu et al., 2018) and relation classification (Obamuyide and Vlachos, 2019). The meta-learning framework however is more general, with the algorithms or underlying ideas applied, e.g., to continual learning (Gupta et al., 2020), semi-supervised learning (Ren et al., 2018), multi-task learning (Yu et al., 2020b) and, as in our case, domain generalization (Li et al., 2018a).

Very recently, MAML was applied to semantic parsing tasks (Huang et al., 2018; Guo et al., 2019a; Sun et al., 2019). These approaches simulate *few-shot* learning scenarios in training by constructing a pseudo-task for each example. Given an example, similar examples are retrieved from the original training set. MAML then encourages strong performance on the retrieved examples after an update on the original example, simulating test-time fine-tuning. Lee et al. (2019) use matching networks (Vinyals et al., 2016) to enable *one-shot* text-to-SQL parsing where tasks for meta-learning are defined by SQL templates, i.e., a parser is expected to generalize to a new SQL template with one example. In contrast, the tasks we construct for meta-learning aim to encourage *generalization* across domains, instead of *adaptation* to a new task with one (or few) examples. One clear difference lies in how meta-train and meta-test sets are constructed. In previous work (e.g., Huang et al. 2018), these come from the *same* domain whereas we simulate domain *shift* and sample different sets of domains for meta-train and meta-test.

Domain Generalization Although the notion of domain generalization has been less explored in semantic parsing, it has been studied in other areas such as computer vision (Ghifary et al., 2015; Zaheer et al., 2017; Li et al., 2018b). Recent work (Li et al., 2018a; Balaji et al., 2018) employed optimization-based meta-learning to handle domain shift

issues in domain generalization. We employ the meta-learning objective originally proposed in Li et al. (2018a), where they adapt MAML to encourage generalization in unseen domains (of images). Based on this objective, we propose a cheap alternative that only requires first-order gradients, thus alleviating the overhead of computing second-order derivatives required by MAML.

3.6 Summary and Discussion

The task of cross-domain semantic parsing has been gaining momentum in recent years. To address the environment grounding and optimization mismatch problem that arise in this setting, we propose a new architecture, namely RAT-SQL, for cross-domain semantic parsing, and a model-agnostic training algorithm, namely DG-MAML, to boost domain generalization. Specifically, RAT-SQL handles environment grounding by addressing the schema encoding and linking challenges. Central to RAT-SQL is the relation-aware self-attention mechanism which jointly learns schema and question representations based on their alignment with each other and schema relations. To take the discrepancy between training and test domains into account during training, DG-MAML learns from a set of virtual cross-domain parsing tasks, instead of learning from individual data points. By optimizing towards better target-domain performance in each simulated task, DG-MAML encourages a parser to generalize better to unseen domains.

Follow-Up Work Since the publication of RAT-SQL, follow-up work have built upgraded version on top of it using pre-training technique or better decoders, or adapted it for more complex settings. Deng et al. (2020) propose a set of pre-training tasks (such as column grounding, value grounding) which are specifically designed to help RAT-SQL perform schema-linking. Yu et al. (2020a) augment RAT-SQL with better input representations which are obtained by pre-training over massive synthetically-generated parallel data based on handcrafted rules. Further in this direction, we also explored ways to pre-train RAT-SQL using synthetic data generated within a data-driven two-stage generative framework in Wang et al. (2021d). The original RAT-SQL use a grammar-based decoder to enforce syntactic constraints of SQL, but this design choice comes at the cost of efficiency as the decoder is hard to run in parallel. To address the efficiency issue, some recent work has tried to improve the decoder of RAT-SQL. Scholak et al. (2020) propose an efficient Transformer-based decoder which relies on

relation-aware attention to incorporate certain syntactic constraints. Rubin and Berant (2020) introduce a semi-autoregressive decoder which generates SQL programs in a bottom-up, rather than top-down autoregressive, manner. The speedup comes from the bottom-up strategy which allows decoding all sub-trees of a certain height in parallel. In this chapter, RAT-SQL is only applied on the single-turn setting where a user express an intent via one utterance. Yu et al. (2021) adapt RAT-SQL to more complex conversational settings where a user has multiple interactions with an interface, and the adapted RAT-SQL achieves state-of-the-art performance.

RAT-SQL can also be generalized to program formalisms other than SQL, as the core component of relation-aware encoding is very general and not restricted to SQL. In Chapter 5, we adapt RAT-SQL to logical form based semantic parsing and it obtains state-of-the-art performance as well.

Reproducibility The original version of RAT-SQL is available at `https://github.com/Microsoft/rat-sql`. The updated version of RAT-SQL which contains the implementations of DG-MAML and supports SQL value prediction is available at `https://github.com/berlino/tensor2struct-public`.

Chapter 4

Learning from Denotations

The bottleneck of semantic parsing, like many other areas in NLP, is the lack of enough labeled data while contemporary neural systems are typically data-hungry. Specifically, in semantic parsing, we aim to build parsers that can perform well on some target domains of interest, but data collection on target domains are typically expensive. In the previous chapter, we built cross-domain semantic parsers, along with a specialized training algorithm, so that the parsers can generalize well to unseen target domains, without the need to collect any form of labeled data from the target domains. Cross-domain generalization is indeed appealing but there seems to be a performance upper bound on cross-domain semantic parsers as many questions require domain-specific knowledge (Suhr et al., 2020). In this and the following chapter, we explore an orthogonal direction on how to take advantage of cheaper labeled data, or unlabeled data.

Annotating programs (e.g., SQL) for given utterances would require expert knowledge of programs (e.g., knowing how to program in SQL) and the environment against which they are executed (e.g., knowing which table/columns from a relational database to use). In some cases, where the environments are simple for humans to interpret (e.g., Web tables), it is possible to reduce the burden of annotation by only labeling answers (or denotations) that correspond to questions[1]. However, the learning signals provided by denotations are much weaker than those from programs, which poses a great generalization challenge for a semantic parser to learn from them. In this chapter, we seek a parser that can take better advantage of question-denotation pairs in this weakly supervised semantic parsing setting (Liang et al., 2011; Berant et al., 2013) such

[1]Note that this assumption does not hold for large and complex relational databases where answers (i.e., execution results of SQL) could be long, thus hard for manual annotation.

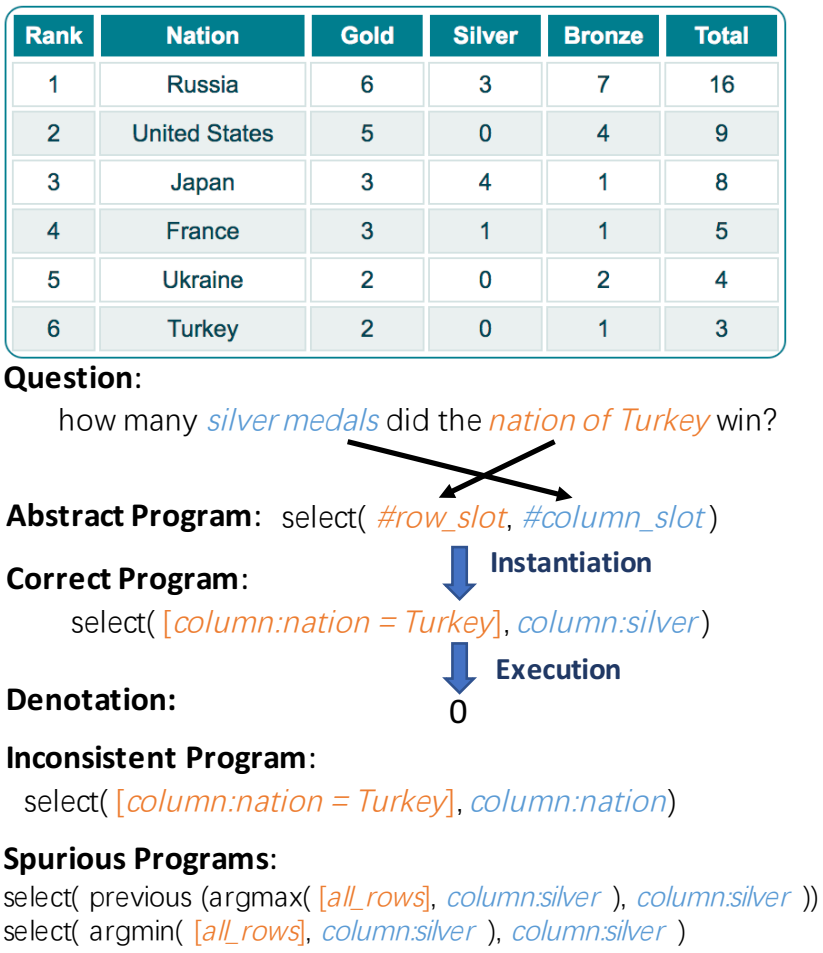


Figure 4.1: After generating an abstract program for a question, our parser finds alignments between slots (with prefix #) and question spans. Based on the alignment, it instantiates each slot and a complete program is executed against a table to obtain a denotation. Spurious programs can also execute to the correct denotation, but their slots cannot be well-aligned.

that the annotation effort for building natural language interfaces is reduced.

Two major unique challenges arise when learning from denotations:

- training of a semantic parser requires exploring a large search space of possible programs to find those which are *consistent*, i.e., execute to correct denotations;
- a parser should be robust to *spurious* programs which accidentally execute to correct denotations, but do not reflect the semantics of a question.

To handle these challenges, we propose a weakly-supervised neural semantic parser that features structured latent alignments to bias learning towards *correct* programs which are consistent but not spurious.

Our intuition is that correct programs should respect certain constraints, were they to be aligned to the question text, while spurious and inconsistent programs do not. For instance, in Figure 4.1, the answer to the question ("0") can be obtained by executing the correct program which selects the number of Turkey's silver medals. However, the same answer can be also obtained by the spurious programs shown in the figure. The first program can be paraphrased as: find the row with the largest number of silver medals and then select the number of silver medals from the previous row. The spurious programs differ from the correct one in that they repeatedly use the column "silver". Whereas, in the question, the word "silver" only refers to the target column containing the answer, it also mistakenly triggers the appearance of the column "silver" in the row selection condition. This constraint, i.e., that a text span within a question cannot trigger two semantically distinct operations (e.g., selecting target rows and target columns) can provide a useful inductive bias. We propose to capture structural constraints by modeling the alignments between programs and questions *explicitly* as structured latent variables.

Considering the large search space of possible programs, an alignment model that takes into account the full range of correspondences between program operations and question spans would be very expensive. To make the process tractable, we introduce a two-stage approach that features *abstract* programs. Specifically, we decompose semantic parsing into two steps: 1) a natural language utterance is first mapped to an abstract program which is a composition of high-level operations; 2) the abstract program is then instantiated with low-level operations that usually involve relations and entities specific to the environment at hand. This decomposition is motivated by the observation that only a small number of sensible abstract programs can be instantiated into consistent programs. Similar ideas of using abstract meaning representations have been explored with fully-supervised semantic parsers (Dong and Lapata, 2018; Finn et al., 2017) and in other related tasks (Goldman et al., 2018; Herzig and Berant, 2018; Nye et al., 2019).

For structured data in tabular format, we abstract two basic operations of row selection and column selection from programs: these are handled in the second (instantiation) stage. As shown in Figure 4.1, the question is first mapped to the abstract program "select (#*row_slot*, #*column_slot*)" whose two slots are subsequently instantiated with filter conditions (row slot) and a column name (column slot). During the instantiation of abstract programs, each slot should refer to the question to obtain its specific semantics. In Figure 4.1, *row_slot* should attend to "nation of Turkey" while *column_slot* needs to

attend to "silver medals". The structural constraint discussed above now corresponds to assuming that each span in a question can be aligned to a *unique* row or column slot. Under this assumption, the instantiation of spurious programs will be discouraged. The uniqueness constraint would be violated by both spurious programs in Figure 4.1, since "column:silver" appears in the program twice but can be only aligned to the span "silver medals" once.

The first stage (i.e., mapping a question onto an abstract program) is handled with a sequence-to-sequence model. The second stage (i.e., program instantation) is approached with local classifiers: one per slot in the abstract program. The classifiers are conditionally independent given the abstract program and a latent alignment. Instead of marginalizing out alignments, which would be intractable, we use structured attention (Kim et al., 2017), i.e., we compute the marginal probabilities for individual span-slot alignment edges and use them to weight the input to the classifiers. As we discuss below, the marginals in our constrained model are computed with dynamic programming.

We perform experiments on two open-domain question answering datasets in the setting of learning from denotations. Our model achieves an execution accuracy of 44.5% in WIKITABLEQUESTIONS and 79.3% in WIKISQL, which both surpass previous state-of-the-art methods in the same weakly-supervised setting. In WIKISQL, our parser is better than recent supervised parsers that are trained on question-program pairs.

Contributions From this chapter, we begin to explore weaker forms of supervision than utterance-program pairs. In this chapter, we aim to utilize denotations of programs, rather than programs, as they are easier to obtain for certain forms of knowledge (e.g., tabular knowledge). We approach the problem of learning from denotation from the perspective of incorporating structured inductive biases into a parser:

- we introduce an alignment model as a means of differentiating between correct and spurious programs;
- we propose a neural semantic parser that performs tractable alignments by first mapping questions to abstract programs.

Empirically, the parser achieve state-of-the-art performance on two semantic parsing benchmarks, namely WIKITABLEQUESTIONS and WIKISQL, at the time of publication. The code for reproducing the results in this chapter is available at `https://github.com/berlino/weaksp_em19`.

4.1 Background

Given table t, our task is to map a natural utterance x to program z, which is then executed against a table to obtain denotation $[[z]]_t = d$. We train our parser only based on d without access to correct programs z^*. Our experiments focus on two benchmarks, namely WIKITABLEQUESTIONS (Pasupat and Liang, 2015) and WIKISQL (Zhong et al., 2017) where each question is paired with a Wikipedia table and a denotation. Figure 4.1 shows a simplified example taken from WIKITABLEQUESTIONS.

4.1.1 Grammars

Executable programs z that can query tables are defined according to a language. Specifically, the search space of programs is constrained by grammar rules so that it can be explored efficiently. We adopt the the variable-free language of Liang et al. (2018) and define an *abstract* grammar and an *instantiation* grammar which decompose the generation of a program in two stages.[2]

The first stage involves the generation of an abstract version of a program which, in the second stage, gets instantiated. Abstract programs only consider compositions of high-level functions, such as superlatives and aggregation, while low-level functions and arguments, such as filter conditions and entities, are taken into account in the next step. In our table-based datasets, abstract programs do not include two basic operations of querying tables: row selection and column selection. These operations are handled at the instantiation stage. In Figure 4.1 the abstract program has two slots for row and column selection, which are filled with the conditions "column:nation = Turkey" and "column:silver" at the instantiation stage. The two stages can be easily merged into one step when conducting symbolic combinatorial search. The motivation for the decomposition is to facilitate the learning of our neural semantic parser and the handling of structured alignments.

Abstract Grammar Our abstract grammar has five basic types: ROW, COLUMN, STRING, NUMBER, and DATE; COLUMN is further sub-typed into STRING_COLUMN, NUMBER_COLUMN, and DATE_COLUMN; other basic types are augmented with LIST to represent a list of elements like LIST[ROW]. Arguments and return values of functions are typed using these basic types.

[2]We also extend their grammar to additionally support operations of conjunction and disjunction.

Abstract Program:

select (first (#rows_slot) #column_slot)

Derivation Tree:

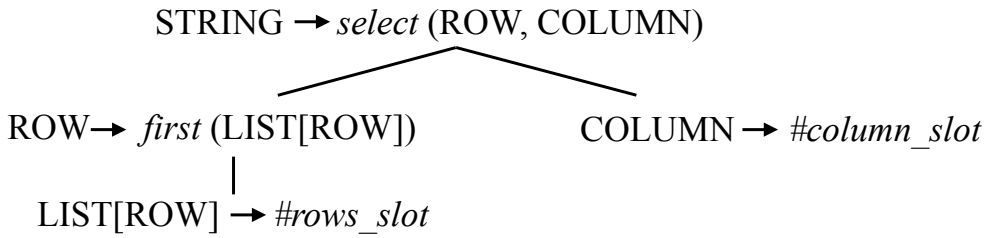


Figure 4.2: An abstract program and its derivation tree. Capitalized words indicate types of function arguments and their return value.

Function composition can be defined recursively based on a set of production rules, each corresponding to a function type. For instance, function ROW $\rightarrow$ *first*(LIST[ROW]) selects the first row from a list of rows and corresponds to production rule "ROW $\rightarrow$ *first*".

The abstract grammar has two additional types for slots (aka terminal rules) which correspond to row and column selection:

$$\text{LIST[ROW]} \rightarrow \textit{\#row\_slot}$$
$$\text{COLUMN} \rightarrow \textit{\#column\_slot}$$

An example of an abstract program and its derivation tree is shown in Figure 4.2. We linearize the derivation by traversing it in a left-to-right depth-first manner. We represent the tree in Figure 4.2 as a sequence of production rules: "ROOT $\rightarrow$ STRING, STRING $\rightarrow$ *select*, ROW $\rightarrow$ *first*", LIST[ROW] $\rightarrow$ *#row_slot*, COLUMN $\rightarrow$ *#column_slot*". The first action is always to select the return type for the root node.

Given a specific table t, the abstract grammar $\mathcal{H}_t$ will depend on its column types. For example, if the table does not have number cells, "max/min" operations will not be executable.

Instantiation Grammar A column slot is directly instantiated by selecting a column; a row slot is filled with one or multiple conditions (COND) which are jointed together with conjunction (*OR*) and disjunction (*AND*) operators:

COND $\rightarrow$ COLUMN OPERATOR VALUE
#row_slot $\rightarrow$ COND (*AND* COND)*
#row_slot $\rightarrow$ COND (*OR* COND)*

where OPERATOR $\in [>, <, =, \geq, \leq]$ and VALUE is an entity in the form of a string ('UK'), a number (e.g., '3'), or a date (e.g., 'July 1993'). A special condition *#row_slot* $\rightarrow$ *all_rows* is defined to signify that a program queries all rows.

4.1.2 Search for Consistent Programs

A problematic aspect of learning from denotations is that, since annotated programs are not available (e.g., for WIKITABLEQUESTIONS), we have no means to directly evaluate the proposed grammar. As an evaluation proxy, we measure the coverage of our grammar in terms of consistent programs. Specifically, we exhaustively search for all consistent programs for each question in the training set. While the space of programs is exponential, we observed that abstract programs that can be instantiated into correct programs are not very complex in terms of the number of production rules. When enumerating programs, we restrict the number of production rules at the first stage of generating abstract programs, and in this way the search process becomes tractable.

We find that 83.6% of questions in WIKITABLEQUESTIONS are covered by at least one consistent program. However, each question eventually has 200 consistent programs on average and most of them are spurious. Treating them as ground truth poses a great challenge for learning a semantic parser. The coverage for WIKISQL is 96.6% and each question generates 84 consistent programs on average.

Another important observation is that there is only a limited number of abstract programs that can be instantiated into consistent programs. The number of such abstract programs is 23 for WIKITABLEQUESTIONS and 6 for WIKISQL, suggesting that there are a few patterns underlying several utterances. This motivates us to design a semantic parser that first maps utterances to abstract programs. For the sake of generality, we do not restrict our parser to abstract programs in the training set.[3]

4.2 Model

After obtaining consistent programs z for each question through offline search, we next show how to learn a parser that can generalize to new questions and tables.

[3]That is, generating abstract programs is not reduced to a multi-label classification problem. We elaborate on this later.

4.2.1 Training and Inference

Our learning objective $\mathcal{J}$ is to maximize the log-likelihood of the marginal probability of all consistent programs, which are generated by mapping an utterance x to an interim abstract program h:

$$\mathcal{J} = \log\{\sum_{h \in \mathcal{H}_t} p(h|x,t) \sum_{[[z]]=d} p(z|x,t,h)\}. \tag{4.1}$$

During training, our model only needs to focus on abstract programs that have successful instantiations of consistent programs and it does not have to explore the whole space of possible programs.

At test time, a parser chooses the program $\hat{z}$ with the highest probability:

$$\hat{h}, \hat{z} = \arg\max_{h \in \mathcal{H}_t,\ z} p(h|x,t)p(z|x,t,h). \tag{4.2}$$

To make it more efficient, we only choose top-k abstract programs to instantiate through beam search. $\hat{z}$ is then executed to obtain its denotation as the final prediction.

Next, we will explain the basic components of our neural parser. Our model first encodes a question and a table with an input encoder; then generates abstract programs with a seq2seq model. Finally, these abstract programs are instantiated while relying on a structured alignment model.

4.2.2 Input Encoder

Each word in an utterance is mapped to a distributed representation through an embedding layer. Following previous work (Neelakantan et al., 2017; Liang et al., 2018), we also add an indicator feature specifying whether the word appears in the table. This feature is mapped to a learnable vector. Additionally, in WIKITABLEQUESTIONS, we use POS tags from the CoreNLP annotations released with the dataset and map them to vector representations. The final representation for a word is the concatenation of the vectors above. A bidirectional LSTM (Hochreiter and Schmidhuber, 1997) is then used to obtain a contextual representation $\boldsymbol{l}_i$ for the i_{th} word.

A table is represented by a set of columns. Each column is encoded by averaging the embeddings of words under its column name. We also have a column type feature (i.e., number, date or string) and an indicator feature signaling whether at least one entity in the column appears in the utterance.

4.2.3 Generating Abstract Programs

Instead of extracting abstract programs as templates, similarly to Xu et al. (2017) and Finn et al. (2017), we generate them with a seq2seq model. Although template-based approaches would be more efficient in practice, a seq2seq model is more general since it could generate unseen abstract programs which fixed templates could not otherwise handle.

Our goal is to generate a sequence of production rules that lead to abstract programs. During decoding, the hidden state $\boldsymbol{g}_j$ of the j_{th} timestep is computed based on the previous production rule, which is mapped to an embedding $\boldsymbol{a}_{j-1}$. We also incorporate an attention mechanism (Luong et al., 2015) to compute a contextual vector $\boldsymbol{b}_j$. Finally, a score vector $\boldsymbol{s}_j$ is computed by feeding the concatenation of the hidden state and context vector to a multilayer perceptron (MLP):

$$\begin{aligned} \boldsymbol{g}_j &= \text{LSTM}(\boldsymbol{g}_{j-1}, \boldsymbol{a}_{j-1}) \\ \boldsymbol{b}_j &= \text{Attention}(\boldsymbol{g}_j, \boldsymbol{l}) \\ \boldsymbol{s}_j &= \text{MLP}_1([\boldsymbol{g}_j; \boldsymbol{b}_j]) \\ p(a_j|x,t,a_{<j}) &= \text{softmax}_{a_j}(\boldsymbol{s}_j) \end{aligned} \tag{4.3}$$

where the probability of production rule a_j is computed by the softmax function. According to our abstract grammar, only a subset of production rules will be valid at the j-th time step. For instance, in Figure 4.2, production rule "STRING $\rightarrow$ *select*" will only expand to rules whose left-hand side is ROW, which is the type of the first argument of *select*. In this case, the next production rule is "ROW $\rightarrow$ *first*". We thus restrict the normalization of softmax to only focus on these valid production rules. The probability of generating an abstract program $p(h|x,t)$ is simply the product of the probability of predicting each production rule $\prod_j p(a_j|x,t,a_{<j})$.

After an abstract program is generated, we need to instantiate slots in abstract programs. So the desired representation of slots should depend on the meaning of abstract programs. To achieve this, our model first encodes the abstract program using a bi-directional LSTM, similar to Dong and Lapata (2018). As a result, the representation of a slot is contextually aware of the entire abstract program .

4.2.4 Instantiating Abstract Programs

To instantiate an abstract program, each slot must obtain its specific semantics from the question. We model this process by an alignment model which learns the corre-

spondence between slots and question spans. Formally, we use a binary alignment matrix $\boldsymbol{A}$ with size $m \times n \times n$, where m is the number of slots and n is the number of tokens. In Figure 4.1, the alignment matrix will only have $\boldsymbol{A}_{0,6,8} = 1$ and $\boldsymbol{A}_{1,2,3} = 1$ which indicates that the first slot is aligned with "nation of Turkey", and the second slot is aligned with "silver medals". The second and third dimension of the matrix represent the start and end position of a span.

We model alignments as discrete latent variables and condition the instantiation process on the alignments as follows:

$$\sum_{[z]=d} p(z|x,t,h) = \sum_{\boldsymbol{A}} p(\boldsymbol{A}|x,t,h) \sum_{[z]=d} p(z|x,t,h,\boldsymbol{A}). \tag{4.4}$$

We will first discuss the instantiation model $p(z|x,t,h,\boldsymbol{A})$ and then elaborate on how to avoid marginalization in the next section.

Each slot in an abstract program can be instantiated by a set of candidates following the instantiation grammar. For efficiency, we use local classifiers to model the instantiation of each slot independently:

$$p(z|x,t,h,\boldsymbol{A}) = \prod_{s \in S} p(s \rightarrow c|x,t,h,\boldsymbol{A}), \tag{4.5}$$

where S is the set of slots and c is a candidate following our instantiation grammar. "$s \rightarrow c$" represents the instantiation of slot s into candidate c.

Recall that there are two types of slots, one for rows and one for columns. All column names in the table are potential instantiations of column slots. We represent each column slot candidate by the average of the embeddings of words in the column name. Based on our instantiation grammar in Section 4.1.1, candidates for row slots are represented as follows: 1) each condition is represented with the concatenation of the representations of a column, an operator, and a value. For instance, condition "string_column:nation = Turkey" in Figure 4.1 is represented by vector representations of the column 'nation', the operator '=', and the entity 'Turkey'; 2) multiple conditions are encoded by averaging the representations of all conditions and adding a vector representation of *AND*/*OR* to indicate the relation between them.

For each slot, the probability of generating a candidate is computed with softmax normalization on a score function:

$$p(s \rightarrow c|x,t,h,\boldsymbol{A}) \propto \exp\{\text{MLP}([\boldsymbol{s};\boldsymbol{c}])\}, \tag{4.6}$$

where $\boldsymbol{s}$ is the representation of the span that slot s is aligned with, and $\boldsymbol{c}$ is the representation of candidate c. The representations $\boldsymbol{s}$ and $\boldsymbol{c}$ are concatenated and fed to a

MLP. We use the same MLP architecture but different parameters for column and row slots.

4.2.5 Structured Attention

We first formally define a few structural constraints over alignments and then explain how to incorporate them efficiently into our parser.

The intuition behind our alignment model is that row and column selection operations represent distinct semantics, and should therefore be expressed by distinct natural language expressions. Hence, we propose the following constraints:

Unique Span In most cases, the semantics of a row selection or a column selection is expressed uniquely with a single contiguous span:

$$\forall k \in [1, |S|], \quad \sum_{i,j} \boldsymbol{A}_{k,i,j} = 1, \tag{4.7}$$

where $|S|$ is the number of slots.

No Overlap Spans aligned to different slots should not overlap. Formally, at most one span that contains word i can be aligned to a slot:

$$\forall i \in [1, n], \quad \sum_{k,j} \boldsymbol{A}_{k,i,j} \leq 1. \tag{4.8}$$

As an example, the alignments in Figure 4.1 follow the above constraints. Intuitively, the one-to-one mapping constraint aims to assign distinct and non-overlapping spans to slots of abstract programs. To further bias the alignments and improve efficiency, we impose additional restrictions: (1) a row slot must be aligned to a span that contains an entity since conditions that instantiate the slot would require entities for filtering; (2) a column slot must be aligned to a span with length 1 since most column names only have one word.

Marginalizing out all $\boldsymbol{A}$ in Equation (4.4) would be very expensive considering the exponential number of possible alignments. We approximate the marginalization by moving the outside expectation directly inside over $\boldsymbol{A}$. As a result, we instead optimize the following objective:

$$\mathcal{J} \approx \log \Big\{ \sum_{h \in \mathcal{H}_t} p(h|x,t) \sum_{[[z]]=d} p(z|x,t,h,\mathbb{E}[\boldsymbol{A}]) \Big\}, \tag{4.9}$$

where $\mathbb{E}[\boldsymbol{A}]$ are the marginals of $\boldsymbol{A}$ with respect to $p(\boldsymbol{A}|x,t,h)$.

The idea of using differentiable surrogates for discrete latent variables has been used in many other works like differentiable data structures (Grefenstette et al., 2015; Graves et al., 2014) and attention-based networks (Bahdanau et al., 2015; Kim et al., 2017). Using marginals $\mathbb{E}[\boldsymbol{A}]$ can be viewed as *structured attention* between slots and question spans.

The marginal probability of the alignment matrix $\boldsymbol{A}$ can be computed efficiently using dynamic programming (see Täckström et al. 2015 for details). An alignment is encoded into a path in a weighted lattice where each vertex has $2^{|S|}$ states to keep track of the set of covered slots. The marginal probability of edges in this lattice can be computed by the forward-backward algorithm (Wainwright et al., 2008). The lattice weights, represented by a scoring matrix $\boldsymbol{M} \in \mathbb{R}^{m \times n \times n}$ for all possible slot-span pairs, are computed using the following scoring function:

$$\boldsymbol{M}_{k,i,j} = \text{MLP}_2([\boldsymbol{r}(k); \textbf{span}[i:j]]), \tag{4.10}$$

where $\boldsymbol{r}(k)$ represents the k_{th} slot and $\textbf{span}[i:j]$ represents the span from word i to j. Recall that we obtain $\boldsymbol{r}(k)$ by encoding a generated abstract program. A span is represented by averaging the representations of the words therein. These two representations are concatenated and fed to a MLP to obtain a score. Since $\mathbb{E}[\boldsymbol{A}]$ is not discrete anymore, the aligned representation of slot $\boldsymbol{s}$ in Equation (4.6) becomes the weighted average of representations of all spans in the set.

4.3 Experiments

We evaluated our model on two semantic parsing benchmarks, WIKITABLEQUESTIONS and WIKISQL. We compare against two common baselines to demonstrate the effectiveness of using abstract programs and alignment. We also conduct detailed analysis which shows that structured attention is highly beneficial, enabling our parser to differentiate between correct and spurious programs. Finally, we break down the errors of our parser so as to examine whether structured attention is better at instantiating abstract programs.

4.3.1 Experimental Setup

Datasets WIKITABLEQUESTIONS contains 2,018 tables and 18,496 utterance-denotation pairs. The dataset is challenging as 1) the tables cover a wide range of domains and unseen tables appear at test time; and 2) the questions involve a variety of operations such

Supervised by Denotations	Dev.	Test
Pasupat and Liang (2015)	37.0	37.1
Neelakantan et al. (2017)	34.1	34.2
Haug et al. (2018)	—	34.8
Zhang et al. (2017)	40.4	43.7
Liang et al. (2018)	42.6	43.9
Agarwal et al. (2019)	43.2	44.1
Typed Seq2Seq	37.3	38.3
Abstract Programs		
f.w. standard attention	39.4	41.4
f.w. structured attention	**43.7**	**44.5**

Table 4.1: Results on WIKITABLEQUESTIONS. *f.w.* stands for slots filled with.

as superlatives, comparisons, and aggregation (Pasupat and Liang, 2015). WIKISQL has 24,241 tables and 80,654 utterance-denotation pairs. The questions are logically simpler and only involve aggregation, column selection, and conditions. The original dataset is annotated with SQL queries, but we only use the execution result for training. In both datasets, tables are extracted from Wikipedia and cover a wide range of domains, and training and testing tables are disjoint.

Entity extraction is important during parsing since entities are used as values in filter conditions during instantiation. String entities are extracted by string matching utterance spans and table cells. In WIKITABLEQUESTIONS, numbers and dates are extracted from the CoreNLP (Manning et al., 2014) annotations released with the dataset. WIKISQL does not have entities for dates, and we use string-based normalization to deal with numbers.

Implementation We obtained word embeddings by a linear projection of GloVe pre-trained embeddings (Pennington et al., 2014) which were fixed during training. Attention scores were computed based on the dot product between two vectors. Each MLP is a one-hidden-layer perceptron with ReLU as the activation function. Dropout (Srivastava et al., 2014) was applied to prevent overfitting. All models were trained with Adam (Kingma and Ba, 2015). Implementations of abstract and instantiation grammars were based on AllenNLP (Gardner et al., 2017).

4.3.2 Baselines

Aside from comparing our model against previously published approaches, we also implemented the following baselines:

Typed Seq2Seq Programs were generated using a sequence-to-sequence model with attention (Dong and Lapata, 2016). Similar to Krishnamurthy et al. (2017), we constrained the decoding process so that only well-formed programs are predicted. This baseline can be viewed as merging the two stages of our model into one stage where generation of abstract programs and their instantiations are performed with a shared decoder.

Standard Attention The aligned representation of slot s in Equation (4.6) is computed by a standard attention mechanism: $\boldsymbol{s} = \text{Attention}(\boldsymbol{r}(s), \boldsymbol{l})$ where $\boldsymbol{r}(s)$ is the representation of slot s from abstract programs. Each slot is aligned independently with attention, thus there are no global structural constraints on alignments.

4.3.3 Main Results

Results on WIKITABLEQUESTIONS are shown in Table 4.1. The structured-attention model achieves the best performance, compared against the two baselines and previous approaches. The standard attention baseline with abstract programs is superior to the typed Seq2Seq model, demonstrating the effectiveness of decomposing semantic parsing into two stages. Results on WIKISQL are shown in Table 4.2. The structured-attention model is again superior to our two baseline models. Interestingly, its performance surpasses previously reported weakly-supervised models (Liang et al., 2018; Agarwal et al., 2019) and is even on par with fully supervised ones (Dong and Lapata, 2018).

The gap between the standard attention baseline and the typed Seq2Seq model is not very large on WIKISQL, compared to WIKITABLEQUESTIONS. Recall from Section 4.1.2 that WIKISQL only has 6 abstract programs that can be successfully instantiated. For this reason, our decomposition alone may not be very beneficial if coupled with standard attention. In contrast, our structured-attention model consistently performs much better than both baselines.

For complete comparison with previous work, we also provide the performance of ensembling our best model that utilizes abstract programs and structured attention in Table 4.3. By using the same ensemble size, our model consistently achieves better

Supervised by Programs	Dev.	Test
Zhong et al. (2017)	60.8	59.4
Wang et al. (2017a)	67.1	66.8
Xu et al. (2017)	69.8	68.0
Huang et al. (2018)	68.3	68.0
Yu et al. (2018a)	74.5	73.5
Sun et al. (2018b)	75.1	74.6
Dong and Lapata (2018)	79.0	78.5
Shi et al. (2018)	**84.0**	**83.7**
Supervised by Denotations	Dev.	Test
Liang et al. (2018)	72.2	72.6
Agarwal et al. (2019)	74.9	74.8
Typed Seq2Seq	74.5	74.7
Abstract Programs		
f.w. standard attention	75.2	75.3
f.w. structured attention	**79.4**	**79.3**

Table 4.2: Results on WIKISQL. *f.w.*: slots filled with.

performance compared with Liang et al. (2018) and Agarwal et al. (2019) .

4.3.4 Analysis of Spuriousness

To understand how well structured attention can help a parser differentiate between correct and spurious programs, we analyzed the posterior distribution of consistent programs given a denotation: $p(z|x,t,d)$ where $[[z]] = d$.

WIKISQL includes gold-standard SQL annotations, which we do not use in our experiments but exploit here for analysis. Specifically, we converted the annotations released with WIKISQL to programs licensed by our grammar. We then computed the log-probability of these programs according to the posterior distribution as a measure of how well a parser can identify them amongst all consistent programs $\log \sum_{z^*} p(z^*|x,t,d)$, where z^* denotes correct programs. The average log-probability assigned to correct programs by structured and standard attention is -0.37 and -0.85, respectively. This gap confirms that structured attention can bias our parser towards correct programs during

Models	WIKITABLEQUESTIONS	WIKISQL
Liang et al. (2018)	46.3	74.2
Agarwal et al. (2019)	46.9	76.9
Our model	**47.3**	**81.7**

Table 4.3: Results of ensembled models. The ensemble size for WIKITABLEQUESTIONS and WIKISQL are 10 and 5 respectively.

Error Types	standard	structured
Abstraction Error	19.2	20.0
Instantiation Error	41.5	36.2
Coverage Error	39.2	43.8

Table 4.4: Proportion of errors on the development set in WIKITABLEQUESTIONS.

learning.

4.3.5 Error Analysis

We further manually inspected the output of our structured-attention model and the standard attention baseline in WIKITABLEQUESTIONS. Specifically, we randomly sampled 130 error cases independently from both models and classified them into three categories.

Abstraction Errors If a parser fails to generate an abstract program, then it is impossible for it to instantiate a consistent complete program.

Instantiation Errors These errors arise when abstract programs are correctly generated, but are mistakenly instantiated either by incorrect column names or filter conditions.

Coverage Errors These errors arise from implicit assumptions made by our parser: a) there is a long tail of unsupported operations that are not covered by our abstract programs; b) if entities are not correctly identified and linked, abstract programs cannot be correctly instantiated.

Table 4.4 shows the proportion of errors attested by the two attention models. We observe that structured attention suffers less from instantiation errors compared against the standard attention baseline, which points to the benefits of the structured alignment model.

4.4 Related Work

Learning from Denotations To improve the efficiency of searching for consistent programs, Zhang et al. (2017) use a macro grammar induced from cached consistent programs. Unlike Zhang et al. (2017) who abstract entities and relations from logical forms, we take a step further and abstract the computation of row and column selection. Our work also differs from Pasupat and Liang (2016) who resort to manual annotations to alleviate spuriousness. Instead, we equip our parser with an inductive bias to rule out spurious programs during training. Recently, reinforcement learning based methods address the computational challenge by using a memory buffer (Liang et al., 2018) which stores consistent programs and an auxiliary reward function (Agarwal et al., 2019) which provides feedback to deal with spurious programs. Guu et al. (2017) employ various strategies to encourage even distributions over consistent programs in cases where the parser has been misled by spurious programs.

Modelling Alignments In classical semantic parsing systems, alignments are explicitly and naturally accommodated within certain grammar formalisms. For example, Zettlemoyer and Collins (2005) leverage CCG where alignments are intrinsic properties of CCG lexicons. Wong and Mooney (2007b) employ synchronous context-free grammars to model the correspondences between natural questions and lambda calculus. These grammar formalisms account for *full correspondences* between natural language and programs in the sense that alignments are built in a hierarchical manner where alignments at all levels, from word-level to phrase-level, are jointly considered. This is of course highly desirable based on the common belief that natural language, as well as programs, are built with hierarchical compositions of primitives (such as words). However, computational systems based on such grammar formalisms are typically expensive as the space of full correspondences is very large. Our work can be situated between these symbolic systems and neural seq2seq with attention networks (Bahdanau et al., 2015) in the sense that it models *partial correspondences* of certain fragments, i.e., NL and program fragments about row and column selection in this chapter. This allows for

more tractable computation as well as retaining end-to-end training of seq2seq models.

4.5 Summary and Discussion

To address the challenges that arise in the setting of learning from denotations, we proposed a neural semantic parser that features abstract programs and latent structured alignments. Our parser achieves state-of-the-art performance on two benchmarks, WIKITABLEQUESTIONS and WIKISQL. Empirical analysis shows that the inductive bias introduced by the alignment model helps our parser differentiate between correct and spurious programs. As a result, the annotation effort for building semantic parsers can be indeed reduced by taking better advantage of cheap supervision of denotations.

As we mentioned before, the assumption that denotations are easier to obtain than programs is only true in cases where the environments are simple enough for humans to quickly interpret. Hence, in this chapter, we will mainly focus on semantic parsing tasks where the environments are small Web tables, instead of large and complex relational databases or knowledge bases.

Structured Alignments Although we use structured alignments to mostly enforce the uniqueness constraint described above, other types of inductive biases can be useful and could be encoded in our two-stage framework. For example, we could replace the uniqueness constraint with modeling the number of slots aligned to a span, or favor sparse alignment distributions. Crucially, the two-stage framework makes it easier to inject prior knowledge about datasets and formalisms while maintaining efficiency. Alignments can exhibit different properties (e.g., monotonicity or bijectivity), depending on the meaning representation language (e.g., logical forms or SQL), the definition of abstract programs, and the domain at hand. These properties can be often captured within a probabilistic alignment model and hence provide a useful inductive bias to the parser. In Chapter 6, we will introduce another semantic parser that can capture latent segment-to-segment alignments for generalization.

Pre-trained Representations Though we do not leverage pre-trained representations such as BERT (Devlin et al., 2019) in experiments, they have shown to be very useful for the task of learning from denotations in later work (Min et al., 2019b). Recent work on table-specific representations (Yin et al., 2020; Yu et al., 2020a) shows that these specialized representations can further resolve the issue of spurious programs. Note that

these representations can be easily coupled with our parser by upgrading the embedding layer to a pre-trained encoder. Concretely, in Yu et al. (2020a), we show that, when our parser is augmented with specialized pre-trained representations, it can achieve new state-of-the-art performance.

In the next chapter, we take a step further in the same direction of utilizing weak supervision, and explore ways of taking advantage of unlabeled data. Instead of baking latent alignments (i.e., model biases) into the parser, we use a different methodology of developing specialized training objectives (i.e., learning biases).

Chapter 5

Learning from Executions

Following the same thread as the previous chapter, we continue to study the ways of utilizing examples that are cheaper to obtain than labeled ones for semantic parsing. Specifically, we explore the possibility of further reducing the burden of annotation by looking into the more extreme setting where there are no annotations available at all (neither as programs or as denotations) for a large number of utterances. This semi-supervised learning setting resembles a common real-life scenario where a small amount of labeled examples results in a reasonably good semantic parser that is acceptable by end-user upon deploying, and massive numbers of user utterances can be collected afterwards (Iyer et al., 2017). Effectively utilizing the unlabeled data makes it possible for a semantic parser to improve over time without human involvement, resulting in a continuous positive feedback loop.

Our key observation is that not all candidate programs[1] for an utterance will be semantically valid. This implies that only some candidate programs can be executed and obtain non-empty execution results.[2] As illustrated in Figure 5.1, executability is a weak signal that can differentiate between semantically valid and invalid programs. On unlabeled utterances, we can encourage a parser to only focus on executable programs ignoring non-executable ones. Moreover, the executability of a program can be obtained from an executor for free without requiring human effort. Executability has previously been used to guide the decoding of a semantic parser (Wang et al., 2018). We take a step further to directly use this weak signal for learning from unlabeled utterances.

The assumption underlying executability is that programs, apart from satisfying

[1] As illustrated in Figure 2.1 of Chapter 2, candidate programs are those licensed by a particular grammar.

[2] In the rest of this chapter, we extend the meaning of 'executability', and use it to refer to the case where a program is executable and obtains non-empty results.

Question: list all 3 star rated thai restaurants		
Program Candidates	Gold	Exe
select restaurant *where* star_rating = thai	✗	✗
select restaurant *where* cuisine > 3	✗	✗
select restaurant *where* star_rating = 3	✗	✓
select restaurant *where* star_rating = 3 *and* cuisine = thai	✓	✓

Figure 5.1: Candidate programs for an utterance can be classified by executability (Exe); note that the gold program is always in the set of executable programs. We propose to ultilize the weak yet freely available signal of executablility for learning.

syntactic constraints, should also meet the semantic constraint of producing non-empty outputs. Previous work successfully incorporates syntactic constraints by relying on a type-based decoder (e.g., Krishnamurthy et al. (2017)), which only permits syntactically valid programs during generation. However, semantic constraints cannot be captured by typed-based decoders. For example, a type-based parser can generate a program "select restaurant where star cuisine = 3", which is syntactically meaningful as 'cuisine' can be compared with any string, including '3', in the condition clause. But this program will be treated as invalid if it does not output any restaurant. [3]

To learn from executability, we resort to marginal likelihood training, i.e., maximizing the marginal likelihood of all executable programs for an unlabeled NL utterance. However, the space of all possible programs is exponentially large, as well as the space of executable ones. Hence, simply marginalizing over all executable programs is intractable. Typical approximations use beam search to retrieve a handful of ('seen') programs, which are used to approximate the entire space. Using such approximations can lead to optimization getting trapped in undesirable local minima. For example, we observe that encouraging a model to exploit seen executable programs hinders exploration and reinforces the preference for shorter programs, as discussed in Section 5.4.3. This happens because shorter programs are both more likely to be among 'seen' programs (probably due to using locally-normalized autoregressive modeling) and more likely to be executable. To alleviate these issues, we derive three new alternative

[3] The executability assumption is strong, as a user might ask questions that have no answers. To relax this assumption, we can add a binary variable to indicate whether a program would generate non-empty output. The resulting learning problem is more challenging as more uncertainty needs to be incorporated.

objectives, relying on a new interpretation of marginal likelihood training from the perspective of posterior regularization. Our proposed objectives encode two kinds of inductive biases:

- **discouraging seen non-executable programs**, which plays a similar role to encouraging seen executable ones but does not share its drawback of hindering exploration;

- **encouraging sparsity** among executable programs, which encourages a parser to only focus on a subset of executable programs by softly injecting a sparsity constraint. This is desirable, as there are only one or few correct programs for each utterance (see Figure 5.1), and an accurate parser should assign probability mass only to this subset.

We collectively call these objectives X-PR, as a shorthand for Execution-guided Posterior Regularization.

We conduct experiments on two semantic parsing tasks: text-to-LF (logical form) parsing over a knowledge base and text-to-SQL (Zelle and Mooney, 1996) parsing over a relational database. Concretely, we evaluate our methods on the OVERNIGHT (Wang et al., 2015) and GEOQUERY (Zelle and Mooney, 1996) datasets. We simulate the semi-supervised learning setting by treating 70% of the training data as unlabeled. Empirical results show that our method can substantially boost the performance of a parser, trained only on labeled data, by utilizing a large amount of unlabeled data.

Contributions Following the same direction as the previous chapter, we continue to explore weaker forms of supervision. In this chapter, we look at a more extreme form of supervision – a large number of unlabeled examples that could be readily available in certain real-life scenarios. We approach the problem of learning from these unlabeled examples from the perspective of designing specialized training objectives:

- we show how to exploit unlabeled utterances by taking advantage of their executability, which admittedly is a weak but free learning signal;

- to better learn from executability, we propose a set of new objectives based on posterior regularization.

We empirically show that our method can help a base parser achieve substantially better performance by utilizing unlabeled data on OVERNIGHT and GEOQUERY.

5.1 Executability as Learning Signal

In this section, we formally define our semi-supervised learning setting and show how to incorporate executability into the training objective whilst relying on the marginal likelihood training framework. We also present two conventional approaches to optimizing marginal likelihood.

5.1.1 Problem Definition

Given a set of labeled NL-program pairs $\{(x_i^l, y_i^l)\}_{i=1}^N$ and a set of unlabeled NL utterances $\{x_j\}_{j=1}^M$, where N and M denote the sizes of the respective datasets, we would like to learn a neural parser $p(y|x,\boldsymbol{\theta})$, parameterized by $\boldsymbol{\theta}$, that maps utterances to programs. The objective to minimize consists of two parts:

$$\mathcal{J} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_{\text{sup}}(x_i^l, y_i^l) + \lambda\frac{1}{M}\sum_{j=1}^{M}\mathcal{L}_{\text{unsup}}(x_i) \tag{5.1}$$

where $\mathcal{L}_{sup}$ and $\mathcal{L}_{unsup}$ denote the supervised and unsupervised loss, respectively. For labeled data, we use the negative log-likelihood of gold programs; for unlabeled data, we instead use the log marginal likelihood (MML) of *all executable programs*. Specifically, they are defined as follows:

$$\mathcal{L}_{\text{sup}}(x,y) = -\log p(y|x,\boldsymbol{\theta}) \tag{5.2}$$

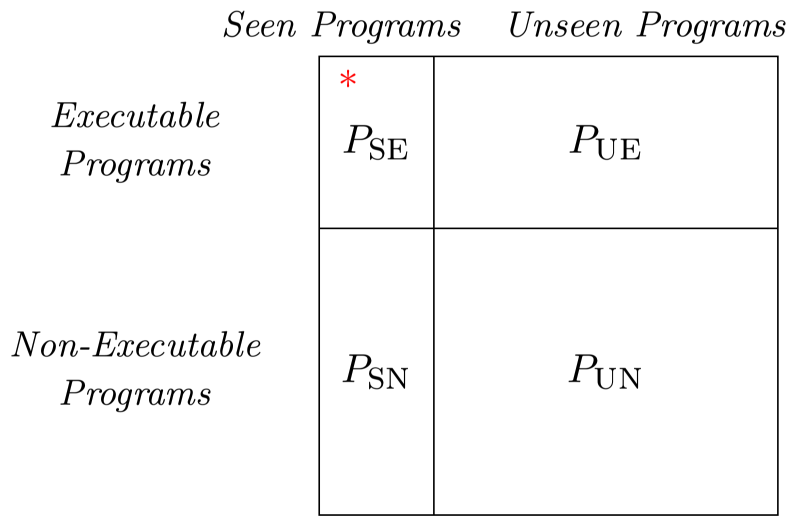
$$\mathcal{L}_{\text{unsup}}(x) = -\log\sum_{y} R(y)p(y|x,\boldsymbol{\theta}) \tag{5.3}$$

where $R(y)$ is a binary reward function that returns 1 if y is executable and 0 otherwise. In practice, this function is implemented by running a task-specific executor, e.g., a SQL executor.

Another alternative to unsupervised loss is REINFORCE (Sutton et al., 1999), i.e., maximize the expected $R(y)$ with respect to $p(y|x,\theta)$. However, as presented in Guu et al. (2017), this objective usually underperforms MML, which is consistent with our initial experiments.

5.1.2 Self-Training and Top-K MML

MML in Equation (5.3) requires marginalizing over all executable programs which is intractable. Conventionally, we resort to beam search to explore the space of programs and collect executable ones. To illustrate, we can divide the space of programs into four



(a) Partitioned program space. Red asterisk denotes the most probable executable program y^*. P stands for program; subscript S stands for seen, U for unseen, E for executable, and N for non-executable.

$$\mathcal{L}_{\text{ST}}(x,\boldsymbol{\theta}) = -\log p(y^*|x,\boldsymbol{\theta})$$

$$\mathcal{L}_{\text{top-k}}(x,\boldsymbol{\theta}) = -\log \sum_{y\in P_{\text{SE}}} p(y|x,\boldsymbol{\theta})$$

$$\mathcal{L}_{\text{repulsion}}(x,\boldsymbol{\theta}) = -\log\big(1-\sum_{y\in P_{\text{SN}}} p(y|x,\boldsymbol{\theta})\big)$$

$$\mathcal{L}_{\text{gentle}}(x,\boldsymbol{\theta}) = -p(P_{\text{SE}}\cup P_{\text{SN}})\log \sum_{y\in P_{\text{SE}}} p(y|x,\boldsymbol{\theta}) - p(P_{\text{UE}}\cup P_{\text{UN}})\log \sum_{y\in P_{\text{UE}}\cup P_{\text{UN}}} p(y|x,\boldsymbol{\theta})$$

$$\mathcal{L}_{\text{sparse}}(x,\boldsymbol{\theta}) = -\sum_{y\in P_{\text{SE}}} q_{\text{sparse}}(y)\log p(y|x,\boldsymbol{\theta})$$

(b) Five objectives to approximate MML.

Figure 5.2: In (a) the program space is partitioned along two dimentions: executability and observability. In (b) we show two commonly used objectives (Self-Training and Top-K MML) and the three objectives proposed in this chapter.

parts based on whether they are executable and observed, as shown in Figure 5.2a. For example, programs in $P_{\text{SE}} \cup P_{\text{SN}}$ are seen in the sense that they are retrieved by beam search. Programs in $P_{\text{SE}} \cup P_{\text{UE}}$ are all executable, though only programs in P_{SE} can be directly observed.

Two common approximations of Equation (5.3) are Self-Training (ST) and Top-K MML, and they are defined as follows:

$$\mathcal{L}_{\text{ST}}(x, \boldsymbol{\theta}) = -\log p(y^*|x, \boldsymbol{\theta}) \tag{5.4}$$

$$\mathcal{L}_{\text{top-k}}(x, \boldsymbol{\theta}) = -\log \sum_{y \in P_{\text{SE}}} p(y|x, \boldsymbol{\theta}) \tag{5.5}$$

where y^* denotes the most probable program, and it is approximated by the most probable one from beam search.

It is obvious that both methods only exploit programs in P_{SE}, i.e., executable programs retrieved by beam search. In cases where a parser successfully includes the correct programs in P_{SE}, both approximations should work reasonably well. However, if a parser is uncertain and P_{SE} does not contain the gold program, it would then mistakenly exploit incorrect programs in learning, which is problematic.

A naive solution to improve Self-Training or Top-K MML is to explore a larger space, e.g., increase the beam size to retrieve more executable programs. However, this would inevitably increase the computation cost of learning. Empirically, we find that increasing beam size, after it exceeds a certain threshold, is no longer beneficial for learning. In this chapter, we instead propose better approximations without increasing beam size.

5.2 Method

We first present a view of MML in Equation (5.3) from the perspective of posterior regularization. This new perspective helps us derive three alternative approximations of MML: Repulsion MML, Gentle MML, and Sparse MML.

5.2.1 Posterior Regularization

Posterior regularization (PR) allows to inject linear constraints into posterior distributions of generative models, and it can be extended to discriminative models (Ganchev et al., 2010). In our case, we try to constrain the parser $p(y|x, \theta)$ to only assign probability mass to executable programs. Instead of imposing hard constraints, we softly

penalize the parser if it is far away from a desired distribution $q(y)$, which is defined as $\mathbb{E}_q[R(y)] = 1$. Since R is a binary reward function, $q(y)$ is constrained to only place mass on executable programs whose rewards are 1. We denote all such desired distributions as the family Q.

Specifically, the objective of PR is to penalize the KL-divergence between Q and p, which is:

$$\begin{aligned} \mathcal{J}_Q(\boldsymbol{\theta}) &= D_{\mathrm{KL}}[Q||p(y|x,\boldsymbol{\theta})] \\ &= \min_{q\in Q} D_{\mathrm{KL}}[q(y)||p(y|x,\boldsymbol{\theta})] \end{aligned} \tag{5.6}$$

By definition, the objective has the following upper bound:

$$\begin{aligned} \mathcal{J}(\boldsymbol{\theta}, q) &= D_{\mathrm{KL}}[q(y)||p(y|x,\boldsymbol{\theta})] \\ &= -\sum_y q(y)\log p(y|x,\boldsymbol{\theta}) - \mathcal{H}(q) \end{aligned} \tag{5.7}$$

where $q \in Q$, $\mathcal{H}$ denotes the entropy. We can use block-coordinate descent, an EM iterative algorithm to optimize it.

$$\begin{aligned} \mathrm{E}: q^{t+1} &= \arg\min_{q\in Q} D_{\mathrm{KL}}[q(y)||p(y|x,\boldsymbol{\theta}^t)] \\ \mathrm{M}: \boldsymbol{\theta}^{t+1} &= \arg\min_{\boldsymbol{\theta}} -\sum_y q^{t+1}(y)[\log p(y|x,\boldsymbol{\theta})] \end{aligned}$$

During the E-step, we try to find a distribution q from the constrained set Q that is closest to the current parser p in terms of KL-divergence. We then use q as a 'soft label' and minimize the cross-entropy between q and p during the M-step. Note that q is a constant vector and has no gradient wrt. $\boldsymbol{\theta}$ during the M-step.

The E-step has a closed-form solution:

$$q^{t+1}(y) = \begin{cases} \frac{p(y|x,\boldsymbol{\theta}^t)}{p(P_{\mathrm{SE}}\cup P_{\mathrm{UE}})} & y \in P_{\mathrm{SE}} \cup P_{\mathrm{UE}} \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

where $p(P_{\mathrm{SE}} \cup P_{\mathrm{UE}}) = \sum_{y'\in P_{\mathrm{SE}}\cup P_{\mathrm{UE}}} p(y'|x,\boldsymbol{\theta}^t)$. $q^{t+1}(y)$ is essentially a re-normalized version of p over executable programs. Interestingly, if we use the solution in the M-step, the gradient wrt. $\boldsymbol{\theta}$ is equivalent to the gradient of MML in Equation (5.3). That is, optimizing PR with the EM algorithm is equivalent to optimizing MML. See the proof blow.

Proof. Since $q(y)$ is 0 for non-executable programs, we only need to compute it for

executable programs in $\mathcal{V}$. We need to solve the following optimization problem:

$$\begin{aligned} \min_{q} \; & -\sum_{y\in\mathcal{V}} q(y)\log p(y|x,\boldsymbol{\theta}) + \sum_{y\in\mathcal{V}} q(y)\log q(y) \\ s.t. \quad & \sum_{y\in\mathcal{V}} q(y) = 1 \\ & q(y) \geq 0 \end{aligned} \tag{5.9}$$

By solving it with KKT conditions, we can see that $q(y) \propto p(y|x,\boldsymbol{\theta})$. Since $q(y)$ needs to sum up to 1, it is easy to obtain that $q(y) = \frac{p(y|x,\boldsymbol{\theta})}{\sum_{y\in\mathcal{V}} p(y|x,\boldsymbol{\theta})}$. □

The connection between EM and MML is not new, and it has been well-studied for classification problems (Amini and Gallinari, 2002; Grandvalet and Bengio, 2005). In our problem, we additionally introduce PR to accommodate the executability constraint, and instantiate the general EM algorithm.

Although the E-step has a closed-form solution, computing q is still intractable due to the large search space of executable programs. However, this PR view provides new insight on what it means to approximate MML. In essence, conventional methods can be viewed as computing an approximate solution of q. Specifically, Self-Training corresponds to a delta distribution that only focuses on the most probable y^*.

$$q_{\text{ST}}^{t+1}(y) = \begin{cases} 1 & y = y^* \\ 0 & \text{otherwise} \end{cases} \tag{5.10}$$

Top-K MML corresponds to a re-normalized distribution over P_{SE}.

$$q_{\text{top-k}}^{t+1}(y) = \begin{cases} \frac{p(y|x,\boldsymbol{\theta}^t)}{p(P_{\text{SE}})} & y \in P_{\text{SE}} \\ 0 & \text{otherwise} \end{cases} \tag{5.11}$$

Most importantly, this perspective leads us to deriving three new approximations of MML, which we collectively call X-PR.

5.2.2 Repulsion MML and Gentle MML

As mentioned previously, Self-Training and Top-K MML should be reasonable approximations in cases where gold programs are retrieved, i.e., they are in the seen executable subset (P_{SE} in Figure 5.2a). However, if a parser is uncertain, i.e., beam search cannot retrieve the gold programs, exclusively exploiting P_{SE} programs is undesirable. Hence, we consider ways of taking unseen executable programs (P_{UE} in Figure 5.2a) into account. Since we never directly observe unseen programs (P_{UE} or P_{UN}), our heuristics do

not discriminate between executable and non-executable programs ($P_{\text{UE}} \cup P_{\text{UN}}$). In other words, upweighting P_{UE} programs will inevitably upweight P_{UN}.

Based on the intuition that the correct program is included in either seen executable programs (P_{SE}) or unseen programs (P_{UE} and P_{UN}), we can simply push a parser away from seen non-executable programs (P_{SN}). Hence, we call such method Repulsion MML. Specifically, the first heuristic approximates Equation (5.8) as follows:

$$q^{t+1}_{\text{repulsion}}(y) = \begin{cases} \frac{p(y|x,\boldsymbol{\theta}^t)}{1-p(P_{\text{SN}})} & y \notin P_{\text{SN}} \\ 0 & \text{otherwise} \end{cases} \tag{5.12}$$

Another way to view this heuristic is that we distribute the probability mass from seen non-executable programs (P_{SN}) to other programs. In contrast, the second heuristic is more 'conservative' about unseen programs as it tends to trust seen executable P_{SN} programs more. Specifically, the second heuristic uses the following approximations to solve the E-step.

$$q^{t+1}_{\text{gentle}}(y) = \begin{cases} \frac{p(P_{\text{SE}\cup\text{SN}})}{p(P_{\text{SE}})} p(y|x,\boldsymbol{\theta}^t) & y \in P_{\text{SE}} \\ p(y|x,\boldsymbol{\theta}^t) & y \in P_{\text{UE}} \cup P_{\text{UN}} \\ 0 & y \in P_{\text{SN}} \end{cases} \tag{5.13}$$

Intuitively, it shifts the probability mass of seen non-executable programs (P_{SN}) directly to seen executable programs (P_{SE}). Meanwhile, it neither upweights nor downweights unseen programs. We call this heuristic Gentle MML. Compared with Self-Training and Top-K MML, Repulsion MML and Gentle MML lead to better exploration of the program space, as only seen non-executable (P_{SN}) programs are discouraged.

5.2.3 Sparse MML

Sparse MML is based on the intuition that in most cases there is only one or few correct programs among all executable programs. As mentioned in Section 5.5, spurious programs that are executable, but do not reflect the semantics of an utterance are harmful. One empirical evidence from previous work (Min et al., 2019b) is that Self-Training outperforms Top-K MML for weakly-supervised question answering. Hence, exploiting all seen executable programs can be sub-optimal. Following recent work on sparse distributions (Martins and Astudillo, 2016; Niculae et al., 2018), we propose to encourage sparsity of the 'soft label' q. Encouraging sparsity is also related to the minimum entropy and low-density separation principles which are commonly used in semi-supervised learning (Grandvalet and Bengio, 2005; Chapelle and Zien, 2005).

To achieve this, we first interpret the entropy term $\mathcal{H}$ in Equation (5.7) as a regularization of q. It is known that entropy regularization always results in a dense q, i.e., all executable programs are assigned non-zero probability. Inspired by SparseMax (Martins and Astudillo, 2016), we instead use L2 norm for regularization. Specifically, we replace our PR objective in Equation (5.7) with the following one:

$$\mathcal{J}_{\text{sparse}}(\boldsymbol{\theta}, q) = -\sum_{y} q(y) \log p(y|s, \boldsymbol{\theta}) + \frac{1}{2}||q||_2^2 \tag{5.14}$$

where $q \in Q$. Similarly, it can be optimized by the EM algorithm:

$$\text{E}: q^{t+1} = \text{SparseMax}_Q(\log p(y|x, \boldsymbol{\theta}^t)) \tag{5.15}$$

$$\text{M}: \boldsymbol{\theta}^{t+1} = \arg\min_{\boldsymbol{\theta}} -\sum_{y} q^{t+1}(y)[\log p(y|x, \boldsymbol{\theta})] \tag{5.16}$$

where the top-E-step can be solved by the SparseMax operator, which denotes the Euclidean projection from the vector of logits $\log p(y|x, \boldsymbol{\theta}^t)$ to the simplex Q. Again, we solve the E-step approximately. One of the approximations is to use top-k SparseMax which constrain the number of non-zeros of q to be less than k. It can be solved by using a top-k operator and followed by SparseMax (Correia et al., 2020). In our case, we use beam search to approximate the top-k operator and the resulting approximation for the E-step is defined as follows:

$$q^{t+1}_{\text{sparse}} = \text{SparseMax}_{y \in P_{\text{SE}}}\left(\log p(y|x, \boldsymbol{\theta}^t)\right) \tag{5.17}$$

Intuitively, q^{t+1}_{sparse} occupies the middle ground between Self-Training (uses y^* only) and Top-K MML (uses all P_{SE} programs). With the help of sparsity of q introduced by SparseMax, the M-step will only promote a subset of P_{SE} programs.

To summarize, we propose three new approximations of MML for learning from executions. They are designed to complement Self-Training and Top- K MML via discouraging seen non-executable programs and introducing sparsity. In the following sections, we will empirically show that they are superior to Self-Training and Top-K MML for semi-supervised semantic parsing. The approximations we proposed may also be beneficial for learning from denotations in Chapter 4, and weakly supervised question answering (Min et al., 2019b), but we leave this to future work.

5.3 Semantic Parsers

In principle, our X-PR framework is model-agnostic, i.e., it can be coupled with any semantic parser for semi-supervised learning. In this chapter, we reuse RAT-SQL from

Type of x	Type of y	Edge label	Description
Entity	Entity	RELATED-F	there exists a property p s.t. $(x,p,y) \in \mathcal{K}$
		RELATED-R	there exists a property p s.t. $(y,p,x) \in \mathcal{K}$
Entity	Property	HAS-PROPERTY-F	there exists an entity e s.t. $(x,y,e) \in \mathcal{K}$
		HAS-PROPERTY-R	there exists an entity e s.t. $(e,y,x) \in \mathcal{K}$
Property	Entity	PROP-TO-ENT-F	there exists an entity e s.t. $(y,x,e) \in \mathcal{K}$
		PROP-TO-ENT-R	there exists an entity e s.t. $(e,x,y) \in \mathcal{K}$
Utterance Token	Entity	EXACT-MATCH	x and y are the same word
		PARTIAL-MATCH	token x is contained in entity y
Entity	Utterance Token	EXACT-MATCH-R	y and x are the same word
		PARTIAL-MATCH-R	token y is contained in entity x
Utterance Token	Property	P-EXACT-MATCH	x and y are the same word
		P-PARTIAL-MATCH	token x is contained in property y
Property	Utterance Token	P-EXACT-MATCH-R	y and x are the same word
		P-PARTIAL-MATCH-R	token y is contained in property y

Table 5.1: Relation types used for text-to-LF parsing. Suffix 'F' and 'R' denote forward and reverse, respectively, to differentiate the directionality of the relation.

Chapter 3. We generalize the original RAT-SQL so that it can handle both text-to-LF as well as text-to-SQL parsing. Central to the generalization is to generalize schema encoding and schema linking in the context of text-to-SQL parsing to the following notions:

- *enviroment encoding*: encoding enviroments, i.e., a knowledge base consisting of a set of triples; a relational database represented by its schema
- *enviroment linking*: linking mentions to intended elements of environments, i.e., mentions of entities and properties of knowledge bases; mentions of tables and columns of relational databases

Under this generalization, RAT-SQL introduced in Chapter 3 can be treated as a general framework to handle enviroment encoding and linking. To adapt the framework to a particular task, we only need to specify the task-dependent relations for enviroment encoding and linking. Specifically, the relations used for text-to-LF are shown in Table 5.1. For text-to-SQL parsing, we reuse the relations defined in Chapter 3. The successful generalization of RAT-SQL to text-to-LF, as we will show during the experiments, fullfils our promise in Chapter 3 that RAT-SQL is indeed a general framework instead of being specific to text-to-SQL parsing.

5.4 Experiments

To evaluate X-PR, we present experiments on semi-supervised semantic parsing. We also analyze how the objectives affect the training process.

5.4.1 Semi-Supervised Learning Setting

We simulate the setting of semi-supervised learning on standard text-to-LF and text-to-SQL parsing benchmarks. Specifically, we randomly sample 30% of the original training data as the labeled data, and use the rest 70% as the unlabeled data. For text-to-LF parsing, we use the OVERNIGHT dataset (Wang et al., 2015), which has eight different domains, each with a different size ranging between 801 and 4,419; for text-to-SQL parsing, we use GEOQUERY (Zelle and Mooney, 1996) which contains 880 utterance-SQL pairs. The semi-supervised setting is very challenging as leveraging only 30% of the original training data would result in only around 300 labeled examples in four domains of OVERNIGHT and also in GEOQUERY.

Supervised Lower and Upper Bounds As baselines, we train two supervised models. The first one only uses the labeled data (30% of the original training data) and discards the unlabeled data in the semi-supervised setting. We view this baseline as a *lower bound* in the sense that any semi-supervised method is expected to surpass this. The second one has extra access to gold programs for the unlabeled data in the semi-supervised setting, which means it uses the full original training data. We view this baseline as an *upper bound* for semi-supervised learning; we cannot expect to approach it as the executability signal is much weaker than direct supervision. Our main experiments aim to show how the proposed objectives can mitigate the gap between the lower- and upper-bound baselines by utilizing 70% unlabeled data.

By comparing the performance of the second baseline (upper bound) with previous methods (Jia and Liang, 2016; Herzig and Berant, 2017; Su and Yan, 2017), we can verify that our semantic parsers are state-of-the-art. See Table 5.2 for detailed comparisons.

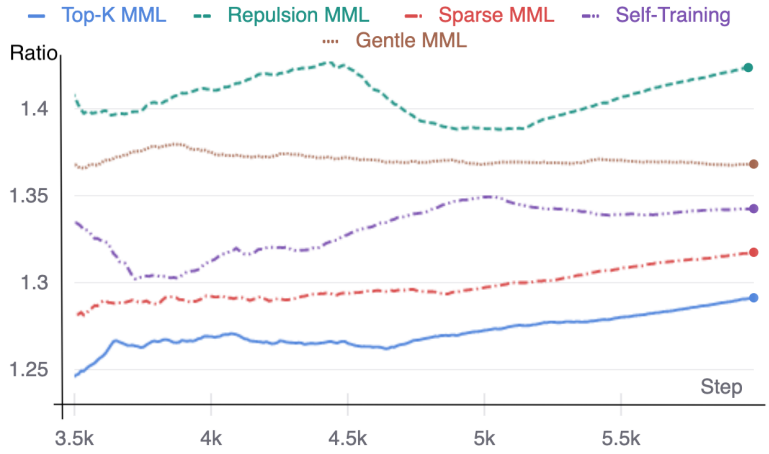
Semi-Supervised Training and Tuning We use stochastic gradient descent to optimize Equation (5.1). At each training step, we sample two batches from the labeled and unlabeled data, respectively. In preliminary experiments, we found that it is crucial to pre-train a parser on supervised data alone; this is not surprising as all of the objectives

Model	BASKETBALL	BLOCKS	CALENDAR	HOUSING	PUBLICATIONS	RECIPES	RESTAURANTS	SOCIAL	Avg.
Wang et al. (2015)	46.3	41.9	74.4	54.5	59.0	70.8	75.9	48.2	58.8
Jia and Liang (2016)	85.2	58.1	78.0	71.4	76.4	79.6	76.2	81.4	75.8
Herzig and Berant (2017)	85.2	61.2	77.4	67.7	74.5	79.2	79.5	80.2	75.6
Su and Yan (2017)	86.2	60.2	79.8	71.4	78.9	84.7	81.6	82.9	78.2
Ours	87.7	62.9	82.1	71.4	78.9	82.4	82.8	80.8	**78.6**
Herzig and Berant (2017)*	86.2	62.7	82.1	78.3	80.7	82.9	82.2	81.7	79.6
Su and Yan (2017)*	88.2	62.7	82.7	78.8	80.7	86.1	83.7	83.1	80.8

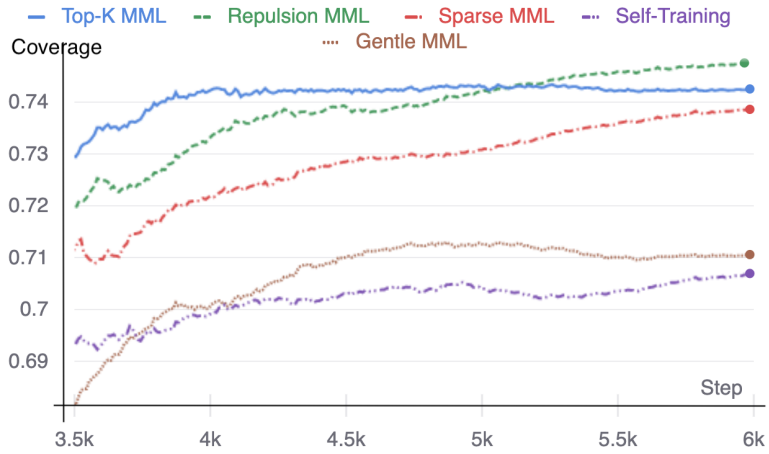
Table 5.2: Test accuracy of supervised models on all domains for OVERNIGHT. Models with * are augmented with cross-domain training.

Model	OVERNIGHT									GEO
	BASKETBALL	BLOCKS	CALENDAR	HOUSING	PUBLICATIONS	RECIPES	RESTAURANTS	SOCIAL	Avg.	
Lower bound	82.2	54.1	64.9	61.4	64.3	72.7	71.7	76.7	68.5	60.6
Self-Training	84.7	52.6	67.9	59.8	68.6	80.1	71.1	77.4	70.3	64.2
Top-K MML	83.1	55.3	68.5	56.9	67.7	73.7	69.9	76.2	68.9	61.3
Repulsion MML	**84.9**	56.3	70.8	60.9	70.3	79.8	72.0	**78.3**	71.7	64.9
Gentle MML	84.1	58.1	70.2	**63.0**	71.5	78.7	72.3	76.4	71.8	65.6
Sparse MML	83.9	**58.6**	**72.6**	60.3	**75.2**	**80.6**	**72.6**	77.8	**72.7**	**67.4**
Upper Bound	87.7	62.9	82.1	71.4	78.9	82.4	82.8	80.8	78.6	74.2

Table 5.3: Execution accuracy of supervised and semi-supervised models on all domains of OVERNIGHT and GEOQUERY. In semi-supervised learning, 30% of the original training examples are treated as labeled and the remaining 70% as unlabeled. Lower bound refers to supervised models that only use labeled examples and discard unlabeled ones whereas upper bound refers to supervised models that have access to gold programs of unlabeled examples. Avg. refers to the average accuracy of the eight OVERNIGHT domains. We average runs over three random splits of the original training data for semi-supervised learning. In average accuracy, the improvement from Sparse MML compared with Self-Training is statistically significant ($p \ll 0.01$, paired permutation test).



(a) Average Ratios.



(b) Coverage of gold programs.

Figure 5.3: Effect of different learning objectives in terms of average ratios and coverage (view in color).

for learning from execution rely on beam search which would only introduce noise with an untrained parser. That is, λ in Equation (5.1) is set to 0 during initial updates, and is switched to a normal value afterwards.

We leave out 100 labeled examples for tuning the hyperparameters. The hyperparameters of the semantic parser are only tuned for the development of the supervised baselines, and are fixed for semi-supervised learning. The only hyperparameter we tune in the semi-supervised setting is the λ in Equation (5.1), which controls how much the unsupervised objective influences learning. After tuning, we use all the labeled examples for supervised training and use the last checkpoints for evaluation on the test set.

5.4.2 Main Results

Our experiments evaluate the objectives presented in Figure 5.2 under a semi-supervised learning setting. Our results are shown in Table 5.3.

Self-Training and Top-K MML First, Top-K MML, which exploits more executable programs than Self-Training, does not yield better performance in six domains of OVERNIGHT and GEOQUERY. This observation is consistent with Min et al. (2019b) where Top-K MML underperforms Self-Training for weakly-supervised question answering. Self-Training outperforms the lower bound in five domains of OVERNIGHT, and on average. In contrast, Top-K MML obtains a similar performance to the lower bound in terms of average accuracy.

X-PR Objectives In each domain of OVERNIGHT and GEOQUERY, the objective that achieves the best performance is always within X-PR. In terms of average accuracy in OVERNIGHT, all our objectives perform better than Self-Training and Top-K MML. Among X-PR, Sparse MML performs best in five domains of OVERNIGHT, leading to a margin of 4.2% compared with the lower bound in terms of average accuracy. In GEOQUERY, Sparse MML also obtain best performance.

Repulsion MML, which is based on the same intuition of discouraging seen non-executable programs, achieves a similar average accuracy to Gentle MML in OVERNIGHT. In contrast, Gentle MML tends to perform better in domains whose parser are weak (such as HOUSING, BLOCKS) indicated by their lower bounds. In GEOQUERY, Gentle MML performs slightly better than Repulsion MML. Although it does not perform better than Repulsion MML, it retrieves more accurate programs and also generates longer programs (see next section for details).

To see how much labeled data would be needed for a supervised model to reach the same accuracy as our semi-supervised models, we conduct experiments using 40% of the original training examples as the labeled data. The supervised model achieves 72.6% on average in OVERNIGHT, implying that 'labeling' 33.3% more examples would yield the same accuracy as our best-performning objective (Sparse MML).

5.4.3 Analysis

To better understand the effect of different objectives, we conduct analysis on the training process of semi-supervised learning. For the sake of brevity, we focus our

analysis on the CALENDAR domain but have drawn similar conclusions for the other domains.

Length Ratio During preliminary experiments, we found that all training objectives tend to favor short executable programs for unlabeled utterances. To quantify this, we define the metric of average ratio as follows:

$$ratio = \frac{\sum_i \sum_{y \in P_{\text{SE}}(x_i)} |y|}{\sum_i |x_i| |P_{\text{SE}}(x_i)|} \tag{5.18}$$

where $P_{\text{SE}}(x_i)$ denotes seen executable programs of x_i, $|x|, |y|$ denotes the length of an utterance and a program, respectively, and $|P_{\text{SE}}(x_i)|$ denotes the number of seen executable programs. Intuitively, average ratio reveals the range of programs that an objective is exploiting in terms of length. This metric is computed in an online manner, and x_i is a sequence of data fed to the training process.

As shown in Figure 5.3a, Top-K MML favors shorter programs, especially during the initial steps. In contrast, Repulsion MML and Gentle MML prefer longer programs. For reference, we can compute the gold ratio by assuming $P_{\text{SE}}(x_i)$ only contains the gold program. The gold ratio for CALENDAR is 2.01, indicating that all objectives are still preferring programs that are shorter than gold programs. However, by not directly exploiting seen executable programs, Repulsion MML and Gentle MML alleviate this issue compared with Top-K MML.

Coverage Next, we analyze how much an objective can help a parser retrieve gold programs for unlabeled data. Since the orignal data contains the gold programs for the unlabeled data, we ultilize them to define the metric of coverage as follows:

$$coverage = \frac{\sum_i I[\hat{y}_i \in P_{\text{SE}}(x_i)]}{\sum_i |x_i|} \tag{5.19}$$

where I is an indicator function, $\hat{y}_i$ denotes the gold program of an utterance x_i. Intuitively, this metric measures how often a gold program is captured in P_{SE}. As shown in Figure 5.3b, Self-Training, which only exploits one program at a time, is relatively weak in terms of retrieving more gold programs. In contrast, Repulsion MML retrieves more gold programs than the others.

As mentioned in Section 5.2.3, SparseMax can be viewed as an interpolation between Self-Training and Top-K MML. This is also reflected in both metrics: Sparse MML always occupies the middle-ground performance between ST and Top-K MML. Interestingly, although Sparse MML is not best in terms of both diagnostic metrics, it still achieves the best accuracy in this domain.

5.5 Related Work

Semi-Supervised Semantic Parsing In the context of semantic parsing, semi-supervised models using limited amounts of parallel data and large amounts of unlabeled data treat either utterances or programs as discrete latent variables and induce them in the framework of generative models (Kočiský et al., 2016; Yin et al., 2018). A challenge with these methods is that (combinatorially) complex discrete variables make optimization very hard, even with the help of variational inference. In this chapter, we seek to directly constrain the discriminative parser with signals obtained from executions. Our method can potentially be integrated into these generative models to regularize discrete variables.

(Underspecified) Sequence-Level Rewards There have been attempts in recent years to integrate sequence-level rewards into sequence-to-sequence training as a way of accommodating task-specific objectives. For example, BLEU can be optimized for coherent text generation (Bosselut et al., 2018) and machine translation (Wu et al., 2018) via reinforcement learning or beam-search (Wiseman and Rush, 2016). In this chapter, we resort to marginal likelihood training to exploit binary executability rewards for semantic parsing (i.e., whether a program is executable or not), which has been shown to be more effective than REINFORCE (Guu et al., 2017).

More importantly, our binary reward is underspecified, i.e., there exist many spurious programs that enjoy the same reward as the gold program. This issue of learning from underspecified rewards underlies many weakly-supervised tasks, e.g., learning from denotations (Liang et al., 2013; Berant et al., 2013), weakly supervised question answering (Min et al., 2019b).

Execution for Semantic Parsing Execution has been utilized in semantic parsing (Wang et al., 2018) and the related area of program synthesis (Chen et al., 2018). These approaches exploit the execution of partial programs to guide the search for plausible complete programs. Although partial execution is feasible for SQL-style programs, it cannot be trivially extended to general meaning representation (e.g., logical forms). In this chapter, we explore a more general setting where execution can be only obtained from complete programs.

5.6 Summary

To take adavantage of a large amount of unlabeled utterances in the semi-supervised setting, we propose to learn a semantic parser from the weak yet freely available executability signals. Due to the large search space of executable programs, conventional approximations of MML training, i.e, Self-Training and Top-K MML, are often sub-optimal. We propose a set of alternative objectives, namely X-PR, through the lens of posterior regularization. Empirical results on semi-supervised learning show that X-PR can help a parser achieve substantially better performance than conventional methods, further bridging the gap between semi-supervised learning and supervised learning. The success signifies that we can indeed reduce human annotation effort by exploiting a large amount of unlabeled utterances if they are readily available.

In this and the previous chapter, we explore the direction of utilizing weak yet cheap (or even free) learning signals in the form of denotations or unlabeled examples, to ultimately reduce the annotation effort for building semantic parsers. from the methodology perspective, we present a two-stage latent-alignment parser and specialized training objectives, respectively, to effectively take advantage of these weak learning signals. Note that these two methodologies are orthogonal, and in principle can be combined for better generalization. For example, we can train a latent-alignment parser with the specialized training objectives for learning from unlabeled examples. In the next chapter, we will switch to the setting where we have labeled examples (i.e., annotated programs paired with utterances), and ask the question that if we can afford to annotate examples, (to what extent) will the resulting parser trained on them cover the space of all possible utterances?

Chapter 6

Systematic Generalization

Previous chapters explore settings that deviate from the conventional supervised in-domain setting, including cross-domain, weakly-supervised, or semi-supervised settings. In this chapter, we revisit the conventional setting to investigate the more fundamental question of linguistic coverage, i.e., to what extent can a semantic parser cover all the possible meanings of user utterances? Typically, this question is trivial if we have access to the distribution of real-life user utterances, e.g., relying on real-life users for collecting data, since in this case we can evaluate linguistic coverage by assessing our parsers on random samples from the distribution. In practice, however, it is very likely that we cannot get access to real-life users, especially in cases where a semantic parser is developed on a new domain for a new application. To handle this, we consider relaxing the assumption of knowing the distribution of user utterance by the following ones: 1) meanings are compositional in that a complex meaning is usually composed of one or multiple sub-meanings; 2) the training data contains data that cover all the possible sub-meanings, but not all the ways of combining them. Under these assumptions, we investigate a concrete question to probe linguistic coverage: if the training data cover all basic sub-meanings, to what extent can the resulting trained parser cover all possible combinations of these sub-meanings. In a rather abstract sense, instead of assuming access to the distribution of meanings directly, we know the distribution of basic sub-meanings in the absence of the distribution of the ways that combine these sub-meanings. To get an intuition, consider examples shown in Figure 6.1. During training, we assume that a parser knows how to process meanings regarding length and longest river. We hope it can generalize to a new meaning that involves both length and longest river.

Apart from the motivation on assessing linguistic coverage, the problem under

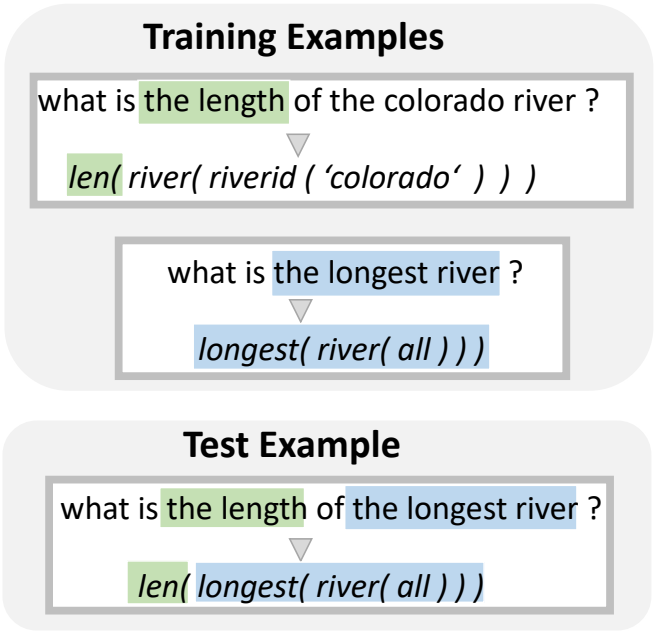


Figure 6.1: A semantic parser needs to generalize to test examples which contain segments from multiple training examples (shown in green and blue).

the relaxed assumptions also has other motivations from the machine learning and cognitive science perspective. Humans can easily understand novel combinations of sub-meanings if exposed to basic sub-meanings before, e.g., it would be easy for a human to address the test example in Figure 6.1. However, unlike humans, conventional sequence-to-sequence (seq2seq) models which are widely used in NLP and semantic parsing, fail to generalize *systematically* (Bahdanau et al., 2019; Lake and Baroni, 2018; Loula et al., 2018), i.e., correctly interpret sentences representing novel combinations of concepts seen in training. From the machine learning standpoint, this problem can be framed as an out-of-distribution generalization problem: how can a parser systematically generalize in the setting where the distribution on combinations of sub-meanings of test data is divergent from that of training data. In the rest of the chapter, we will focus on improving systematic generalization of a parser with the aim at improving linguistic coverage.

Our goal is to provide a mechanism for encouraging systematic generalization in seq2seq models. To get an intuition of our methodology, we consider again the examples shown Figure 6.1. To process the test utterance, the learner needs to first decompose it into two segments previously observed in training (shown in green and blue), and then combine their corresponding program fragments to create a new program. Current seq2seq models fail in this systematic generalization setting Finn et al.

(2017); Keysers et al. (2019). In contrast, traditional grammar formalisms decompose correspondences between utterances and programs into compositional mappings of substructures (Steedman, 2000), enabling grammar-based parsers to recombine rules acquired during training, as needed for systematic generalization. Grammars have proven essential in statistical semantic parsing in the pre-neural era Zettlemoyer and Collins (2012); Wong and Mooney (2006), and have gained renewed interest now as a means of achieving systematic generalization Herzig and Berant (2021); Shaw et al. (2020). However, grammars are hard to create and maintain (e.g., requiring grammar engineering or grammar induction stages) and do not scale well to NLP problems beyond semantic parsing (e.g., machine translation). In this chapter, we argue that the key property of grammar-based models, giving rise to their improved ood performance, is that a grammar implicitly encodes alignments between input and output segments. For example, in Figure 6.1, the expected segment-level alignments are '`the length` $\rightarrow$ `len`' and '`the longest river` $\rightarrow$ `longest(river(all))`'. Instead of developing a full-fledged grammar-based method, we directly model segment-level alignments as structured latent variables. The resulting alignment-driven seq2seq model remains end-to-end differentiable, and, in principle, applicable to any sequence transduction problem.

Segment-Level Alignments Modeling segment-level alignments requires simultaneously inducing a segmentation of input and output sequences and discovering correspondences between the input and output segments. While segment-level alignments have been previously incorporated in neural models (Yu et al., 2016; Wang et al., 2017b), to maintain tractability, these approaches support only monotonic alignments. The monotonicity assumption is reasonable for certain tasks (e.g., summarization), but it is generally overly restrictive (e.g., consider semantic parsing and machine translation). To relax this assumption, we complement monotonic alignments with an extra reordering step. That is, we first permute the source sequence so that segments within the reordered sequence can be aligned monotonically to segments of the target sequence. Coupling *latent permutations* with monotonic alignments dramatically increases the space of admissible segment alignments.

The space of general permutations is exceedingly large, so, to allow for efficient training, we restrict ourselves to *separable* permutations (Bose et al., 1998). We model separable permutations as hierarchical reordering of segments using *permutation trees*. This hierarchical way of modeling permutations reflects the hierarchical nature of

language and hence is arguably more appropriate than 'flat' alternatives (Mena et al., 2018). Interestingly, recent studies (Steedman, 2021; Stanojević and Steedman, 2018) demonstrated that separable permutations are sufficient for capturing the variability of permutations in linguistic constructions across natural languages, providing further motivation for our modeling choice.

Simply marginalizing over all possible separable permutations remains intractable. Instead, inspired by recent work on modeling latent discrete structures (Corro and Titov, 2019; Fu et al., 2020), we introduce a continuous relaxation of the reordering problem. The key ingredients of the relaxation are two inference strategies: *marginal inference*, which yields the expected permutation under a distribution; *MAP inference*, which returns the most probable permutation. In this chapter, we propose efficient dynamic programming algorithms to perform *exact* marginal and MAP inference with separable permutations, resulting in effective differentiable neural modules producing relaxed separable permutations. By plugging this module into an existing module supporting monotonic segment alignments (Yu et al., 2016), we obtain an end-to-end differentiable seq2seq model, supporting non-monotonic segment-level alignments.

Contributions In this chapter, we revisit the conventional supervised in-domain setting, to investigate ways of improving the linguistic coverage of a semantic parser. We approach the problem from the perspective of incorporating structured inductive biases into a parser:

- we propose a seq2seq model for semantic parsing that accounts for latent non-monotonic segment-level alignments;
- we design novel and efficient algorithms for exact marginal and MAP inference with separable permutations, allowing for end-to-end training using a continuous relaxation.

This model is general and applicable to other seq2seq NLP problems such as machine translation. Apart from experiments on semantic parsing, we also show its effectiveness on machine translation.

In favor of a general NLP model, the framework developed in this chapter does not follow the framework of semantic parsers discussed in Chapter 2, e.g., it does not take the grammar of target programs into account for semantic parsing. We leave the direction of exploring structured latent alignments that accommodate program grammars as future work.

As an orthogonal direction to upgrading model architectures for systematic gener-

alization, we also explore how to upgrade the training algorithm of standard seq2seq models. Analogous to DG-MAML algorithm presented in Chapter 3 for domain generalization, we recently find in Conklin et al. (2021) that meta-learning can also be effectively employed to boost systematic generalization. As we mentioned in Section 1.3.3, injecting model and learning bias are two kinds of methodologies we use for addressing generalization challenges of semantic parsing, and this work Conklin et al. (2021) makes the exploration in this regime complete.

6.1 Background and Related Work

6.1.1 Systematic Generalization

Human learners exhibit systematic generalization, which refers to their ability to generalize from training data to novel situations. This is possible due to the *compositionality* of natural languages - to a large degree, sentences are built using an inventory of primitive concepts and finite structure-building mechanisms (Chomsky, 1965). For example, if one understands 'John loves the girl', they should also understand 'The girl loves John' (Fodor and Pylyshyn, 1988). This is done by 'knowing' the meaning of individual words and the grammatical principle of subject-verb-object composition. As pointed out by Goodwin et al. (2020), systematicity entails that primitive units have consistent meaning across different contexts. In contrast, in seq2seq models, the representations of a word are highly influenced by context (see experiments in Lake and Baroni (2018)). This is also consistent with the observation that seq2seq models tend to memorize large chunks rather than discover underlying compositional principles (Hupkes et al., 2019). The memorization of large sequences lets the model fit the training distribution but harms out-of-distribution generalization.

6.1.2 Discrete Alignments as Conditional Computation Graphs

Latent discrete structures enable the incorporation of inductive biases into neural models and have been beneficial for a range of problems. For example, input-dependent module layouts (Andreas et al., 2016) or graphs (Norcliffe-Brown et al., 2018) have been explored in visual question answering. There is also a large body of work on inducing task-specific discrete representations (usually trees) for NL sentences (Yogatama et al., 2016; Niculae et al., 2018; Havrylov et al., 2019; Corro and Titov, 2019). The trees are induced simultaneously with learning a model performing a computation relying on the

tree (typically a recursive neural network (Socher et al., 2011)), while optimizing a task-specific loss. Given the role the structures play in these approaches – i.e., defining the computation flow – we can think of the structures as *conditional computation graphs*.

In this chapter, we induce discrete alignments as conditional computation graphs to guide seq2seq models. Given a source sequence x with n tokens and a target sequence y with m tokens, we optimize the following objective:

$$\boldsymbol{X} = \text{Encode}_\theta(x) \qquad \mathcal{L}_{\theta,\phi}(x,y) = -\log \mathbb{E}_{p_\phi(\boldsymbol{M}|\boldsymbol{X})} p_\theta(y|\boldsymbol{X},\boldsymbol{M}) \tag{6.1}$$

where Encode is a function that embeds x into $\boldsymbol{X} \in \mathbb{R}^{n\times h}$ with h being the hidden size, $\boldsymbol{M} \in \{0,1\}^{n\times m}$ is the alignment matrix between input and output tokens. In this framework, alignments $\boldsymbol{M}$ are separately predicted by $p_\phi(\boldsymbol{M}|\boldsymbol{X})$ to guide the computation $p_\theta(y|\boldsymbol{X},\boldsymbol{M})$ that maps x to y. The parameters of both model components (ϕ and θ) are disjoint.

Relation to Attention Standard encoder-decoder models (Bahdanau et al., 2015) rely on continuous attention weights i.e., $\boldsymbol{M}[:,i] \in \triangle^{n-1}$ for each target token $1 \leq i \leq m$. Discrete versions of attention (aka hard attention) have been studied in previous work (Xu et al., 2015; Deng et al., 2018) and show superior performance in certain tasks. In the discrete case $\boldsymbol{M}$ is a sequence of m categorical random variables. Though discrete, the hard attention only considers word-level alignments, i.e., assumes that each target token is aligned with a single source token. This is a limiting assumption; for example, in traditional statistical machine translation, word-based models (e.g., (Brown et al., 1993)) are known to achieve dramatically weaker results than phrase-based models (e.g., (Koehn et al., 2007)). In this chapter, we aim to bring the power of phrase-level (aka segment-level) alignments to neural seq2seq models. [1]

6.2 Latent Segment Alignments via Separable Permutations

Our method integrates a layer of segment-level alignments with a seq2seq model. The architecture of our model is shown in Figure 6.2. Central to this model is the alignment network, which decomposes the alignment problem into two stages: (i) input reordering

[1] One of our models (see Section 6.2.2) still has a flavor of standard continuous attention in that it approximates discrete alignments with continuous expectation.

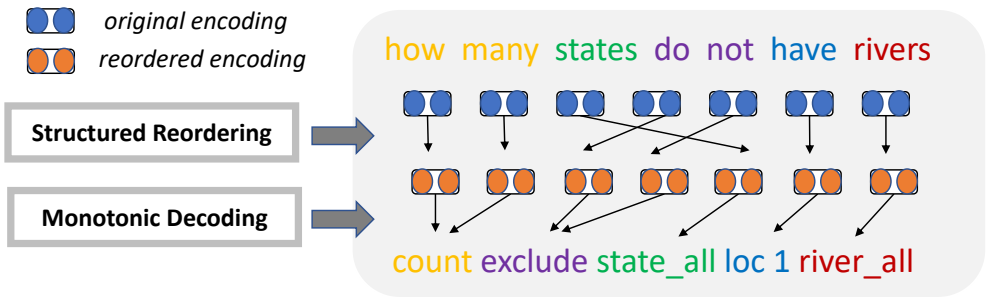


Figure 6.2: The architecture of our seq2seq model for semantic parsing. After encoding the input utterance, our model permutes the input representations using our reordering module. Then, the reordered encodings will be used for decoding the output program in a monotonic manner.

and (ii) monotonic alignment between the reordered sequence and the output. Formally, we decompose the alignment matrix from Eq 6.1 into two parts:

$$\boldsymbol{M} = \boldsymbol{M}_{\text{pe}} \boldsymbol{M}_{\text{mo}} \tag{6.2}$$

where $\boldsymbol{M}_{\text{pe}} \in \mathbb{R}^{n \times n}$ is a permutation matrix, and $\boldsymbol{M}_{\text{mo}} \in \mathbb{R}^{n \times m}$ represents monotonic alignments. With this decomposition, we can rewrite the objective in Eq 6.1 as follows:

$$\mathcal{L}_{\theta,\phi}(x,y) = -\log \mathbb{E}_{p_\phi(\boldsymbol{M}_{\text{pe}}|x)} \mathbb{E}_{p_{\phi'}(\boldsymbol{M}_{\text{mo}}|\boldsymbol{M}_{\text{pe}}\boldsymbol{X})} p_\theta(y|\boldsymbol{M}_{\text{pe}}\boldsymbol{X}, \boldsymbol{M}_{\text{mo}}) \tag{6.3}$$

where $\boldsymbol{M}_{\text{pe}}\boldsymbol{X}$ denotes the reordered representation. With a slight abuse of notation, ϕ now denotes the parameters of the model generating permutations, and ϕ' denotes the parameters used to produce monotonic alignments. Given the permutation matrix $\boldsymbol{M}_{\text{pe}}$, the second expectation $\mathbb{E}_{p_\phi(\boldsymbol{M}_{\text{mo}}|\boldsymbol{M}_{\text{pe}}\boldsymbol{X})} p_\theta(y|\boldsymbol{M}_{\text{pe}}\boldsymbol{X}, \boldsymbol{M}_{\text{mo}})$, which we denote as $p_{\theta,\phi'}(y|\boldsymbol{M}_{\text{pe}}\boldsymbol{X})$, can be handled by existing methods, such as SSNT (Yu et al., 2016) and SWAN (Wang et al., 2017b). In the rest of the chapter, we choose SSNT as the module for handling monotonic alignment.[2] We can rewrite the objective we optimize in the following compact form:

$$\mathcal{L}_{\theta,\phi,\phi'}(x,y) = -\log \mathbb{E}_{p_\phi(\boldsymbol{M}_{\text{pe}}|x)} p_{\theta,\phi'}(y|\boldsymbol{M}_{\text{pe}}\boldsymbol{X}) \tag{6.4}$$

6.2.1 Structured Latent Reordering by Binary Permutation Trees

Inspired by Stanojević and Steedman (2018) and Steedman (2021), we restrict word reorderings to separable permutations. Formally, separable permutations are defined

[2] In our initial experiments, we found that SWAN works as well as SSNT but is considerably slower.

in terms of binary permutation trees (aka separating trees (Bose et al., 1998)), i.e., if a permutation can be represented by a separating tree, it is separable. A binary permutation tree over a permutation of a sequence $1 \dots n$ is a binary tree in which each node represents the ordering of a segment $i \dots j$; the children exhaustively split their parent into sub-segments $i \dots k$ and $k+1 \dots j$. Each node has a binary label that decides whether the segment of the left child precedes that of the right child.

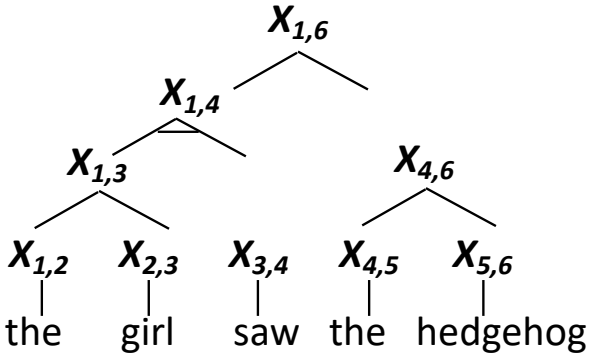


Figure 6.3: The tree represents the reordered sentences 'saw the girl the hedgehog' where $\triangle, \wedge$ denotes *Inverted* and *Straight*, respectively.

Separable Permutations Represented by BTG Bracketing transduction grammar (BTG, Wu, 1997), which is proposed in the context of machine translation, is the corresponding grammar to represent binary permutation trees. Specifically, we use a monolingual variant of the original BTG, which is a context-free grammar (CFG) with only one terminal X and three production rules:

$$X \xrightarrow{Straight} X \;\; X$$
$$X \xrightarrow{Inverted} X \;\; X$$
$$X \rightarrow w$$

where $w \in \Sigma$ denotes terminal words. The original BTG is a synchronous grammar describing the generative process of bilingual strings (i.e., input and output). In contrast, the variant we use only describes the generative process of monolingual strings. This variant differs from conventional CFG in that the first two production rules are labeled with orientations (i.e., Straight or Inverted) with respect to the ordering of the target language.

Separable permutations can be fully covered by BTG as each separable permutation can be represented by at least one BTG derivation tree. [3] Hence, the problem of learning to reorder an input string can be cast as a parsing problem where we need to learn how to generate a BTG parse given an input string.

[3] Note that the mapping is not bijective, i.e., a separable permutation could correspond to multiple derivation trees. Another variant of BTG (Figure 10 of Wu (1997)) can be used to resolve the ambiguity and guarantee bijectivity.

This hierarchical approach to generating separable permutations reflects the compositional nature of language, and, thus, appears more appealing than using 'flat' alternatives (Mena et al., 2018; Grover et al., 2019; Cuturi et al., 2019). Moreover, with BTGs, we can incorporate segment-level features to model separable permutations, and design tractable algorithms for learning and inference. However, separable permutations may not capture all the needed reorderings for all seq2seq tasks. For example, Stanojević and Steedman (2018) point out that BTG cannot capture all the expected reorderings for machine translation of any language pair. One possible way to address this issue is to stack more than one separable permutations (i.e., $\boldsymbol{M}_{\text{pe}} = \boldsymbol{M}_{\text{pe1}}\boldsymbol{M}_{\text{pe2}}\dots$) to cover more kinds of permutations beyond separable permutations. We leave the exploration for future work.

Probabilistic Separable Permutations by BTG Parsing We aim to learn a probabilistic BTG parser that assigns high probabilities to derivations that encode the right reordering. We use a discriminative parsing strategy (Durrett and Klein, 2015) where the scores of each rule is conditioned on the whole input string. This strategy has been found to be empirically better in terms of parsing performance compared to conventional PCFGs whose rule scores are independent of input strings. The disrcriminative parser relies on the following anchored rules:

$$\begin{aligned} \mathcal{S}_{i,j,k} &: \quad X_i^k \xrightarrow{Straight} X_i^j X_j^k \\ I_{i,j,k} &: \quad X_i^k \xrightarrow{Inverted} X_i^j X_j^k \\ \mathcal{T}_i &: \quad X_i^{i+1} \rightarrow x_i \end{aligned}$$

where X_i^k is the anchored non-terminal covering the segment from i to k (excluding k). The first two rules decide whether to keep or invert two segments when constructing a larger segment; the last rule states that every word x_i in an utterance is associated with a non-terminal X_i^{i+1}. An example is shown in Figure 6.3. Through this example, we note that the first two rules only signify which segments to inverse; an additional process of interpreting the tree (i.e., performing actual actions of keeping or inverting segments) is needed to obtain the permutated sequence.

By assigning a score to each anchored rule using segment-level features, we obtain a distribution over all possible derivations, and use it to compute the objective in Eq 6.4.

$$p_\phi(D|x) = \frac{\prod_{R\in D} f_\phi(R)}{Z(x,\phi)}, \quad \mathcal{L}_{\theta,\phi,\phi'}(x,y) = -\log \mathbb{E}_{p_\phi(D|x)} p_{\theta,\phi'}(y|\boldsymbol{M}_{\text{pe}}^D \boldsymbol{X}) \tag{6.5}$$

where f_ϕ is a score function assigning a (non-negative) weight to an anchored rule $R \in \{\mathcal{S}, I, \mathcal{T}\}$, $Z(x,\phi) = \sum_{D'} \prod_{R\in D'} f_\phi(R)$ is the partition function, which can be computed

using the inside algorithm, $\boldsymbol{M}_{\text{pe}}^{D}$ is the permutation matrix corresponding to the derivation D.

The resulting probabilistic BTG does not have generative power anymore and is merely a module to assign preference to BTG parses. In this probabilistic BTG, the weight is only normalized at the derivation level. As we will see in Algorithm 2, we are interested in normalizing the weight of production rules and converting it to an equivalent BTG with probabilistic rule scores, following Smith and Johnson (2007), so that the probability of a derivation can be computed as follows:

$$p_{\phi}(D|x) = \prod_{R \in D} G_{\phi}(R) \tag{6.6}$$

where $G_{\phi}(R)$ is the weight of the production rule R under the normalized BTG.

The challenge with optimizing the objective in Eq 6.5 is that the search space of possible derivations is exponential, making the estimation of the gradients with respect to parameters of the reordering component (ϕ) non-trivial. We now present two differentiable surrogates we use.

6.2.2 Soft Reordering: Computing Marginal Permutations

The first strategy is to use the deterministic expectation of permutations to softly reorder a sentence, analogous to the way standard attention approximates categorical random variables. Specifically, we use the following approximation:

$$\boldsymbol{M}'_{\text{pe}} = \mathbb{E}_{p_{\phi}(D|x)} \boldsymbol{M}_{\text{pe}}^{D}$$
$$\mathcal{L}_{\theta,\phi,\phi'}(x,y) \approx -\log p_{\theta,\phi'}(y|\boldsymbol{M}'_{\text{pe}}\boldsymbol{X})$$

where $\boldsymbol{M}'_{\text{pe}}$ is the marginal permutation matrix, and it can be treated as structured attention (Kim et al., 2017). Methods for performing marginal inference for anchored rules, i.e., computing the marginal distribution of production rules are well-known in NLP (Manning and Schutze, 1999). However, we are interested in the marginal permutation matrix (or equivalently the expectation of the matrix components) as the matrix is the data structure that is ultimately used in our model. As a key contribution of this work, we propose an efficient algorithm to exactly compute the marginal permutation matrix using dynamic programming.

In order to compute the marginal permutation matrix we need to marginalize over the exponentially many derivations of each permutation. We propose to map a derivation of BTG into its corresponding permutation matrix in a recursive manner. Specifically, we

Algorithm 2 Dynamic programming for computing marginals and differentiable sampling of permutation matrix wrt. a parameterized grammar

Input: $G_\phi(R)$: probablity of an anchored rule R
sampling: whether perform sampling

for $i := 1$ **to** n **do**
  $\boldsymbol{E}_i^{i+1} = \mathbf{1}$
end for
for $w := 2$ **to** n **do** ▷ width of spans
  for $i := 1$ **to** $n - w + 1$ **do** ▷ start point
    $k := i + w$ ▷ end point
    if *sampling* **then**
      $\hat{G}_\phi(R) = \text{s\_arg}\max(G_\phi(R))$ ▷ differentiable sampling
    else
      $\hat{G}_\phi(R) = G_\phi(R)$ ▷ computing marginal
    end if
    for $j := i + 1$ **to** $k - 1$ **do**
      $\boldsymbol{E}_i^k \mathrel{+}= \hat{G}_\phi(\mathcal{S}_{i,j,k})(\boldsymbol{E}_i^j \oplus \boldsymbol{E}_j^k)$
      $\boldsymbol{E}_i^k \mathrel{+}= \hat{G}_\phi(I_{i,j,k})(\boldsymbol{E}_i^j \ominus \boldsymbol{E}_j^k)$
    end for
  end for
end for
return $\boldsymbol{E}_1^{n+1}$

first associate word i with an identity permutation matrix $\boldsymbol{M}_i^{i+1} = \mathbf{1}$; then we associate *Straight* and *Inverted* rules with direct $\oplus$ and skew $\ominus$ sums of permutation matrices, respectively:

$$\boldsymbol{A} \oplus \boldsymbol{B} = \begin{bmatrix} \boldsymbol{A} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{B} \end{bmatrix} \qquad \boldsymbol{A} \ominus \boldsymbol{B} = \begin{bmatrix} \mathbf{0} & \boldsymbol{A} \\ \boldsymbol{B} & \mathbf{0} \end{bmatrix}$$

For example, the permutation matrix of the derivation tree shown in Figure 6.3 can be obtained by:

$$\boldsymbol{M}_1^6 = \Big(\big((\boldsymbol{M}_1^2 \oplus \boldsymbol{M}_2^3) \ominus \boldsymbol{M}_3^4 \big) \oplus (\boldsymbol{M}_4^5 \oplus \boldsymbol{M}_5^6) \Big) \tag{6.7}$$

Intuitively, the permutation matrix of long segments can be constructed by composing permutation matrices of short segments. Motivated by this, we propose a dynamic programming algorithm, which takes advantage of the observation that we can reuse the permutation matrices of short segments when computing permutation matrices of long

segments, as shown in Algorithm 2. While the above equation is defined over discrete permutation matrices encoding a single derivation, the algorithm applies recursive rules to expected permutation matrices. Central to the algorithm is the following recursion:

$$\boldsymbol{E}_i^k = \sum_{i<j<k} G_\phi(\mathcal{S}_{i,j,k})(\boldsymbol{E}_i^j \oplus \boldsymbol{E}_j^k) + G_\phi(I_{i,j,k})(\boldsymbol{E}_i^j \ominus \boldsymbol{E}_j^k) \tag{6.8}$$

where $\boldsymbol{E}_i^k$ is the expected permutation matrix for the segment from i to k, $G_\phi(R)$ is the probability of employing the production rule R, defined in Eq 6.6. Overall, Algorithm 2 is a bottom-up method that constructs expected permutation matrices incrementally in Step 13 and 14, while relying on the probability of the associated production rule.

6.2.3 Hard Reordering: Gumbel-Permutation by Differentiable Sampling

During **inference**, for efficiency, it is convenient to rely on the most probable derivation D' and its corresponding most probable y:

$$\arg\max_y p_{\theta,\phi'}(y|\boldsymbol{M}_{\text{pe}}^{D'}\boldsymbol{X}) \tag{6.9}$$

where $D' = \arg\max_D p_\phi(D|x)$. The use of discrete permutations $\boldsymbol{M}_{\text{pe}}^{D'}$ during inference and soft reorderings during training lead to a training-inference gap which may be problematic. Inspired by recent Gumbel-Softmax operator (Jang et al., 2016; Maddison et al., 2016) that relaxes the sampling procedure of a categorical distribution using the Gumbel-Max trick, we propose a differentiable procedure to obtain an approximate sample $\boldsymbol{M}_{\text{pe}}^{D}$ from $p(D|x)$. Concretely, the Gumbel-Softmax operator relaxes the perturb-and-MAP procedure (Papandreou and Yuille, 2011), where we add noises to probability logits and then relax the MAP inference (i.e., arg max in the categorical case); we denote this operator as s_arg max. In our structured case, we perturb the logits of the probabilities of production rules $G_\phi(R)$, and relax the structured MAP inference for our problem. Recall that $p(D|x)$ is converted to a normalized BTG, and MAP inference is algorithmically similar to marginal inference. Intuitively, for each segment, instead of marginalizing over all possible production rules in marginal inference, we choose the one with the highest probability (i.e., a local MAP inference with categorical random variables) during MAP inference. By relaxing each local MAP inference with Gumbel-Softmax (Step 8 of Algorithm 2), we obtain a differentiable sampling procedure. [4] We

[4] If we change s_arg max with arg max in Step 8 of Algorithm 2, we will obtain the algorithm for exact MAP inference.

choose Straight-Through Gumbel-Softmax so that the return of Algorithm 2 is a discrete permutation matrix, and in this way we close the training-inference gap faced by soft reordering.

Summary We propose two efficient algorithms for computing marginals and obtaining samples of separable permutations with their distribution parameterized via BTG. In both algorithms, the normalized BTG plays an important role of decomposing a global problem into sub-problems, which explains why we convert $p(D|x)$ into a normalized BTG in Eq 6.6. Relying on the proposed algorithms, we present two relaxations of the discrete permutations that let us induce latent reorderings with end-to-end training. We refer to the resulting system as *ReMoto*, short for a seq2seq model with Reordered-then-Monotone alignments. Soft-*ReMoto* and Hard-*ReMoto* denote the versions which use soft marginal permutations and hard Gumbel permutations, respectively.

SSNT for Monotonic Alignments We briefly explain SSNT, the module we use on top of our reordering module for monotonically generating output. Intuitively, SSNT can be viewed as a finite state transducer (FST) that alternates between consuming an input segment and generating an output segment. The alignments ($\boldsymbol{M}_{\text{mo}}$ in Equation 6.3) between the reordered input and output strings is operationalized by a sequence of local decisions of alternating. Central to SSNT is the dynamic programming algorithm that efficiently enumerates all possible sequences of local decisions, as the monotonic alignments are unknown and treated as discrete latent variables. That SSNT consumes input strings in a left-to-right, segment-by-segment manner provides a strong inductive bias that the input of SSNT should have monotonic correspondence with respect to the output. When coupled with our reordering module, the joint model, when trained end-to-end, learns to induce the right reordered input based on a given input and output pair. Intuitively, two modules will learn to adjust to each other: our reordering model needs to generate reordered input strings that SSNT finds suitable in terms of monotonic generation.

Different from our reordering module, segments are not first-class objects during modeling in SSNT. In our reordering module, permutation matrices $\boldsymbol{M}_{\text{pe}}$ are constructed by hierarchically reordering input segments. In contrast, $\boldsymbol{M}_{\text{mo}}$ modeled by SSNT is realized by a series of token-level decisions, e.g., whether to keep consuming the next input token. Thus, properties of segments (e.g., segment-level features) are not fully exploited in SSNT. In this sense, one potential way to further improve *ReMoto* is to

explore better alternatives to SSNT that can treat segments as first-class objects as well.

Reordering in Previous Work In traditional statistical machine translation (SMT), reorderings are typically handled by a distortion model (e.g., Al-Onaizan and Papineni, 2006) in a pipeline manner. Neubig et al. (2012), Nakagawa (2015) and Stanojević and Sima'an (2015) also use BTGs for modeling reorderings. Stanojević and Sima'an (2015) go beyond binarized grammars, showing how to support 5-ary branching permutation trees. Still, they assume the word alignments have been produced on a preprocessing step, using an alignment tool Och and Ney (2003). Relying on these alignments, they induce reorderings. Inversely, we rely on latent reordering to induce the underlying word and segment alignments. Modeling segments provides a strong inductive bias, reflecting the intuition that sequence transduction in NLP can be largely accomplished by manipulations at the level of segments. In contrast, there is no explicit notion of segments in conventional seq2seq methods.

Reordering modules have been previously used in neural models, and can be assigned to the following two categories. First, reordering components (Huang et al., 2017; Chen et al., 2019) were proposed for neural machine translation. However, they are not structured or sufficiently constrained in the sense that they may produce invalid reorderings (e.g., a word is likely to be moved to more than one new position). In contrast, our module is a principled way of dealing with latent reorderings. Second, the generic permutations (i.e., one-to-one matchings or sorting), though having differentiable counterparts (Mena et al., 2018; Grover et al., 2019; Cuturi et al., 2019), do not suit our needs as they are defined in terms of tokens, rather than segments. For comparison, in our experiments, we design baselines that are based on Gumbel-Sinkhorn Network (Mena et al., 2018), which is used previously in NLP (e.g., Lyu and Titov (2018)).

6.3 Experiments

First, we consider two diagnostic tasks where we can test the neural reordering module on its own. Then we further assess our general seq2seq model *ReMoto* on two real-world NLP tasks.

Dataset	Input	Output
Arithmetic	$((1+9)*((7+8)/4))$	$((19+)((78+)4/)*)$
SCAN-SP	jump twice after walk around left thrice	after (twice (jump), thrice(walk (around, left)))
GeoQuery	how many states do not have rivers ?	count(exclude(state(all), loc_1(river(all))))

Table 6.1: Examples of input-output pairs for parsing tasks.

	Arithmetic		**SCAN-SP**	
Model	IID	LEN	IID	LEN
Seq2Seq	100.0	0.0	100.0	13.9
LSTM-based Tagging	100.0	20.6	100.0	57.7
Sinkhorn-Attention Tagging	99.5	8.8	100.0	48.2
Soft-*ReMoto*	100.0	**86.9**	100.0	100.0
- with shared parameters	100.0	40.9	100.0	100.0
Hard-*ReMoto*	100.0	83.3	100.0	100.0

Table 6.2: Accuracy (%) on the arithmetic and SCAN-SP tasks.

6.3.1 Diagnostic Tasks

Arithmetic We design a task of converting an arithmetic expression in infix format to the one in postfix format. An example is shown in Table 6.1. We create a synthetic dataset by sampling data from a PCFG. In order to generalize, a system needs to learn how to manipulate internal sub-structures (i.e., segments) while respecting well-formedness constraints. This task can be solved by the shunting-yard algorithm but we are interested to see if neural networks can solve it and generalize ood by learning from raw infix-postfix pairs. For standard splits (IID), we randomly sample 20k infix-postfix pairs whose nesting depth is set to be between 1 and 6; 10k, 5k, 5k of these pairs are used as train, dev and test sets, respectively. To test systematic generalization, we create a Length split (LEN) where training and dev examples remain the same as IID splits, but test examples have a nesting depth of 7. In this way, we test whether a system can generalize to unseen longer input.

SCAN-SP We use the SCAN dataset (Lake and Baroni, 2018), which consists of simple English commands coupled with sequences of discrete actions. Here we use the semantic parsing version, SCAN-SP (Herzig and Berant, 2021), where the goal is to predict programs corresponding to the action sequences. An example is shown

in Table 6.1. As in these experiments our goal is to test the reordering component alone, we remove parentheses and commas in programs. For example, the program `after (twice (jump), thrice(walk (around, left)))` is converted to a sequence: `after twice jump thrice walk around left`. In this way, the resulting parentheses-free sequence can be viewed as a reordered sequence of the NL utterance 'jump twice after walk around left thrice'.[5] Apart from the standard split (IID, aka simple split (Lake and Baroni, 2018)), we create a Length split (LEN) where the training set contains NL utterances with maximum length 5, while utterances in the dev and test sets have minimum length of 6.[6]

Baselines and Results In both diagnostic tasks, we use *ReMoto* with a trivial monotonic alignment matrix $\boldsymbol{M}_{\text{mo}}$ (an identity matrix) in Eq 6.3. Essentially, *ReMoto* becomes a sequence tagging model. We consider three baselines: (1) vanilla Seq2Seq models with Luong attention (Luong et al., 2015); (2) an LSTM-based tagging model which learn the reordering implicitly, and can be viewed as a version *ReMoto* with a trivial $\boldsymbol{M}_{\text{pe}}$ and $\boldsymbol{M}_{\text{mo}}$; (3) Sinkhorn Attention that replaces the permutation matrix of Soft-*ReMoto* in Eq 6.4 by Gumbel-Sinkhorn networks (Mena et al., 2018).

We report results by averaging over three runs in Table 6.2. In both datasets, almost all methods achieve perfect accuracy in IID splits. However, baseline systems cannot generalize well to the challenging LEN splits. In contrast, our methods, both Soft-*ReMoto* and Hard-*ReMoto* perform very well on LEN splits, surpassing the best baseline system by large margins ($> 40\%$). The results indicate that *ReMoto*, particularly its neural reordering module, has the right inductive bias to learn reorderings. We also test a variant Soft-*ReMoto* where parameters θ, ϕ with shared input embeddings. This variant does not generalize well to the LEN split on the arithmetic task, showing that it is beneficial to split models of the 'syntax' (i.e., alignment) and 'semantics', confirming what has been previously observed (Havrylov et al., 2019; Russin et al., 2019).

6.3.2 Semantic Parsing

Our second experiment is on semantic parsing where *ReMoto* models the latent alignment between NL utterances and their corresponding programs. We use GeoQuery

[5] The grammar of the programs is known so we can reconstruct the original program from the intermediate parentheses-free sequences using the grammar.

[6] Since we use the program form, the original length split (Lake and Baroni, 2018), which is based on the length of action sequence, is not very suitable in our experiments.

Model	EN			ZH			DE		
	IID	TEMP	LEN	IID	TEMP	LEN	IID	TEMP	LEN
Seq2Seq	75.7	38.8	21.8	72.5	25.4	19.8	56.1	18.8	15.2
Syntactic Attention (Russin et al., 2019)	74.3	39.1	18.3	70.2	27.9	18.7	54.3	19.3	14.2
SSNT (Yu et al., 2016)	75.3	38.7	19.1	71.6	23.8	17.8	55.2	19.8	14.1
Soft-*ReMoto*	74.5	39.3	19.8	73.4	30.3	17.3	55.8	19.5	13.4
Hard-*ReMoto*	75.2	43.2	23.2	74.3	45.7	22.3	55.6	22.3	16.6

Table 6.3: Exact-match accuracy (%) on three splits of the multilingual GeoQuery dataset. Numbers underlined are significantly better than others (p-value ≤ 0.05 using the paired permutation test).

dataset (Zelle and Mooney, 1996) which contains 880 utterance-programs pairs. The programs are in variable-free form (Kate et al., 2005); an example is shown in Table 6.1. Similarly to SCAN-SP, we transform the programs into parentheses-free form which have better structural correspondence with utterances.[7] An example of such parentheses-free form is shown in Figure 6.2. Apart from the standard version, we also experiment with the Chinese and German versions of GeoQuery (Jones et al., 2012; Susanto and Lu, 2017). Since different languages exhibit divergent word orders (Steedman, 2021), the results in the multilingual setting will tell us if our model can deal with this variability.

In addition to standard IID splits, we create a LEN split where the training examples have parentheses-free programs with a maximum length 4; the dev and test examples have programs with a minimum length 5. We also experiment with the TEMP split (Herzig and Berant, 2021) where training and test examples have programs with disjoint templates.

Baselines and Results Apart from conventional seq2seq models, for comparison, we also implemented the syntactic attention Russin et al. (2019). Our model *ReMoto* is similar in spirit to the syntactic attention, 'syntax' in their model (i.e., alignment) and 'semantics' (i.e., producing the representation relying on the alignment) are separately modeled. In contrast to our structured mechanism for modeling alignments, their syntactic attention still relies on the conventional attention mechanism. We also compare with SSNT, which can be viewed as an ablated version of *ReMoto* by removing our reordering module.

Results are shown in Table 6.3. For the challenging TEMP and LEN splits, our best performing model Hard-*ReMoto* achieves consistently stronger performance than

[7] Again, we can reconstruct the original programs based on the grammar.

seq2seq, syntactic attention and SSNT. Thus, our model bridges the gap between conventional seq2seq models and specialized state-of-the-art grammar-based models Shaw et al. (2020); Herzig and Berant (2021).[8]

6.3.3 Machine Translation

Our final experiment is on small-scale machine translation tasks, where *ReMoto* models the latent alignments between parallel sentences from two different languages. To probe systematic generalization, we also create a LEN split for each language pair in addition to the standard IID splits.

English-Japanese We use the small en-ja dataset extracted from TANKA Corpus. The original split (IID) has 50k/500/500 examples for train/dev/test with lengths 4-16 words.[9] We create a LEN split where the English sentences of training examples have a maximum length 12 whereas the English sentences in dev/test have a minimum length 13. The LEN split has 50k/538/538 examples for train/dev/test, respectively.

Chinese-English We extract a subset from FBIS corpus (LDC2003E14) by filtering English sentences with length 4-30. We randomly shuffle the resulting data to obtain an IID split which has 141k/3k/3k examples for train/dev/test, respectively. In addition, we create a LEN split where English sentences of training examples have a maximum length 29 whereas the English sentences of dev/test examples have a length 30. The LEN split has 140k/4k/4k examples as train/dev/test sets respectively.

Baselines and Results In addition to the conventional seq2seq, we compare with the original SSNT model which only accounts for monotonic alignments. We also implemented a variant that combines SSNT with the local reordering module (Huang et al., 2017) as our baseline to show the advantage of our structured ordering module.

Results are shown in Table 6.4. Our model, especially Hard-*ReMoto*, consistently outperforms other baselines on both splits. In EN-JA translation, the advantage of our best-performance Hard-*ReMoto* is slightly more pronounced in the LEN split than in the IID split. In ZH-EN translation, while SSNT and its variant do not outperform seq2seq in the LEN split, *ReMoto* can still achieve better results than seq2seq. These results

[8] NQG (Shaw et al., 2020) achieves 35.0% in the English LEN, and SBSP (Herzig and Berant, 2021) (without lexicon) achieves 65.9% in the English TEMP in execution accuracy. Both models are augmented with pre-trained representations (BERT).

[9] https://github.com/odashi/small_parallel_enja

t

	EN-JA		ZH-EN	
	IID	LEN	IID	LEN
Seq2Seq	35.6	25.3	21.4	18.1
SSNT (Yu et al., 2016)	36.3	26.5	20.5	17.3
Local Reordering (Huang et al., 2017)	36.0	27.1	21.8	17.8
Soft-*ReMoto*	36.6	27.5	22.3	19.2
Hard-*ReMoto*	**37.4**	**28.7**	**22.6**	**19.5**

Table 6.4: BLEU scores on the EN-JA and ZH-EN translation.

original input:	在$_{\text{in}}^{1}$ 美国$_{\text{usa}}^{2}$ 哪些$_{\text{which}}^{3}$ 州$_{\text{state}}^{4}$ 与^{5} 最长$_{\text{longest}}^{6}$ 的^{7} 河流$_{\text{river}}^{8}$ 接壤$_{\text{border}}^{9}$
reordered input:	州$_{\text{state}}^{4}$ 接壤$_{\text{border}}^{9}$ 最长$_{\text{longest}}^{6}$ 的^{7} 河流$_{\text{river}}^{8}$ 与^{5} 哪些$_{\text{which}}^{3}$ 美国$_{\text{usa}}^{2}$ 在$_{\text{in}}^{1}$
prediction:	`state`^{4} `next_to_2`^{9} `longest river`^{6,7,8} `loc_2 countryid_ENTITY`^{5,3,2}
ground truth:	`state next_to_2 longest river loc_2 countryid_ENTITY`
original input:	according^{1} to^{2} the^{3} newspaper^{4} ,^{5} there^{6} was^{7} a^{8} big^{9} fire^{10} last^{11} night^{12}
reordered input:	according^{1} to^{2} the^{3} newspaper^{4} ,^{5} night^{12} last^{11} big^{9} fire^{10} a^{8} there^{6} was^{7}
prediction:	新によれば、^{1,2,3,4,5} 昨夜^{12} 大^{11,9} 火事^{10} があ^{8,6} った^{7}
ground truth:	新によると昨夜大火事があった

Table 6.5: Output examples of Chinese semantic parsing and English-Japanese translation. For clarity, the input words are labeled with position indices, and, for semantic parsing, with English translations. A prediction consists of multiple segments, each annotated with a superscript referring to input tokens.

show that our model is better than its alternatives at generalizing to longer sentences for machine translation.

Interpretability Latent alignments, apart from promoting systematic generalization, also lead to better interpretability as discrete alignments reveal the internal process for generating output. For example, in Table 6.5, we show a few examples from our model. Each output segment is associated with an underlying rationale, i.e. a segment of the reordered input.

6.4 Summary

To improve systematic generalization for linguistic coverage, we propose a new seq2seq model for semantic parsing that accounts for latent segment-level alignments. Central to this model is a novel structured reordering module which is coupled with existing modules to handle non-monotonic segment alignments. We model reorderings as separable permutations and propose an efficient dynamic programming algorithm to perform marginal inference and sampling. It allows latent reorderings to be induced with end-to-end training. This model is general and applicable to other seq2seq NLP problems such as machine translation. Empirical results on both synthetic and real-world datasets show that our model can achieve better systematic generalization than conventional seq2seq models.

Chapter 7

Conclusions

In this thesis, we presented several latent-alignment semantic parsers and specialized training objectives to address the generalization challenges that arise in an array of settings inspired by practical scenarios. Such settings extend the conventional semantic parsing setting along three dimensions: the shift from in-domain to cross-domain setting, from strong to weak (and even no) supervision, from compositionally in-distribution to out-of-distribution settings. Our findings in the thesis can be summarized as follows:

- Structural *model biases* introduced by the proposed latent-alignment models are highly beneficial in such practical and challenging settings. By capturing the structured correspondences between natural language utterances and programs, latent-alignment models can take advantage of weak supervisions (chapter 4), generalize well to unseen domains (Chapter 3) and novel natural language utterances with known atoms (Chapter 6).

- In contrast to improving model architectures, *learning biases* introduced by the proposed specialized training objectives, are model-agnostic and surprisingly powerful ways of injecting prior knowledge. We design meta-learning inspired objectives for domain generalization (Chapter 3) and compositional generalization (Chapter 6), and a posterior-regularization inspired objective for learning from executions (Chapter 5).

In general, we can conclude that model biases and learning biases are two effective ways of augmenting and improving models of semantic parsing. The former one has been studied for decades since the pre-neural era, from rule-based to grammar-based, then to neural-based models. The latent-alignment models we proposed aim at effectively mixing symbolic grammar-based models and neural seq2seq models. In contrast, the

latter notion of learning biases is relatively new to this field, but we show that it is very promising and appealing in many respects. First, they are model-agnostic, implying that you can plug in a general-purpose seq2seq model without much of an engineering effort. Second, such training objectives seem complementary to model biases if we were to inject them together. For instance, in Chapter 3, we show that meta-learning objective can still further boost the performance of our state-of-the-art alignment-based parser. We believe that the general idea of learning biases for neural models can be extended to other related NLP tasks such as knowledge-based question answering.

The inductive biases address the data underspecification problem i.e., many models are compatible with the given data, but only some of them can generalize to new data. Apart from the two kinds of inductive biases above, which are ways to restrict and regularize the model, we have also explored ways of manipulating data i.e., injecting prior knowledge through data augmentation (Wang et al., 2021d). In real-life scenarios, it might be desirable to combine those strategies together to maximize performance.

7.1 Future Work

Though this thesis explores a set of practical settings that extend the standard semantic parsing settings inspired by real-life scenarios, it does not consider many other complex settings such as conversational semantic parsing (Hemphill et al., 1990; Yu et al., 2019b,a), interactive semantic parsing (Li and Jagadish, 2014; Gur et al., 2018; Yao et al., 2019a), and mulitilingual semantic parsing (Jones et al., 2012; Susanto and Lu, 2017; Sherborne et al., 2020). We will discuss some problems in these settings, where applying or extending our methodology is possible.

Conversational Semantic Parsing When facing a complex goal of querying structural data, users tend to achieve it via multiple thematically related questions, instead of asking it in one shot. In such conversational settings, a parser would need to revise or refer to programs generated in previous turns (Andreas et al., 2020). As a result, the structural correspondences of interest for semantic parsing not only exist among natural language utterances, programs and data schema, but also have dependencies on previously generated programs. Hence, to extend our methodology of modeling latent alignments, a structural mechanism of explicitly modeling the latent revision or reference would be necessary for such settings.

Interactive Semantic Parsing The implicit assumption made in typical settings of semantic parsing is that a user utterance can be mapped to an executable program, i.e., there is a clear one-to-one mapping at utterance level. However, in practice, users may ask incomplete or ambiguous questions which can have zero or multiple correct interpretations. In the interactive setting, a natural language interface is supposed to ask clarification questions in the presence of ambiguity or incompleteness. Uncertainty is often used as an informative signal (Yao et al., 2019b) to detect such undesirable questions. We speculate that latent-alignment models could potentially provide a better estimation of uncertainty compared with standard seq2seq models as vague questions presumably do not exhibit structural alignments, thus do not have a corresponding program with high probability. Moreover, the latent alignments induced by our model might serve as rationales when a system asks clarification questions, which would make it easier for users to understand the motivation of automatically generated clarification questions. Overall, latent-alignment models seem to an appealing family for interactive semantic parsing and require further investigation.

Mulitilingual Semantic Parsing In Chapter 6, we show that our latent-alignment models can generalize across different languages when labeled examples are available in each language. However, this might not be practical as it would be extremely expensive to annotate examples for each existing language. A more economical setting is to transfer semantic parsers trained in English, which has rich resources in many aspects, to other languages. As utterances in different languages would result in different structural correspondences (e.g., due to different word orders), it would be interesting to investigate how to transfer knowledge learned from one latent-alignment model for language A to another language B; whether there exists a single centralized model that can explicitly captures structural correspondences for all languages. Moreover, our meta-learning objective can be potentially extended to multilingual semantic parsing based on the intuition that improving a semantic parser on language A should also be beneficial for a related language B.

Bibliography

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

Alfred V Aho and Jeffrey D Ullman. 1973. *The theory of parsing, translation, and compiling*, volume 1. Prentice-Hall Englewood Cliffs, NJ.

Ekin Akyurek and Jacob Andreas. 2021. Lexicon learning for few shot sequence modeling. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4934–4946, Online. Association for Computational Linguistics.

Yaser Al-Onaizan and Kishore Papineni. 2006. Distortion models for statistical machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 529–536.

Massih-Reza Amini and Patrick Gallinari. 2002. Semi-supervised logistic regression. In *ECAI*, pages 390–394.

Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, Hao Fang, Alan Guo, David Hall, Kristin Hayes, Kellie Hill, Diana Ho, Wendy Iwaszuk, Smriti Jha, Dan Klein, Jayant Krishnamurthy, Theo Lanman, Percy Liang, Christopher H. Lin, Ilya Lintsbakh, Andy McGovern, Aleksandr Nisnevich, Adam Pauls, Dmitrij Petters, Brent Read, Dan Roth, Subhro Roy, Jesse Rusak, Beth Short, Div Slomin, Ben Snyder, Stephon Striplin, Yu Su, Zachary Tellman, Sam Thomson, Andrei Vorobev, Izabela Witoszko, Jason Wolfe, Abby Wray, Yuchen Zhang, and Alexander Zotov. 2020. Task-oriented dialogue as dataflow synthesis. *Transactions of the Association for Computational Linguistics*, 8:556–571.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 39–48.

Jacob Andreas, Andreas Vlachos, and Stephen Clark. 2013. Semantic parsing as machine translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 47–52, Sofia, Bulgaria. Association for Computational Linguistics.

Martin Arjovsky. 2020. *Out of distribution generalization in machine learning*. Ph.D. thesis, New York University.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *arXiv:1607.06450*.

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

Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville. 2019. Systematic generalization: what is required and can it be learned? *ICLR*.

Yogesh Balaji, Swami Sankaranarayanan, and Rama Chellappa. 2018. Metareg: Towards domain generalization using meta-regularization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 998–1008. Curran Associates, Inc.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria. Association for Computational Linguistics.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proc. of EMNLP*.

Ben Bogin, Jonathan Berant, and Matt Gardner. 2019a. Representing schema structure with graph neural networks for text-to-SQL parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4560–4565.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019b. Global reasoning over database structures for text-to-SQL parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3659–3664, Hong Kong, China. Association for Computational Linguistics.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250.

Prosenjit Bose, Jonathan F Buss, and Anna Lubiw. 1998. Pattern matching for permutations. *Information Processing Letters*, 65(5):277–283.

Antoine Bosselut, Asli Celikyilmaz, Xiaodong He, Jianfeng Gao, Po-Sen Huang, and Yejin Choi. 2018. Discourse-aware neural rewards for coherent text generation. *arXiv preprint arXiv:1805.03766*.

Peter F Brown, Stephen A Della Pietra, Vincent J Della Pietra, and Robert L Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311.

Gino Brunner, Yang Liu, Damian Pascual, Oliver Richter, Massimiliano Ciaramita, and Roger Wattenhofer. 2020. On identifiability in Transformers. In *International Conference on Learning Representations*.

Ronnie Cann. 1993. *Formal semantics an introduction*. Cambridge University Press, Cambridge [etc. OCLC: 1120437841.

Olivier Chapelle and Alexander Zien. 2005. Semi-supervised classification by low density separation. In *AISTATS*, volume 2005, pages 57–64. Citeseer.

Kehai Chen, Rui Wang, Masao Utiyama, and Eiichiro Sumita. 2019. Neural machine translation with reordering embeddings. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1787–1799.

Xinyun Chen, Chen Liang, Adams Wei Yu, Dawn Song, and Denny Zhou. 2020. Compositional generalization via neural-symbolic stack machines. *arXiv preprint arXiv:2008.06662*.

Xinyun Chen, Chang Liu, and Dawn Song. 2018. Execution-guided neural program synthesis. In *International Conference on Learning Representations*.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 263–270, Ann Arbor, Michigan. Association for Computational Linguistics.

DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. 2020. RYANSQL: Recursively applying sketch-based slot fillings for complex text-to-SQL in cross-domain databases. *arXiv preprint arXiv:2004.03125*.

Noam Chomsky. 1965. *Aspects of the theory of syntax*, 50th anniversary edition edition. Number no. 11 in Massachusetts Institute of Technology. Research Laboratory of Electronics. Special technical report. The MIT Press, Cambridge, Massachusetts.

Henry Conklin, Bailin Wang, Kenny Smith, and Ivan Titov. 2021. Meta-learning to compositionally generalize. In *ACL*.

Gonçalo M Correia, Vlad Niculae, Wilker Aziz, and André FT Martins. 2020. Efficient marginalization of discrete and structured latent variables via sparsity. *arXiv preprint arXiv:2007.01919*.

Caio Corro and Ivan Titov. 2019. Learning latent trees with stochastic perturbations and differentiable dynamic programming. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5508–5521, Florence, Italy. Association for Computational Linguistics.

Marco Cuturi, Olivier Teboul, and Jean-Philippe Vert. 2019. Differentiable ranks and sorting using optimal transport. *arXiv preprint arXiv:1905.11885*.

Deborah A Dahl, Madeleine Bates, Michael K Brown, William M Fisher, Kate Hunicke-Smith, David S Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the atis task: The atis-3 corpus. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2020. Structure-grounded pretraining for text-to-sql. *arXiv preprint arXiv:2010.12773*.

Yuntian Deng, Yoon Kim, Justin Chiu, Demi Guo, and Alexander Rush. 2018. Latent alignment and variational attention. In *Proc. of NeurIPS*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proc. of ACL*.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proc. of ACL*.

Greg Durrett and Dan Klein. 2015. Neural CRF parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 302–312, Beijing, China. Association for Computational Linguistics.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. 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.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org.

Jerry A Fodor and Zenon W Pylyshyn. 1988. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71.

Yao Fu, Chuanqi Tan, Bin Bi, Mosha Chen, Yansong Feng, and Alexander M Rush. 2020. Latent template induction with gumbel-crfs. *arXiv preprint arXiv:2011.14244*.

Yarin Gal and Zoubin Ghahramani. 2016. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In *Advances in Neural Information Processing Systems 29*, pages 1019–1027.

Kuzman Ganchev, Joao Graça, Jennifer Gillenwater, and Ben Taskar. 2010. Posterior regularization for structured latent variable models. *The Journal of Machine Learning Research*, 11:2001–2049.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. Allennlp: A deep semantic natural language processing platform.

Muhammad Ghifary, W Bastiaan Kleijn, Mengjie Zhang, and David Balduzzi. 2015. Domain generalization for object recognition with multi-task autoencoders. In *Proceedings of the IEEE international conference on computer vision*, pages 2551–2559.

Ofer Givoli and Roi Reichart. 2019. Zero-shot semantic parsing for instructions. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4454–4464, Florence, Italy. Association for Computational Linguistics.

Omer Goldman, Veronica Latcinnik, Ehud Nave, Amir Globerson, and Jonathan Berant. 2018. Weakly supervised semantic parsing with abstract examples. In *Proc. of ACL*.

Emily Goodwin, Koustuv Sinha, and Timothy J O'Donnell. 2020. Probing linguistic systematicity. *arXiv preprint arXiv:2005.04315*.

Yves Grandvalet and Yoshua Bengio. 2005. Semi-supervised learning by entropy minimization. In *Advances in neural information processing systems*, pages 529–536.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 2015. Learning to transduce with unbounded memory. In *Proc. of NeurIPS*, pages 1828–1836.

Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. 2019. Stochastic optimization of sorting networks via continuous relaxations. *arXiv preprint arXiv:1903.08850*.

Jiatao Gu, Yong Wang, Yun Chen, Victor O. K. Li, and Kyunghyun Cho. 2018. Meta-learning for low-resource neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3622–3631, Brussels, Belgium. Association for Computational Linguistics.

Sumit Gulwani, Oleksandr Polozov, Rishabh Singh, et al. 2017. Program synthesis. *Foundations and Trends® in Programming Languages*, 4(1-2):1–119.

Daya Guo, Duyu Tang, Nan Duan, Ming Zhou, and Jian Yin. 2019a. Coupling retrieval and meta-learning for context-dependent semantic parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 855–866, Florence, Italy. Association for Computational Linguistics.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019b. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535.

Yuting Guo, Xiangjue Dong, Mohammed Ali Al-Garadi, Abeed Sarker, Cecile Paris, and Diego Mollá Aliod. 2020. Benchmarking of transformer-based pre-trained models on social media text classification datasets. In *Proceedings of the The 18th Annual Workshop of the Australasian Language Technology Association*, pages 86–91, Virtual Workshop. Australasian Language Technology Association.

Gunshi Gupta, Karmesh Yadav, and Liam Paull. 2020. La-maml: Look-ahead meta learning for continual learning. *arXiv preprint arXiv:2007.13904*.

Izzeddin Gur, Semih Yavuz, Yu Su, and Xifeng Yan. 2018. DialSQL: Dialogue based structured query generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1339–1349, Melbourne, Australia. Association for Computational Linguistics.

Kelvin Guu, Panupong Pasupat, Evan Liu, and Percy Liang. 2017. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1051–1062, Vancouver, Canada. Association for Computational Linguistics.

Till Haug, Octavian-Eugen Ganea, and Paulina Grnarova. 2018. Neural multi-step reasoning for question answering on semi-structured tables. *ECIR*.

Serhii Havrylov, Germán Kruszewski, and Armand Joulin. 2019. Cooperative learning of disjoint syntax and semantics. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human*

Language Technologies, Volume 1 (Long and Short Papers), pages 1118–1128, Minneapolis, Minnesota. Association for Computational Linguistics.

Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen. 2019. X-SQL: reinforce schema representation with context. *arXiv preprint arXiv:1908.08113*.

Vincent J. Hellendoorn, Charles Sutton, Rishabh Singh, Petros Maniatis, and David Bieber. 2020. Global relational models of source code. In *International Conference on Learning Representations*.

Charles T. Hemphill, John J. Godfrey, and George R. Doddington. 1990. The ATIS spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*.

Gary G Hendrix. 1982. Natural-language interface. *American Journal of Computational Linguistics*, 8(2):56–61.

Jonathan Herzig and Jonathan Berant. 2017. Neural Semantic Parsing over Multiple Knowledge-bases. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 623–628, Stroudsburg, PA, USA.

Jonathan Herzig and Jonathan Berant. 2018. Decoupling structure and lexicon for zero-shot semantic parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1619–1629, Brussels, Belgium. Association for Computational Linguistics.

Jonathan Herzig and Jonathan Berant. 2021. Span-based semantic parsing for compositional generalization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 908–921, Online. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam Shazeer, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. 2019. Music Transformer. In *International Conference on Learning Representations*.

Po-Sen Huang, Chenglong Wang, Rishabh Singh, Wen-tau Yih, and Xiaodong He. 2018. Natural language to structured query generation via meta-learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 732–738.

Po-Sen Huang, Chong Wang, Sitao Huang, Dengyong Zhou, and Li Deng. 2017. Towards neural phrase-based machine translation. *arXiv preprint arXiv:1706.05565*.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. 2019. The compositionality of neural networks: integrating symbolism and connectionism. *arXiv:1908.08351 [cs, stat]*. ArXiv: 1908.08351.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. 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, Vancouver, Canada. Association for Computational Linguistics.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Bevan Jones, Mark Johnson, and Sharon Goldwater. 2012. Semantic parsing with Bayesian tree transducers. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 488–496, Jeju Island, Korea. Association for Computational Linguistics.

Aishwarya Kamath and Rajarshi Das. 2018. A survey on semantic parsing. *arXiv preprint arXiv:1812.00978*.

Rohit J Kate, Yuk Wah Wong, and Raymond J Mooney. 2005. Learning to transform natural to formal languages. In *AAAI*, volume 5, pages 1062–1068.

Amol Kelkar, Rohan Relan, Vaishali Bhardwaj, Saurabh Vaichal, and Peter Relan. 2020. Bertrand-DR: Improving text-to-SQL using a discriminative re-ranker. *arXiv preprint arXiv:2002.00557*.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, et al. 2019. Measuring compositional generalization: A comprehensive method on realistic data. *arXiv preprint arXiv:1912.09713*.

Najoung Kim and Tal Linzen. 2020. COGS: A Compositional Generalization Challenge Based on Semantic Interpretation. *arXiv:2010.05465 [cs]*. ArXiv: 2010.05465.

Yoon Kim, Carl Denton, Luong Hoang, and Alexander M Rush. 2017. Structured attention networks. In *Proc. of ICLR*.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proc. of ICLR*.

Tomáš Kočiský, Gábor Melis, Edward Grefenstette, Chris Dyer, Wang Ling, Phil Blunsom, and Karl Moritz Hermann. 2016. Semantic parsing with semi-supervised sequential autoencoders. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1078–1087, Austin, Texas. Association for Computational Linguistics.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic. Association for Computational Linguistics.

Philipp Koehn, Franz J. Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 127–133.

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017*

Conference on Empirical Methods in Natural Language Processing, pages 1516–1526, Copenhagen, Denmark. Association for Computational Linguistics.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, pages 2873–2882. PMLR.

Dongjun Lee, Jaesik Yoon, Jongyun Song, Sanggil Lee, and Sungroh Yoon. 2019. One-shot learning for text-to-sql generation.

Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. 2018a. Learning to generalize: Meta-learning for domain generalization. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Fei Li and H. V. Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84.

Haoliang Li, Sinno Jialin Pan, Shiqi Wang, and Alex C Kot. 2018b. Domain generalization with adversarial feature learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5400–5409.

Chen Liang, Mohammad Norouzi, Jonathan Berant, Quoc V Le, and Ni Lao. 2018. Memory augmented policy optimization for program synthesis and semantic parsing. In *Proc. of NeurIPS*.

P. Liang, M. I. Jordan, and D. Klein. 2011. Learning dependency-based compositional semantics. In *Proc. of ACL*.

Percy Liang. 2016. Learning executable semantic parsers for natural language understanding. *Communications of the ACM*, 59(9):68–76.

Percy Liang, Michael I. Jordan, and Dan Klein. 2013. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4870–4888, Online. Association for Computational Linguistics.

Joao Loula, Marco Baroni, and Brenden M Lake. 2018. Rearranging the familiar: Testing compositional generalization in recurrent networks. *arXiv preprint arXiv:1807.07545*.

Wei Lu. 2014. Semantic parsing with relaxed hybrid trees. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1308–1318, Doha, Qatar.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proc. of EMNLP*.

Chunchuan Lyu and Ivan Titov. 2018. AMR parsing as graph prediction with latent alignment. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 397–407, Melbourne, Australia. Association for Computational Linguistics.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. 2016. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

Christopher Manning and Hinrich Schutze. 1999. *Foundations of statistical natural language processing*. MIT press.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

Andre Martins and Ramon Astudillo. 2016. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International Conference on Machine Learning*, pages 1614–1623.

Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. 2018. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*.

Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. 2018. Learning latent permutations with gumbel-sinkhorn networks. *arXiv preprint arXiv:1802.08665*.

Arthur Mensch and Mathieu Blondel. 2018. Differentiable dynamic programming for structured prediction and attention. *Proc. of ICML*.

Qingkai Min, Yuefeng Shi, and Yue Zhang. 2019a. A pilot study for Chinese SQL semantic parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3652–3658, Hong Kong, China. Association for Computational Linguistics.

Sewon Min, Danqi Chen, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2019b. A discrete hard EM approach for weakly supervised question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2851–2864, Hong Kong, China. Association for Computational Linguistics.

Tetsuji Nakagawa. 2015. Efficient top-down BTG parsing for machine translation preordering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 208–218, Beijing, China. Association for Computational Linguistics.

Arvind Neelakantan, Quoc V Le, Martin Abadi, Andrew McCallum, and Dario Amodei. 2017. Learning a natural language interface with neural programmer. In *Proc. of ICLR*.

Graham Neubig, Taro Watanabe, and Shinsuke Mori. 2012. Inducing a discriminative parser to optimize machine translation reordering. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 843–853, Jeju Island, Korea. Association for Computational Linguistics.

Alex Nichol, Joshua Achiam, and John Schulman. 2018. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*.

Vlad Niculae, André FT Martins, Mathieu Blondel, and Claire Cardie. 2018. Sparsemap: Differentiable sparse structured inference. *arXiv preprint arXiv:1802.04223*.

Will Norcliffe-Brown, Efstathios Vafeias, and Sarah Parisot. 2018. Learning conditioned graph structures for interpretable visual question answering. *arXiv preprint arXiv:1806.07243*.

Maxwell Nye, Luke Hewitt, Joshua Tenenbaum, and Armando Solar-Lezama. 2019. Learning to infer program sketches. *Proc. of ICML*.

Maxwell I Nye, Armando Solar-Lezama, Joshua B Tenenbaum, and Brenden M Lake. 2020. Learning compositional rules via neural program synthesis. *arXiv preprint arXiv:2003.05562*.

Abiola Obamuyide and Andreas Vlachos. 2019. Model-agnostic meta-learning for relation classification with limited supervision. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5873–5879, Florence, Italy. Association for Computational Linguistics.

Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

George Papandreou and Alan L Yuille. 2011. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. In *Proc. of ICCV*.

P. Pasupat and P. Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proc. of ACL*.

Panupong Pasupat and Percy Liang. 2016. Inferring logical forms from denotations. In *Proc. of ACL*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proc. of EMNLP*.

Ana-Maria Popescu, Oren Etzioni, , and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, pages 149–157.

Sachin Ravi and Hugo Larochelle. 2016. Optimization as a model for few-shot learning.

Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. 2018. Meta-learning for semi-supervised few-shot classification. *arXiv preprint arXiv:1803.00676*.

Ohad Rubin and Jonathan Berant. 2020. Smbop: Semi-autoregressive bottom-up semantic parsing. *arXiv preprint arXiv:2010.12412*.

Jake Russin, Jason Jo, Randall C O'Reilly, and Yoshua Bengio. 2019. Compositional generalization in a deep seq2seq model by separating syntax and semantics. *arXiv preprint arXiv:1904.09708*.

Torsten Scholak, Raymond Li, Dzmitry Bahdanau, Harm de Vries, and Chris Pal. 2020. Duorat: Towards simpler text-to-sql models. *arXiv preprint arXiv:2010.11119*.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2020. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? *arXiv preprint arXiv:2010.12725*.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-Attention with Relative Position Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468.

Tom Sherborne, Yumo Xu, and Mirella Lapata. 2020. Bootstrapping a crosslingual semantic parser. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 499–517, Online. Association for Computational Linguistics.

Tianze Shi, Kedar Tatwawadi, Kaushik Chakrabarti, Yi Mao, Oleksandr Polozov, and Weizhu Chen. 2018. Incsql: Training incremental text-to-sql parsers with non-deterministic oracles. *arXiv preprint arXiv:1809.05054*.

Noah A Smith and Mark Johnson. 2007. Weighted and probabilistic context-free grammars are equally expressive. *Computational Linguistics*, 33(4):477–491.

Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. 2011. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 conference on empirical methods in natural language processing*, pages 151–161.

Yan Song, Shuming Shi, Jing Li, and Haisong Zhang. 2018. Directional skip-gram: Explicitly distinguishing left and right context for word embeddings. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 175–180, New Orleans, Louisiana. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958.

Miloš Stanojević and Khalil Sima'an. 2015. Reordering grammar induction. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 44–54.

Miloš Stanojević and Mark Steedman. 2018. Formal basis of a language universal. *Computational Linguistics*, pages 1–34.

Mark Steedman. 2000. *The syntactic process*, volume 24. MIT press Cambridge, MA.

Mark Steedman. 2021. A formal universal of natural language grammar. *Language*, 96(3):618–660.

Yu Su and Xifeng Yan. 2017. Cross-domain semantic parsing via paraphrasing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1235–1246, Copenhagen, Denmark.

Alane Suhr, Ming-Wei Chang, Peter Shaw, and Kenton Lee. 2020. Exploring unexplored generalization challenges for cross-database semantic parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8372–8388, Online. Association for Computational Linguistics.

Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William Cohen. 2018a. Open domain question answering using early fusion of knowledge bases and text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4231–4242, Brussels, Belgium. Association for Computational Linguistics.

Yibo Sun, Duyu Tang, Nan Duan, Yeyun Gong, Xiaocheng Feng, Bing Qin, and Daxin Jiang. 2019. Neural semantic parsing in low-resource settings with back-translation and meta-learning.

Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, and Ming Zhou. 2018b. Semantic parsing with syntax-and table-aware sql generation. In *Proc. of ACL*.

Raymond Hendy Susanto and Wei Lu. 2017. Semantic parsing with neural hybrid trees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. 1999. Policy gradient methods for reinforcement learning with function approximation. In *NIPs*, volume 99, pages 1057–1063. Citeseer.

Oscar Täckström, Kuzman Ganchev, and Dipanjan Das. 2015. Efficient inference and structured learning for semantic role labeling. *TACL*.

Lappoon R. Tang and Raymond J. Mooney. 2000. Automated construction of database interfaces: Intergrating statistical and relational learning for semantic parsing. In *2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 133–141.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *International Conference on Learning Representations*.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. 2016. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638.

Martin J Wainwright, Michael I Jordan, et al. 2008. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305.

Bailin Wang, Mirella Lapata, and Ivan Titov. 2021a. Learning from executions for semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*.

Bailin Wang, Mirella Lapata, and Ivan Titov. 2021b. Meta-learning for domain generalization in semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*.

Bailin Wang, Mirella Lapata, and Ivan Titov. 2021c. Structured reordering for modeling latent alignments in sequence transduction. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Bailin Wang, Ivan Titov, and Mirella Lapata. 2019. Learning semantic parsers from denotations with latent structured alignments and abstract programs. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Bailin Wang, Wenpeng Yin, Xi Victoria Lin, and Caiming Xiong. 2021d. Learning to synthesize data for semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2760–2766, Online. Association for Computational Linguistics.

Chenglong Wang, Marc Brockschmidt, and Rishabh Singh. 2017a. Pointing out SQL queries from text. Technical report, MSR.

Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. 2018. Robust text-to-sql generation with execution-guided decoding. *arXiv preprint arXiv:1807.03100*.

Chong Wang, Yining Wang, Po-Sen Huang, Abdelrahman Mohamed, Dengyong Zhou, and Li Deng. 2017b. Sequence modeling via segmentations. In *International Conference on Machine Learning*, pages 3674–3683. PMLR.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1332–1342, Beijing, China. Association for Computational Linguistics.

Terry Winograd. 1971. MIT AI Technical Report 235: Procedures as a Representation for Data in a Computer Program for Understanding Natural Language. Technical report, MIT, Cambridge, MA, USA.

Sam Wiseman and Alexander M Rush. 2016. Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960*.

Yuk Wah Wong and Raymond Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 439–446, New York City, USA. Association for Computational Linguistics.

Yuk Wah Wong and Raymond Mooney. 2007a. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 960–967, Prague, Czech Republic.

Yuk Wah Wong and Raymond Mooney. 2007b. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proc. of ACL*.

W. A. Woods. 1973. Progress in natural language understanding: An application to lunar geology. In *Proceedings of the June 4-8, 1973, National Computer Conference and Exposition*, AFIPS '73, pages 441–450, New York, NY, USA. ACM.

Dekai Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational linguistics*, 23(3):377–403.

Lijun Wu, Fei Tian, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. 2018. A study of reinforcement learning for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3612–3621, Brussels, Belgium. Association for Computational Linguistics.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

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

Kenji Yamada and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 523–530, Toulouse, France. Association for Computational Linguistics.

Ziyu Yao, Xiujun Li, Jianfeng Gao, Brian Sadler, and Huan Sun. 2019a. Interactive semantic parsing for if-then recipes via hierarchical reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2547–2554.

Ziyu Yao, Yu Su, Huan Sun, and Wen-tau Yih. 2019b. Model-based interactive semantic parsing: A unified framework and a text-to-SQL case study. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5447–5458, Hong Kong, China. Association for Computational Linguistics.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1321–1331, Beijing, China. Association for Computational Linguistics.

Pengcheng Yin and Graham Neubig. 2017. A Syntactic Neural Model for General-Purpose Code Generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426, Online. Association for Computational Linguistics.

Pengcheng Yin, Chunting Zhou, Junxian He, and Graham Neubig. 2018. StructVAE: Tree-structured latent variable models for semi-supervised semantic parsing. In *Pro-*

ceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 754–765, Melbourne, Australia. Association for Computational Linguistics.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. 2016. Learning to compose words into sentences with reinforcement learning. *arXiv preprint arXiv:1611.09100*.

Lei Yu, Jan Buys, and Phil Blunsom. 2016. Online segment to segment neural transduction. *arXiv preprint arXiv:1609.08194*.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018a. Typesql: Knowledge-based type-aware neural text-to-sql generation. In *Proc. of NAACL*.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. 2020a. Grappa: Grammar-augmented pre-training for table semantic parsing. *arXiv preprint arXiv:2009.13845*.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018b. SyntaxSQLNet: Syntax tree networks for complex and cross-domain text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1653–1663, Brussels, Belgium. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. 2019a. CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1962–1979, Hong Kong, China. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Alex Polozov, Christopher Meek, and Ahmed Hassan Awadallah. 2021. {SC}ore: Pre-training for context representation in conversational semantic parsing. In *International Conference on Learning Representations*.

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. 2018c. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019b. SParC: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4511–4523, Florence, Italy. Association for Computational Linguistics.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020b. Gradient surgery for multi-task learning. *NeurIPS*.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. 2017. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055.

Luke S Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: structured classification with probabilistic categorial grammars. In *Proc. of UAI*.

Luke S Zettlemoyer and Michael Collins. 2012. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *arXiv preprint arXiv:1207.1420*.

Rui Zhang, Tao Yu, He Yang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-based SQL query generation for cross-domain context-dependent questions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*.

Yuchen Zhang, Panupong Pasupat, and Percy Liang. 2017. Macro grammars and holistic triggering for efficient semantic parsing. In *Proceedings of the 2017 Conference on*

Empirical Methods in Natural Language Processing, pages 1214–1223, Copenhagen, Denmark. Association for Computational Linguistics.

Victor Zhong, Mike Lewis, Sida I. Wang, and Luke Zettlemoyer. 2020. Grounded adaptation for zero-shot executable semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6869–6882, Online. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *arXiv:1709.00103 [cs]*.