Weakly Supervised Semantic Parsing by Learning from Mistakes

Jiaqi Guo[†*] **Jian-Guang Lou**[‡] **Ting Liu**[†] **Dongmei Zhang**[‡]
[†]Xi'an Jiaotong University, Xi'an, China
[‡]Microsoft Research, Beijing, China
jasperguo2013@stu.xjtu.edu.cn tingliu@mail.xjtu.edu.cn
{jlou,dongmeiz}@microsoft.com

Abstract

Weakly supervised semantic parsing (WSP) aims at training a parser via utterance-denotation pairs. This task is challenging because it requires (1) searching consistent logical forms in a huge space; and (2) dealing with spurious logical forms. In this work, we propose Learning from Mistakes (LFM), a simple yet effective learning framework for WSP. LFM utilizes the mistakes made by a parser during searching, i.e., generating logical forms that do not execute to correct denotations, for tackling the two challenges. In a nutshell, LFM additionally trains a parser using utterance-logical form pairs created from mistakes, which can quickly bootstrap the parser to search consistent logical forms. Also, it can motivate the parser to learn the correct mapping between utterances and logical forms, thus dealing with the spuriousness of logical forms. We evaluate LFM on WikiTableQuestions, WikiSQL, and TabFact in the WSP setting. The parser trained with LFM outperforms the previous state-of-the-art semantic parsing approaches on the three datasets. Also, we find that LFM can substantially reduce the need for labeled data. Using only 10% of utterance-denotation pairs, the parser achieves 84.2 denotation accuracy on WikiSQL, which is competitive with the previous state-of-the-art approaches using 100% labeled data.

1 Introduction

Semantic parsing is the task of mapping a natural language utterance to a *logical form* that can be executed against a knowledge base to obtain a *denotation*. Much progress has been made in this area, thanks to the emergence of datasets that include a large number of utterance-logical form pairs. However, collecting such pairs at scale is generally expensive, because annotators must be skilled at programming. By contrast, collecting utterance-denotation pairs is much cheaper, because it can be performed by non-experts. Hence, it is tempting to train a semantic parser via utterance-denotation pairs, framing a **w**eakly supervised **s**emantic **p**arsing problem (WSP) (Clarke et al., 2010; Liang et al., 2013; Zhang et al., 2017).

Rank	Nation	Gold	Silver	Bronze
1	France	3	1	1
2	Ukraine	2	1	2
3	Turkey	2	0	1
4	Sweden	2	0	0
5	Iran	1	2	1

Utterance: Who won the most silver medals?
Denotation: Iran

Logical Form	Consistent	Spurious
`(hop Nation (argmax Silver rows))`	✓	
`(hop Nation (argmax Rank rows))`	✓	✓
`(hop Nation (argmin Gold rows))`	✓	✓
`(hop Nation (first rows))`	×	
`(hop Nation (argmin Silver rows))`	×	

Figure 1: An illustrative example of weakly supervised semantic parsing.

Training a parser from denotations rather than logical forms complicates training in two ways. First, training a parser requires exploring the huge space of logical forms to find those that execute to correct denotations, which we call "*consistent*" logical forms. This is a very difficult search problem due to the combinatorial nature of the search space. Figure 1 presents five logical forms for an utterance-denotation pair, among which the first three are consistent and the rest are mistake logical forms (they do not execute to the correct denotation). Second, consistent logical forms can be "*spurious*". Spurious logical forms accidentally execute to correct denotations, but they do not reflect the meaning of utterances. For example, two of the three consistent logical forms in Figure 1 are spurious, and only the first one is "*correct*", reflecting the utterance's meaning. The presence of spurious

[*]Work done during an internship at Microsoft Research.

Findings of the Association for Computational Linguistics: EMNLP 2021, pages 2603–2617
November 7–11, 2021.

logical forms severely hinders a parser from learning the correct mapping between utterances and logical forms.

Existing approaches for WSP can be categorized into *static* and *dynamic*, according to whether they perform searching at training time. Static approaches heuristically search consistent logical forms offline (Krishnamurthy et al., 2017; Wang et al., 2019a; Min et al., 2019). They assume that there are correct logical forms in the search results, and they do not perform searching at training time. However, this assumption may not hold when the spuriousness is severe. Considering a binary denotation (TRUE or FALSE), 50% of syntactically valid logical forms execute to the correct denotation, regardless of their semantics. Dynamic approaches do not make this assumption. They iteratively search consistent logical forms using a parser and train the parser with the search result in turn (Guu et al., 2017; Liang et al., 2017, 2018; Agarwal et al., 2019). But dynamic approaches generally suffer from a cold-start problem, because it is challenging for a randomly initialized parser to search consistent logical forms in an exponentially large space. Hence, most dynamic approaches require a set of pre-searched consistent logical forms to bootstrap the training.

In this work, we propose **L**earning **f**rom **M**istakes (**LFM** for short), a simple yet effective dynamic learning framework for WSP. The core insight of LFM is that a parser will generate a huge number of mistake logical forms during searching. These mistake logical forms can be fully utilized to overcome the cold-start and spuriousness problems. In a nutshell, every time a parser makes a mistake, LFM synthesizes a faithful utterance for the mistake logical form. Then, LFM trains the parser with this utterance-logical form pair, so that the parser is taught the correct meaning of the mistake logical form. In addition, LFM also trains the parser like existing dynamic approaches, using consistent logical forms with learning objectives such as REINFORCE (Williams, 1992) and Maximum Marginal Likelihood (MML).

LFM has two major advantages over existing dynamic approaches. First, LFM does not need to pre-search consistent logical forms to warm start the training. Instead, it creates utterance-logical form pairs from mistakes on the fly to overcome the cold-start problem. Second, LFM can facilitate a parser learning the correct mapping between utterances and logical forms. Since the synthesized utterances are guaranteed to reflect the meaning of logical forms, a parser can learn the correct mapping from the synthesized utterance-logical form pairs. The idea of LFM is inspired by our human beings. Every time we make a mistake, we try to modify our knowledge to avoid suffering again in the future for the same reason (Giordana and Serra, 2001). Similarly, every time a parser makes a mistake, we try to teach the parser the correct meaning of the mistake logical form and help it avoid the mistake in the next round of searching.

To demonstrate the effectiveness of LFM, we conduct experiments on three challenging semantic parsing datasets in the WSP setting. The neural semantic parser trained with LFM achieves a denotation accuracy of 52.3 on WikiTableQuestions, 86.9 on WikiSQL, and 68.2 on TabFact, which all surpass previous state-of-the-art approaches in the same setting. Through a fine-grained analysis, we show that LFM is effective in addressing the cold-start and spuriousness problems, and LFM is more effective than prior data augmentation techniques for WSP. Also, we find that LFM can substantially reduce the need for labeled data to train a good parser. For example, the parser achieves an accuracy of 84.2 on WikiSQL using only 10% of utterance-denotation pairs, which already performs on par with previous state-of-the-art approaches.

2 Related Work

2.1 Weakly Supervised Semantic Parsing

As mentioned in the previous section, prior approaches for WSP can be categorized into static and dynamic. Static approaches, such as Krishnamurthy et al. (2017), heuristically search consistent logical forms offline and train a parser with the MML objective. When there are too many consistent logical forms for an utterance-denotation pair, they only consider top K shortest logical forms (typically $K \leq 100$) and perform a beam search to approximate the sum in MML. Wang et al. (2019a) introduce an alignment model to distinguish between spurious and correct logical forms. The alignment model is jointly optimized with a parser via MML. Min et al. (2019) replace MML with a discrete hard EM objective and observe improvements on WikiSQL and some reading comprehension datasets.

Dynamic approaches iteratively search consistent logical forms using a parser and optimize the

parser via the search result in turn. For example, Liang et al. (2013) and Berant et al. (2013) perform a beam search on a parser at each training step to search consistent logical forms, and they optimize the parser with an approximated MML that sums over consistent logical forms in the beam. Guu et al. (2017) propose a randomized beam search and a β-meritocratic update strategy to improve the searching of consistent logical forms. Instead of using MML, Liang et al. (2017) optimize a parser with the REINFORCE algorithm (Williams, 1992). They sample logical forms at each training step to compute an unbiased estimate of the gradient. Liang et al. (2018) leverage a memory buffer of consistent logical forms to reduce the variance of policy gradient estimate. Agarwal et al. (2019) introduce an auxiliary reward function to provide fine-grained feedback for dealing with spurious logical forms. Our LFM framework also falls into this dynamic category. Unlike the approaches introduced above that primarily leverage consistent logical forms for optimization, LFM fully utilizes the mistakes made by a parser during searching to address the cold-start and spuriousness problems. Hence, LFM can be considered orthogonal to prior dynamic approaches.

There is another line of work that tackles WSP without logical forms (Neelakantan et al., 2017; Mou et al., 2017; Herzig et al., 2020). Neelakantan et al. (2017) propose a neural model that sequentially predicts symbolic operations over semi-structured tables, and the model can be trained end-to-end with utterance-denotation pairs. Herzig et al. (2020) and Eisenschlos et al. (2020) pre-train a language model for table understanding. They show that the pre-trained model can be used to address WSP with a simple cell selection module and a set of differentiable aggregation operators.

2.2 Data Augmentation for Semantic Parsing

Our work also closely relates to the area of data augmentation for semantic parsing, since LFM synthesizes utterance-logical form pairs. Jia and Liang (2016) induce a synchronous context-free grammar (SCFG) (Chiang, 2005) from manually labeled utterance-logical form pairs. They randomly sample new pairs from the SCFG and train a parser using both labeled and sampled data, leading to significant improvements on several fully supervised semantic parsing tasks. Goldman et al. (2018) manually induce an SCFG and pre-train a neural semantic parser using data sampled from the SCFG. The parser is then finetuned via utterance-denotation pairs using MML. Similar ideas have also been adopted to address the Text-to-SQL problem (Iyer et al., 2017; Yu et al., 2018, 2021). Instead of using SCFG, Guo et al. (2018), Zhong et al. (2020a), and Wang et al. (2021) train a SQL-to-question neural model via utterance-logical form pairs. They synthesize more training data by randomly sampling SQL queries and generating corresponding questions with the model. One shortcoming of the data augmentation work above is that they need to carefully design logical form sampling procedures and pre-define the amount of data to synthesize. It has been found that over-extensive data augmentation will cause a deep-learning model to overfit, leading to even worse performance than that without data augmentation (Shorten and Khoshgoftaar, 2019).

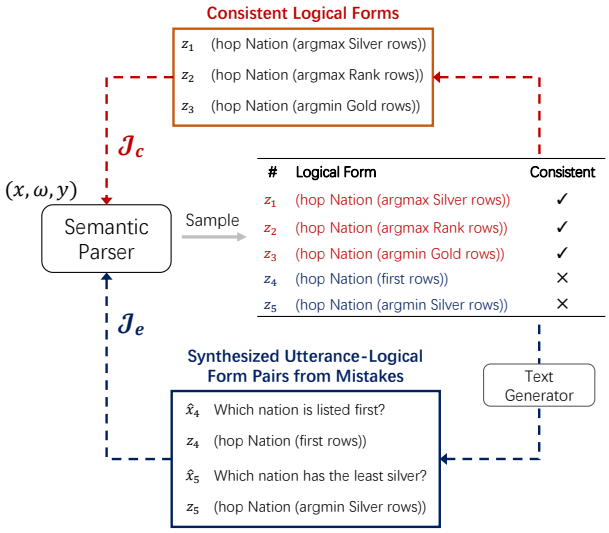

Figure 2: Visualization of a training step in LFM. The input utterance is "Who won the most silver medals?".

In LFM, the mistakes made by a parser serve as the source for data augmentation, and therefore, we do not need extra logical form sampling procedures. Also, we do not need to pre-define the amount of data to synthesize, because synthesis data are created at each training step. As we will show in Section 5.2, LFM is more effective than the other data augmentation techniques in WSP.

3 Learning Framework

In this section, we formally define the task of WSP and describe LFM in detail.

3.1 Preliminaries

Task Formulation Given a training set of N examples $\{(x_i, \omega_i, y_i)\}_{i=1}^{N}$, where x_i is an utterance,

ω_i is the knowledge base that x_i is interested in (e.g., the semi-structured table in Figure 1), and y_i is the denotation of x_i, the goal of WSP is to learn a parser (with parameter $\boldsymbol{\theta}$) that can map an unseen utterance x to a logical form z, such that z executes to the correct denotation y in the knowledge base ω, i.e., $[\![z]\!]^{\omega} = y$. The parser defines a distribution over logical forms conditioned on the given x and ω: $P(z|x, \omega; \boldsymbol{\theta})$.

Text Generator Suppose that we have access to a text generator $G(z, \omega)$, which generates a faithful utterance for a given logical form z and its knowledge base ω. The text generator can be implemented with either an SCFG or a neural network.

3.2 LFM: Learning from Mistakes

The learning objective $\mathcal{J}$ in LFM is made up of two sub-objectives: $\mathcal{J}_c$ and $\mathcal{J}_e$.

$$\mathcal{J} = \mathcal{J}_c + \gamma \mathcal{J}_e \tag{1}$$

Like prior dynamic approaches (Guu et al., 2017; Liang et al., 2017), $\mathcal{J}_c$ primarily leverages consistent logical forms searched during training to optimize a parser. $\mathcal{J}_c$ can be instantiated as MML or REINFORCE. By contrast, $\mathcal{J}_e$ leverages the mistakes made by the parser to overcome the cold-start problem and facilitate the parser learning the correct mapping between utterances and logical forms. γ is a hyper-parameter to blend the two sub-objectives.

Figure 2 visualizes a training step in LFM. Given a training example (x, ω, y), a set of K logical forms $\mathcal{Z} = \{z_j\}_{j=1}^{K}$ are sampled from a parser via beam search or Monte Carlo sampling. Suppose that among the K logical forms, only a subset of them $\mathcal{Z}_c$ are consistent ($[\![z_j]\!]^{\omega} = y$, z_1-z_3 in Figure 2), and the remaining $\mathcal{Z}_e = \mathcal{Z} - \mathcal{Z}_c$ logical forms are mistakes ($[\![z_j]\!]^{\omega} \neq y$, z_4-z_5 in Figure 2).

If $\mathcal{J}_c$ is instantiated as MML, $\mathcal{J}_c$ is derived as follows:

$$\mathcal{J}_c(\boldsymbol{\theta}) = logP(y|x, \omega) = log \sum_{[\![z]\!]^{\omega}=y} P(z|x, \omega; \boldsymbol{\theta}) \tag{2}$$

$$\approx log \sum_{z_j \in \mathcal{Z}_c} P(z_j|x, \omega; \boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{\theta}} \mathcal{J}_c \approx \sum_{z_j \in \mathcal{Z}_c} q(z_j) \nabla_{\boldsymbol{\theta}} logP(z_j|x, \omega; \boldsymbol{\theta}), \tag{3}$$

$$\text{where} \quad q(z_j) = \frac{P(z_j|x, \omega; \boldsymbol{\theta})}{\sum_{z_i \in \mathcal{Z}_c} P(z_i|x, \omega; \boldsymbol{\theta})}.$$

If $\mathcal{J}_c$ is instantiated as REINFORCE, $\mathcal{J}_c$ is derived as follows:

$$\mathcal{J}_c(\boldsymbol{\theta}) = \mathbb{E}_{z \sim P(\cdot|x, \omega; \boldsymbol{\theta})} R(z) \tag{4}$$

$$\nabla_{\boldsymbol{\theta}} \mathcal{J}_c \approx \frac{1}{K} \sum_{z_j \in \mathcal{Z}} R(z_j) \nabla_{\boldsymbol{\theta}} logP(z_j|x, \omega; \boldsymbol{\theta}), \tag{5}$$

where $R(z)$ is a reward function. Following prior REINFORCE-based WSP approaches (Liang et al., 2017, 2018; Agarwal et al., 2019), $R(z)$ is set to 1 if $[\![z]\!]^{w} = y$; otherwise 0. To this end, equation 5 can be re-written as follows:

$$\nabla_{\boldsymbol{\theta}} \mathcal{J}_c \approx \frac{1}{K} \sum_{z_j \in \mathcal{Z}_c} \nabla_{\boldsymbol{\theta}} logP(z_j|x, \omega; \boldsymbol{\theta}) \tag{6}$$

It is clear from Equation 3 and 6 that $\mathcal{J}_c$ primarily leverages consistent logical forms to optimize a parser. However, at the early stage of training, a randomly initialized parser hardly samples consistent logical forms in an exponentially large space, rendering a severe cold-start problem.

To overcome this problem, LFM fully utilizes the large number of mistake logical forms generated by the parser and introduces an extra training objective $\mathcal{J}_e$. Although a mistake logical form $z_j \in \mathcal{Z}_e$ fails to execute to y and does not reflect the meaning of x, we can leverage a text generator G to generate an utterance $\hat{x} = G(z_j, \omega)$, such that z_j reflects the meaning of $\hat{x}$. By optimizing the parser's likelihood of generating z_j given $\hat{x}$ and ω, we can bootstrap the parser and overcome the cold-start problem. Also, we can motivate the parser to learn the correct mapping between utterances and logical forms. Formally, the objective $\mathcal{J}_e$ is defined as follows:

$$\mathcal{J}_e(\boldsymbol{\theta}) = \sum_{z_j \in \mathcal{Z}_e} logP(z_j|G(z_j, \omega), \omega; \boldsymbol{\theta}) \tag{7}$$

$$\nabla_{\boldsymbol{\theta}} J_e = \sum_{z_j \in \mathcal{Z}_e} \nabla_{\boldsymbol{\theta}} logP(z_j|G(z_j, \omega), \omega; \boldsymbol{\theta}) \tag{8}$$

Algorithm 1 summarizes the training procedure of LFM. In each training step, LFM first searches consistent logical forms for a given utterance-denotation pair (Line 4). Then, it optimizes the parser using consistent logical forms with objective $\mathcal{J}_c$ (Line 5). For the remaining mistake logical forms, LFM synthesizes their corresponding utterances and optimizes the parser with $\mathcal{J}_e$ (Line 6-10).

Algorithm 1: Learning from Mistakes

Input: training data $\{(x_i, \omega_i, y_i)\}_{i=1}^{N}$
Output: final parameters $\boldsymbol{\theta}$ of the parser

1 **repeat**
2 Get a batch $\mathcal{B}$ from training data;
3 **for** $(x, \omega, y) \in \mathcal{B}$ **do**
4 Sample logical forms $\mathcal{Z} = \mathcal{Z}_e \cup \mathcal{Z}_e$ from $P(z|x, \omega, y; \boldsymbol{\theta})$;
5 $d\boldsymbol{\theta} \leftarrow d\boldsymbol{\theta} + \nabla_{\boldsymbol{\theta}} \mathcal{J}_c$; // $\mathcal{J}_c$
6 **for** $z \in \mathcal{Z}_e$ **do**
7 $\hat{z} \leftarrow$ FixAndDiversify(z, ω);
8 $\hat{x} \leftarrow$ GenerateUtterance$(\hat{z}, \omega)$;
9 $d\boldsymbol{\theta} \leftarrow d\boldsymbol{\theta} + \gamma \nabla_{\boldsymbol{\theta}} \mathcal{J}_e$; // $\mathcal{J}_e$
10 **end**
11 **end**
12 Update $\boldsymbol{\theta}$ using $d\boldsymbol{\theta}$;
13 **until** *converge or early stop*;

Fix Logical Form	
z_1	*(hop Nation (filter_num_larger Gold "won" rows))*
$\hat{z}_1$	*(hop Nation (filter_num_larger Gold "2.0" rows))*
Diversify Logical Form	
z_2	*(hop Gold (filter_in Nation "France" rows))*
$\hat{z}_2$	*(hop Silver (filter_in Nation "Turkey" rows))*

Table 1: Examples of fixing and diversifying logical forms. z_i is the original logical form, while $\hat{z}_i$ is the fixed or diversified logical form.

3.3 Fixing and Diversifying Logical Forms

At the early stage of training, mistake logical forms are prone to violating semantic constraints. Consider the logical form z_1 in Table 1. The predicate *filter_num_larger* expects a number as its second argument, but a string *"won"* is given in z_1, thus violating the predicate's semantic constraint. The text generator cannot generate meaningful utterances for such invalid logical forms. Hence, to improve the utilization of mistakes, LFM tries to pinpoint the source of violations and automatically fixes them. For example, z_1 is fixed by replacing *"won"* with a randomly generated number *"2.0"*.

In addition, LFM attempts to enrich the diversity of mistake logical forms by randomly replacing a logical form's entities with proper ones in its associated knowledge base. For example, the logical form $\hat{z}_2$ in Table 1 is generated by (1) replacing the column *Gold* in z_2 with *Silver* which has the same data type with *Gold*; and (2) replacing the value *"France"* in z_2 with another cell value (*"Turkey"*) in the *Nation* column.

We perform this fixing and diversifying procedures for mistake logical forms before synthesizing their utterances (Line 7 in Algorithm 1).

	WikiTQ	WikiSQL	TabFact
Logical Form	✗	✓	✗
Binary Denotation	✗	✗	✓
# Tables	2,108	24,241	16,573
# Examples	18,496	80,654	118,275
Train	11,321	56,355	92,283
Dev	2,831	8,421	12,792
Test	4,344	15,878	12,779

Table 2: Dataset statistics.

4 Experimental Setup

In this section, we present the experimental setup for LFM, including datasets, implementations of the semantic parser and text generator.

4.1 Dataset and Metric

We evaluate LFM on the three challenging semantic parsing datasets in the WSP setting. Table 2 summarizes their statistics and characteristics.

WikiTableQuestions (Pasupat and Liang, 2015) WikiTableQuestions (WikiTQ for short) contains semi-structured tables extracted from Wikipedia and crowdsourced question-answer (utterance-denotation) pairs about the tables. The questions involve a wide variety of operations such as comparisons, superlatives, and aggregations.

WikiSQL (Zhong et al., 2017) WikiSQL is to date the largest dataset for the Text-to-SQL problem. It consists of 24,241 tables extracted from Wikipedia and 80,654 question-SQL pairs. To experiment in the WSP setting, we obtain question-answer pairs by executing each SQL query.

TabFact (Chen et al., 2020) TabFact is a large-scale fact verification dataset with 118,275 examples. Each example consists of an utterance, a Wikipedia table, and a binary label indicating whether the facts described in the utterance are supported by the table. This verification problem can be formulated as a semantic parsing problem: an utterance is entailed if its corresponding logical form executes to *True* on the table. Unlike WikiTQ and WikiSQL, denotations in TabFact are *binary*, and thus, the spuriousness is much more severe.

Metric Following prior WSP work (Liang et al., 2013; Krishnamurthy et al., 2017), we evaluate the performance of a semantic parser via *Denotation Accuracy*: a predicted logical form is considered correct if it executes to the correct denotation.

4.2 Neural Semantic Parser

We develop a simple neural semantic parser for experiments. Given an utterance and a table, the

parser jointly encodes them via an input encoder and generates a logical form with a decoder.

Input Encoder The goal of an input encoder is to obtain distributed representations for a given utterance and table. To achieve the goal, the encoder concatenates the utterance with all columns in the table and jointly encodes them with BERT (Devlin et al., 2019). Since a column may be composed of multiple tokens, the encoder obtains its representation by taking the average of its tokens' representations (Yin et al., 2020). In addition, following prior WSP work (Krishnamurthy et al., 2017; Liang et al., 2018; Wang et al., 2019b), we add binary indicator features to specify (1) whether a token in the utterance appears in the table and (2) whether a column is mentioned in the utterance. These features are mapped to learnable embeddings and concatenated to the output of BERT.

Decoder We employ a grammar-based decoder (Yin and Neubig, 2017) with LSTM cells. It interacts with three types of actions to generate a logical form, namely, APPLYRULE, SELCOLUMN, and SELVALUE. APPLYRULE selects a production rule from the query language's context-free grammar (CFG) and applies it to the abstract syntax tree of a logical form. SELCOLUMN employs a pointer network (Vinyals et al., 2015) to select a column from the table. SELVALUE employs two pointer networks to select a span (beg token, end token) from the utterance. Interested readers can refer to (Yin and Neubig, 2017) for more details about the grammar-based decoder.

Logical Form For WikiTQ and WikiSQL, we use the domain-specific query language proposed by Liang et al. (2018). The language is tailored for answering compositional questions on semi-structured tables. To support TabFact, we extend the query language with the predicates designed by Chen et al. (2020). The query language's CFG is available in Section A.2 of supplementary material.

4.3 Text Generator

We implement the text generator using SCFG. An SCFG consists of a set of production rules: $N \rightarrow \langle \alpha, \beta \rangle$, where N is a non-terminal, and α and β are sequence of terminals and non-terminals. Non-terminals in α and β are aligned. Due to the absence of utterance-logical form pairs, we manually induced the SCFG by composing related utterances for each predicate in the query language and summarizing production rules accordingly. Since α can follow the query language's CFG, we only need to summarize β. About 200 utterances were composed to induce the SCFG. Here is a subset of production rules in the SCFG, which are used to synthesize the canonical utterances for the mistake logical forms (z_4 and z_5) shown in Figure 2.

$$
\begin{aligned}
Root &\rightarrow \langle Project\ ,\ Project \rangle \\
Project &\rightarrow \langle \textit{(hop}\ Col\ Target\textit{)}\ ,\ \text{“which}\ Col\ Target\text{”} \rangle \\
Target &\rightarrow \langle Arg\ ,\ Arg \rangle \mid \langle Retrieve\ ,\ Retrieve \rangle \\
Arg &\rightarrow \langle \textit{(argmin}\ Col\ Filter\textit{)}\ ,\ \text{“has the least}\ Col\ Filter\text{”} \rangle \\
Retrieve &\rightarrow \langle \textit{(first}\ Filter\textit{)}\ ,\ \text{“is listed first}\ Filter\text{”} \rangle \\
Filter &\rightarrow \langle \textit{rows}\ ,\ \text{“”} \rangle \\
Col &\rightarrow \langle \textit{nation}\ ,\ \text{“nation”} \rangle \mid \langle \textit{silver}\ ,\ \text{“silver”} \rangle
\end{aligned}
$$

The production rules of Col are determined by the columns of a given table. Since most predicates are shared among three datasets, we can re-use their production rules in SCFG.

4.4 Implementation

We implement LFM and the semantic parser based on Pytorch (Paszke et al., 2019), AllenNLP (Gardner et al., 2018), and the Transformers library (Wolf et al., 2020). We instantiate $\mathcal{J}_c$ in LFM as REINFORCE.[1] Sample size K is set to 5. γ is set to 1 initially and decays exponentially in each training step. We use the uncased base version of BERT in the parser. We use an AdamW (Loshchilov and Hutter, 2019) optimizer with learning rate 2e-5 and a linear decay scheduler to optimize the parameters in BERT. All remaining parameters are optimized with Adam (Kingma and Ba, 2015) using a constant learning rate 5e-4. At inference time, following prior WSP work (Liang et al., 2018), we apply a beam search of size 5, and we do not use ensemble. For all experiments, we report the averaged denotation accuracy of 5 independent runs. Section A.1 and A.3 in supplementary material provide more details about the implementation and hyperparameters. Our source code are publicly available at `https://github.com/JasperGuo/LFM`.

5 Experimental Result

5.1 Main Results

Table 3 and Table 4 compare the denotation accuracy of our parser (trained using LFM) with previous approaches on WikiTQ and WikiSQL. On

[1] We have tried to instantiate $\mathcal{J}_c$ as MML, but we did not observe significant improvements over REINFORCE.

Weakly Supervised Approach	Dev	Test
w/o Pre-trained LM		
Pasupat and Liang (2015)†	37.0	37.1
Zhang et al. (2017)†	40.4	43.7
Krishnamurthy et al. (2017)‡	42.7	43.3
Liang et al. (2018)† (MAPO)	42.3	43.1
Agarwal et al. (2019)†	43.2	44.1
Dasigi et al. (2019)†	42.1	43.9
Wang et al. (2019b)‡ (STRUCTALIGN)	**43.7**	**44.5**
w/ Pre-trained LM		
Herzig et al. (2020)¶♢	-	48.8
Yin et al. (2020)†♢ (MAPO + BERT)	50.3	49.6
Yin et al. (2020)†♢ (MAPO + TaBERT)	52.2	51.8
Yu et al. (2021)‡♢ (STRUCTALIGN + ROBERTA)	50.7	50.9
Yu et al. (2021)‡♢ (STRUCTALIGN + GRAPPA)	51.9	**52.7**
LFM†♡	**53.6** ±0.4	52.3 ±0.2

Table 3: WikiTQ denotation accuracy. † indicates dynamic approaches, ‡ denotes static approaches, and ¶ denotes approaches without generating logical forms. ♡ and ♢ indicate approaches using base and large pre-trained LM, respectively.

Fully Supervised Approach	Dev	Test
w/ Pre-trained LM		
He et al. (2019)♢	89.5	88.7
Lyu et al. (2020)♢	89.1	89.2
Lin et al. (2020)♢	91.7	91.1
Hui et al. (2021)♢	**91.8**	**91.4**
Weakly Supervised Approach	**Dev**	**Test**
w/o Pre-trained LM		
Liang et al. (2018)† (MAPO)	71.8	72.4
Agarwal et al. (2019)†	74.9	74.8
Wang et al. (2019b)‡ (STRUCTALIGN)	**79.4**	**79.3**
w/ Pre-trained LM		
Min et al. (2019)‡♢	84.4	83.9
Herzig et al. (2020)¶♢	85.1	83.6
Yu et al. (2021)‡♢ (STRUCTALIGN + ROBERTA)	82.3	82.3
Yu et al. (2021)‡♢ (STRUCTALIGN + GRAPPA)	85.9	84.7
Shao et al. (2021)‡♢	85.9	85.6
LFM†♡	**87.4** ±0.2	**86.9** ±0.1

Table 4: WikiSQL denotation accuracy of approaches without execution-guided decoding.

Classification Approach	Dev	Test
w/ Pre-trained LM		
Chen et al. (2020)♡	66.1	65.1
Zhong et al. (2020b)♡	71.8	71.7
Shi et al. (2020)♡	72.5	72.3
Yang et al. (2020)♡	74.9	74.4
Eisenschlos et al. (2020)♢	**81.0**	**81.0**
Semantic Parsing Approach	**Dev**	**Test**
w/ Pre-trained LM		
Chen et al. (2020)†♡ (NSM)	63.2	63.5
Chen et al. (2020)‡♡ (LPA-Ranking)	65.2	65.0
LFM†♡	**68.7** ±0.5	**68.2** ±0.4

Table 5: TabFact denotation accuracy.

WikiTQ, our parser improves the state-of-the-art (SOTA) from 51.9 to 53.6 on the development set, and it performs on par with the SOTA on the test set (52.3 vs. 52.7). On WikiSQL, our parser improves the SOTA from 85.9 to 87.4 on the development set, and from 84.7 to 86.9 on the test set. It is worth noting that previous approaches with pre-trained LM, such as STRUCTALIGN + GRAPPA (Yu et al., 2021) and MAPO + TaBERT (Yin et al., 2020), leveraged a wealth of external corpus to pre-train larger LMs for table understanding. Although our parser only uses the base version of BERT, it still rivals or even outperforms them on both datasets.

Table 5 compares our parser with previous approaches on TabFact. Due to the binary denotations of TabFact, previous approaches can be categorized into two groups: *Semantic Parsing* and *Classification*. While the former generates and executes a logical form to verify the facts described in an utterance, the latter sacrifices the interpretability and directly verifies the facts via a neural classification model. We can observe from the table that our parser significantly surpasses previous semantic parsing approaches, but there is still a large gap compared with the SOTA in classification.

5.2 Analysis

Effect of Learning from Mistakes To obtain an in-depth understanding of LFM, we train the parser without utterance-logical form pairs created from mistakes, which amounts to training with REINFORCE. Table 6 presents the experimental results on three datasets (*w/o Mistake*). We can observe that the parser's performance drops significantly, and it lags behind SOTA approaches presented in Table 3 to Table 5 by a large margin. This result shows that jointly training with utterance-logical form pairs created from mistakes is crucial for LFM to achieve the SOTA. Figure 3 presents the denotation accuracy curves on the development set of WikiTQ. It is clear that the parser trained using LFM bootstraps and converges quickly, while the *w/o Mistake* variant converges much slower and ends up with lower accuracy.

To assess the effectiveness of LFM in dealing with the spuriousness of logical forms, we translate golden SQL queries in the development set of WikiSQL to corresponding logical forms in our query language, and we compare the parser's predictions with golden logical forms. For the *w/o Mistake* variant, among the predictions that execute to correct denotations, 79.0% of them are semantically equivalent to golden logical forms. This number is improved from 79.0% to 87.3%, when the parser is trained using LFM, indicating that LFM can facili-

	WikiTQ	WikiSQL	TabFact
LFM	53.6	87.4	68.7
w/o Mistake	47.2 (−6.4)	79.8 (−7.6)	65.1 (−3.6)
w/o Fixing	53.2 (−0.4)	87.0 (−0.4)	67.5 (−1.2)

Table 6: Ablation study results on development sets.

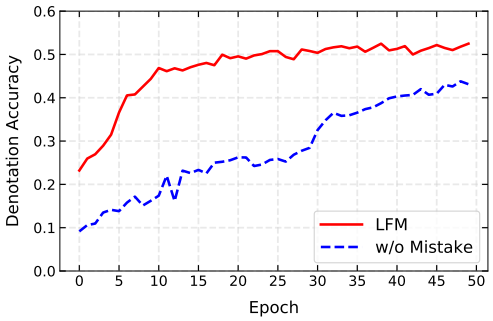

Figure 3: Dev accuracy curves of LFM and the *w/o Mistake* variant on WikiTQ.

tate a parser learning the correct mapping between utterances and logical forms.

Lastly, we ablate the fixing and diversifying mechanism described in Section 3.3 to understand its contribution in LFM. Experimental results are shown in Table 6 (*w/o Fixing*), from which we can observe that this mechanism consistently improves the parser's performance on three datasets.

Comparison with Data Augmentation We compare LFM with other data augmentation techniques for WSP. Prior work explores data augmentation in two primary ways. (1) PRE-TRAIN. Goldman et al. (2018) first pre-train a parser with synthesized utterance-logical form pairs and then finetune the parser via utterance-denotation pairs. (2) JOINT-TRAIN. Guo et al. (2018) obtain extra utterance-denotation pairs from synthesized utterance-logical form pairs, and they jointly train the parser with original utterance-denotation pairs and the extra ones. Both PRE-TRAIN and JOINT-TRAIN synthesize utterance-logical form pairs once in offline, while LFM synthesizes pairs from mistakes in each training step.

In experiments, we synthesize a various number of utterance-logical form pairs using the SCFG (from $\times 0.5$ to $\times 4$ size of training data in WikiTQ).[2] Figure 4 presents the experimental results, from which we can make two main observations. First, PRE-TRAIN improves the parser's accuracy

[2]We randomly sample logical forms in a top-down manner according to the query language's CFG. Sampled logical forms must execute to non-empty denotations. Utterances are then synthesized for each logical form based on the SCFG.

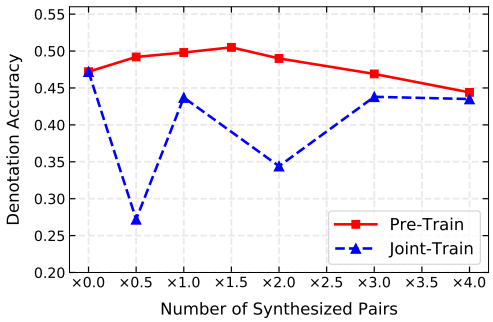

Figure 4: Dev accuracy on WikiTQ with a various number of synthesized utterance-logical form pairs.

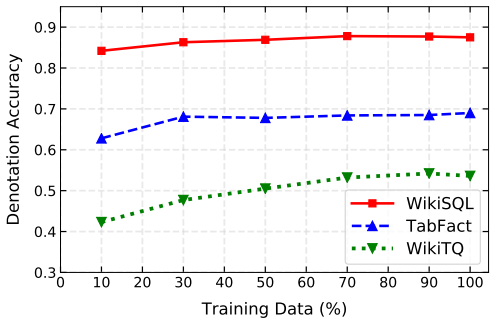

Figure 5: Dev accuracy of parsers trained with a various number of utterance-denotation pairs.

from 47.2 ($\times 0.0$) to 50.5 ($\times 1.5$) on WikiTQ, when $\times 1.5$ size of training data are synthesized for pre-training, but it is still lower than that achieved by LFM (53.6). Adding more synthesized data cannot further improve the accuracy, and it even hurts the performance, which is consistent with the findings in (Shorten and Khoshgoftaar, 2019). Second, JOINT-TRAIN cannot improve the parser's performance, and we observe that training becomes very unstable when less than $\times 2.0$ size of training data are synthesized. These observations suggest that LFM is more effective than the prior data augmentation techniques for WSP.

We also compare a variant of LFM (LFM-RANDOM) that randomly synthesizes utterance-logical forms in each training step rather than synthesizing from mistakes. LFM-RANDOM achieves an accuracy of 53.0 on the development set of WikiTQ, which is slightly worse than LFM. This result suggests that the mistake logical forms generated by a parser during searching serve as a good source for data augmentation.

Data Efficiency Since LFM creates utterance-logical form pairs from mistakes to facilitate training, we study whether it can reduce the need for labeled data (i.e., utterance-denotation pairs).

Figure 5 presents the parser's accuracy achieved by using various proportions of labeled data. On WikiSQL, our parser achieves an accuracy of 84.5 using only 10% of labeled data, which already performs on par with the previous SOTA (85.9). With 70% of labeled data, the parser performs comparably with the one using all labeled data. Similar observations can also be made on TabFact. The parser trained with 30% of labeled data already achieves the SOTA. Adding more labeled data does not bring significant improvements. In terms of WikiTQ, our parser achieves an accuracy of 42.3 using 10% of labeled data, surpassing STRUCTALIGN+GRAPPA (Yu et al., 2021) in the same setting, which achieves 40.7 accuracy. The parser trained with 30% of labeled data performs on par with the *w/o Mistake* variant (47.7 vs. 47.2). Also, with 70% of labeled data, the parser performs comparably with the one using all labeled data. Hence, LFM can substantially reduce the need for labeled data to train a good semantic parser.

6 Conclusion & Future Work

In this work, we present LFM, a simple yet effective dynamic learning framework for WSP. LFM fully utilizes the mistake logical forms generated by a parser during searching to overcome the major challenges in WSP. Experimental results on three semantic parsing datasets show that LFM can effectively address the challenges, and LFM can substantially reduce the need for utterance-denotation pairs to train a good parser.

This work also opens up several avenues for future work. First, further improvements could be made by using a more advanced text generator in LFM. The generator is currently implemented using a hand-crafted SCFG, which often generates unnatural utterances. Second, LFM can be extended to other weakly supervised learning problems where synthesizing inputs (e.g., utterances) from latent variables (e.g., logical forms) is trivial. Consider the problem of learning to solve math word problems via utterance-answer pairs. It is trivial to synthesize an utterance from a math equation. Therefore, LFM could be applied to solve this learning problem.

Acknowledgments

We thank the anonymous reviewers for their helpful discussion and detailed comments. Ting Liu was partially supported by the National Natural Science Foundation of China (61632015, 61772408, 61833015), and the Fundamental Research Funds for the Central Universities.

Impact Statement

Semantic parsing has long been an important paradigm to solve knowledge base question answering problems. In this paper, we present a simple yet effective learning framework to address the major challenges in weakly supervised semantic parsing. This framework is not tied to any neural semantic parser or knowledge base question answering problem. Hence, we consider that there will be no certain societal consequences and ethical aspects caused by our framework.

References

Rishabh Agarwal, Chen Liang, Dale Schuurmans, and Mohammad Norouzi. 2019. Learning to generalize from sparse and underspecified rewards. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 130–140. PMLR.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA. Association for Computational Linguistics.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2020. Tabfact: A large-scale dataset for table-based fact verification. 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.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the world's response. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pages 18–27, Uppsala, Sweden. Association for Computational Linguistics.

Pradeep Dasigi, Matt Gardner, Shikhar Murty, Luke Zettlemoyer, and Eduard Hovy. 2019. Iterative search for weakly supervised semantic parsing. 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

2669–2680, Minneapolis, Minnesota. Association for Computational Linguistics.

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, Minneapolis, Minnesota. Association for Computational Linguistics.

Julian Eisenschlos, Syrine Krichene, and Thomas Müller. 2020. Understanding tables with intermediate pre-training. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 281–296, Online. Association for Computational Linguistics.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A deep semantic natural language processing platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 1–6, Melbourne, Australia. Association for Computational Linguistics.

Attilio Giordana and Alessandro Serra. 2001. *Learning from Mistakes*, pages 89–102. Springer US, Boston, MA.

Omer Goldman, Veronica Latcinnik, Ehud Nave, Amir Globerson, and Jonathan Berant. 2018. Weakly supervised semantic parsing with abstract examples. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1809–1819, Melbourne, Australia. Association for Computational Linguistics.

Daya Guo, Yibo Sun, Duyu Tang, Nan Duan, Jian Yin, Hong Chi, James Cao, Peng Chen, and Ming Zhou. 2018. Question generation from SQL queries improves neural semantic parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1597–1607, Brussels, Belgium. 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.

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

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online. Association for Computational Linguistics.

Binyuan Hui, Xiang Shi, Ruiying Geng, Binhua Li, Yongbin Li, Jian Sun, and Xiaodan Zhu. 2021. Improving text-to-sql with schema dependency learning. *arXiv preprint arXiv:2103.04399*.

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.

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.

Diederick P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

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.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural symbolic machines: Learning semantic parsers on Freebase with weak supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23–33, Vancouver, Canada. Association for Computational Linguistics.

Chen Liang, Mohammad Norouzi, Jonathan Berant, Quoc Le, and Ni Lao. 2018. Memory augmented policy optimization for program synthesis and semantic parsing. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 10015–10027, Red Hook, NY, USA. Curran Associates Inc.

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.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Qin Lyu, Kaushik Chakrabarti, Shobhit Hathi, Souvik Kundu, Jianwen Zhang, and Zheng Chen. 2020. Hybrid ranking network for text-to-sql. *arXiv preprint arXiv:2008.04759*.

Sewon Min, Danqi Chen, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2019. 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.

Lili Mou, Zhengdong Lu, Hang Li, and Zhi Jin. 2017. Coupling distributed and symbolic execution for natural language queries. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2518–2526. PMLR.

Arvind Neelakantan, Quoc V. Le, Martin Abadi, Andrew McCallum, and Dario Amodei. 2017. Learning a natural language interface with neural programmer. In *International Conference on Learning Representations*.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1470–1480, Beijing, China. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Zhihong Shao, Lifeng Shang, Qun Liu, and Minlie Huang. 2021. A mutual information maximization approach for the spurious solution problem in weakly supervised question answering. 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 4111–4124, Online. Association for Computational Linguistics.

Qi Shi, Yu Zhang, Qingyu Yin, and Ting Liu. 2020. Learn to combine linguistic and symbolic information for table-based fact verification. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 5335–5346, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Bailin Wang, Ivan Titov, and Mirella Lapata. 2019a. 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)*, pages 3774–3785, Hong Kong, China. Association for Computational Linguistics.

Bailin Wang, Wenpeng Yin, Xi Victoria Lin, and Caiming Xiong. 2021. 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.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019b. Learning deep transformer models for machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1810–1822, Florence, Italy. Association for Computational Linguistics.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Xiaoyu Yang, Feng Nie, Yufei Feng, Quan Liu, Zhigang Chen, and Xiaodan Zhu. 2020. Program enhanced fact verification with verbalization and graph attention network. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7810–7825, Online. 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, Vancouver, Canada. Association for Computational Linguistics.

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.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, bailin wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, richard socher, and Caiming Xiong. 2021. GraPPa: Grammar-augmented pre-training for table semantic parsing. In *International Conference on Learning Representations*.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018. 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.

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. 2020a. 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. *CoRR*, abs/1709.00103.

Wanjun Zhong, Duyu Tang, Zhangyin Feng, Nan Duan, Ming Zhou, Ming Gong, Linjun Shou, Daxin Jiang, Jiahai Wang, and Jian Yin. 2020b. LogicalFactChecker: Leveraging logical operations for fact checking with graph module network. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6053–6065, Online. Association for Computational Linguistics.

A Supplementary Material

A.1 Implementation Details

Data Pre-processing Following Wang et al. (2019b), we identify mentions of entities (including table names, column names, and cell values) in an utterance via a string-match based method. We also identify numbers and dates in both utterances and tables using the CoreNLP toolkit. The identification results are converted to indicator features for the input encoder, as described in Section 4.2.

Sampling Logical Forms Since we instantiate $\mathcal{J}_c$ as REINFORCE in experiments, we use the Monte Carlo sampling method to sample logical forms. During sampling, when the current action is SELVALUE, the sampled span is constrained to be mentions of cell values, numbers, or dates in an utterance. In this way, the parser is more likely to sample logical forms that meet the query language's semantic constraints. Such a constraint has been widely used in prior WSP approaches to search consistent logical forms (Wang et al., 2019b; Min et al., 2019). Note that this constraint is not used during training and testing.

A.2 Context-Free Grammar

We present the context-free grammar (CFG) of our query language for each dataset. It is used in the grammar-based decoder to generate a logical form. All Col and $Value$ rules in the three CFGs are determined by a given table and utterance.

WikiSQL

$Root \rightarrow Project \mid Meta$
$Project \rightarrow$ *(hop* $Col\ Target$*)*
$Meta \rightarrow$ *(count* $Col\ Target$*)*
$Meta \rightarrow$ *(max* $Col\ Target$*)*
$Meta \rightarrow$ *(min* $Col\ Target$*)*
$Meta \rightarrow$ *(sum* $Col\ Target$*)*
$Meta \rightarrow$ *(average* $Col\ Target$*)*
$Target \rightarrow$ *rows*
$Target \rightarrow Filter$
$Target \rightarrow$ *(intersect* $Filter\ Filter$*)*
$Target \rightarrow$ *(intersect* $Filter$ *(intersect* $Filter\ Filter$*))*
$Target \rightarrow$ *(intersect* $Filter$ *(intersect* $Filter$ *(intersect* $Filter\ Filter$*)))*
$Filter \rightarrow$ *(filter_in* $Col\ Value\ rows$*)*
$Filter \rightarrow$ *(filter_number_less* $Col\ Value\ rows$*)*
$Filter \rightarrow$ *(filter_number_larger* $Col\ Value\ rows$*)*
$Filter \rightarrow$ *(filter_number_equals* $Col\ Value\ rows$*)*
$Col \rightarrow$ *nation* | *silver*
$Value \rightarrow$ "franece" | "turkey"

TabFact

$Root \rightarrow CmpDate \mid CmpNumber \mid CmpString \mid CmpPosition \mid BoolLogic$
$CmpDate \rightarrow$ *(date_greater* $Date\ Date$*)*
$CmpDate \rightarrow$ *(date_equals* $Date\ Date$*)*
$CmpDate \rightarrow$ *(date_not_equals* $Date\ Date$*)*
$CmpDate \rightarrow$ *(all_date_equals* $Col\ Value\ Target$*)*
$CmpDate \rightarrow$ *(all_date_not_equals* $Col\ Value\ Target$*)*
$CmpDate \rightarrow$ *(all_date_greater* $Col\ Value\ Target$*)*
$CmpDate \rightarrow$ *(all_date_greater_equals* $Col\ Value\ Target$*)*
$CmpDate \rightarrow$ *(all_date_less* $Col\ Value\ Target$*)*
$CmpDate \rightarrow$ *(all_date_less_equals* $Col\ Value\ Target$*)*
$Date \rightarrow MinMax \mid Hop \mid Value$
$CmpNumber \rightarrow$ *(num_greater* $Number\ Number$*)*
$CmpNumber \rightarrow$ *(num_equals* $Number\ Number$*)*
$CmpNumber \rightarrow$ *(num_not_equals* $Number\ Number$*)*
$CmpNumber \rightarrow$ *(all_num_equals* $Col\ Value\ Target$*)*
$CmpNumber \rightarrow$ *(all_num_not_equals* $Col\ Value\ Target$*)*
$CmpNumber \rightarrow$ *(all_num_greater* $Col\ Value\ Target$*)*
$CmpNumber \rightarrow$ *(all_num_greater_equals* $Col\ Value\ Target$*)*
$CmpNumber \rightarrow$ *(all_num_less* $Col\ Value\ Target$*)*
$CmpNumber \rightarrow$ *(all_num_less_equals* $Col\ Value\ Target$*)*
$Number \rightarrow MinMax \mid Hop \mid Agg \mid CountRow \mid Value$
$CmpString \rightarrow$ *(is_empty* $Col\ Target$*)*
$CmpString \rightarrow$ *(str_equals* $Hop\ Value$*)*
$CmpString \rightarrow$ *(str_not_equals* $Hop\ Value$*)*
$CmpString \rightarrow$ *(mode_equals* $Col\ Value\ Target$*)*
$CmpString \rightarrow$ *(mode_not_equals* $Col\ Value\ Target$*)*
$CmpString \rightarrow$ *(all_str_equals* $Col\ Value\ Target$*)*
$CmpString \rightarrow$ *(all_str_not_equals* $Col\ Value\ Target$*)*
$Hop \rightarrow$ *(hop* $Col\ Target$*)*
$MinMax \rightarrow$ *(max* $Col\ Target$*)* | *(min* $Col\ Target$*)*
$CountRow \rightarrow$ *(count_distinct* $Col\ Target$*)*
$CountRow \rightarrow$ *(count* $Target$*)* | *(half* $Target$*)* | *(one_third* $Target$*)*
$Agg \rightarrow$ *(sum* $Col\ Target$*)* | *(average* $Col\ Target$*)*
$Agg \rightarrow$ *(diff* $Col\ Target\ Target$*)*
$Target \rightarrow Arg \mid Filter \mid$ *rows*
$Filter \rightarrow$ *(union* $Filter\ Filter$*)*
$Filter \rightarrow$ *(intersect* $Filter\ Filter$*)*
$Filter \rightarrow$ *(filter_in* $Col\ Value\ rows$*)*
$Filter \rightarrow$ *(filter_not_in* $Col\ Value\ rows$*)*
$Filter \rightarrow$ *(filter_number_less* $Col\ Value\ rows$*)*
$Filter \rightarrow$ *(filter_number_less_equals* $Col\ Value\ rows$*)*
$Filter \rightarrow$ *(filter_number_larger* $Col\ Value\ rows$*)*
$Filter \rightarrow$ *(filter_number_larger_equals* $Col\ Value\ rows$*)*
$Filter \rightarrow$ *(filter_number_equals* $Col\ Value\ rows$*)*
$Filter \rightarrow$ *(filter_date_less* $Col\ Value\ rows$*)*
$Filter \rightarrow$ *(filter_date_less_equals* $Col\ Value\ rows$*)*
$Filter \rightarrow$ *(filter_date_larger* $Col\ Value\ rows$*)*
$Filter \rightarrow$ *(filter_date_larger_equals* $Col\ Value\ rows$*)*
$Filter \rightarrow$ *(filter_date_equals* $Col\ Value\ rows$*)*
$Arg \rightarrow$ *(argmax* $Col\ Filter$*)*
$Arg \rightarrow$ *(argmin* $Col\ Filter$*)*
$Col \rightarrow$ *nation* | *silver*
$Value \rightarrow$ "franece" | "turkey"

WikiTableQuestions

$Root \rightarrow Project \mid Meta$
$Project \rightarrow$ *(hop* $Col\ Target$*)*
$Project \rightarrow$ *(mode* $Col\ Target$*)*
$Meta \rightarrow$ *(count* $Col\ Target$*)*
$Meta \rightarrow$ *(max* $Col\ Target$*)*
$Meta \rightarrow$ *(min* $Col\ Target$*)*
$Meta \rightarrow$ *(sum* $Col\ Target$*)*
$Meta \rightarrow$ *(average* $Col\ Target$*)*
$Meta \rightarrow$ *(diff* $Col\ Target\ Target$*)*
$Meta \rightarrow$ *(column_diff* $Col\ Col\ Target$*)*
$Target \rightarrow Filter \mid Retrieve \mid Arg \mid Consecutive$
$Filter \rightarrow$ *rows*
$Filter \rightarrow$ *(union* $Filter\ Filter$*)*
$Filter \rightarrow$ *(intersect* $Filter\ Filter$*)*
$Filter \rightarrow$ *(filter_empty* Col *rows)*
$Filter \rightarrow$ *(filter_not_empty* Col *rows)*
$Filter \rightarrow$ *(filter_in* $Col\ Value$ *rows)*
$Filter \rightarrow$ *(filter_not_in* $Col\ Value$ *rows)*
$Filter \rightarrow$ *(filter_number_less* $Col\ Value$ *rows)*
$Filter \rightarrow$ *(filter_number_less_equals* $Col\ Value$ *rows)*
$Filter \rightarrow$ *(filter_number_larger* $Col\ Value$ *rows)*
$Filter \rightarrow$ *(filter_number_larger_equals* $Col\ Value$ *rows)*
$Filter \rightarrow$ *(filter_number_equals* $Col\ Value$ *rows)*
$Filter \rightarrow$ *(filter_date_less* $Col\ Value$ *rows)*
$Filter \rightarrow$ *(filter_date_less_equals* $Col\ Value$ *rows)*
$Filter \rightarrow$ *(filter_date_larger* $Col\ Value$ *rows)*
$Filter \rightarrow$ *(filter_date_larger_equals* $Col\ Value$ *rows)*
$Filter \rightarrow$ *(filter_date_equals* $Col\ Value$ *rows)*
$Retrieve \rightarrow$ *(previous* $Filter$*)*
$Retrieve \rightarrow$ *(next* $Filter$*)*
$Retrieve \rightarrow$ *(first* $Filter$*)*
$Retrieve \rightarrow$ *(last* $Filter$*)*
$Arg \rightarrow$ *(argmax* $Col\ Filter$*)*
$Arg \rightarrow$ *(argmin* $Col\ Filter$*)*
$Consecutive \rightarrow$ *(consecutive* $Filter$*)*
$Col \rightarrow$ *nation* | *silver*
$Value \rightarrow$ "franece" | "turkey"

A.3 Hyper-Parameters

Table 7 lists the hyper-parameters of our neural semantic parser on three datasets. Most hyper-parameters are shared among three datasets. Experimental results reported in Section 5 are averaged over 5 random runs using seeds $\{100, 200, 300, 400, 500\}$. Experiments are conducted on P40 GPUs with 24GB memory.

Parameter	WikiTQ	WikiSQL	TabFact
Input Encoder			
Pre-train LM	BERT-base	BERT-base	BERT-base
Indicator Feature Embedding Size	10	10	10
Input Encoder Output Size	256	256	256
Use Indicator Features for Synthesized Data	✓	✓	✗
Grammar-based Decoder			
Decoder Layer	1	1	1
Decoder RNN Cell	LSTM	LSTM	LSTM
Decoder Input Dropout	0.5	0.5	0.5
Decoder Hidden Size	256	256	256
Production Rule Embedding Size	256	256	256
Non-Terminal Embedding Size	64	64	64
Decoder Attention Hidden Size	128	128	128
APPLYRULE Classifier Dropout	0.2	0.2	0.2
SELCOLUMN Pointer Network Hidden Size	128	128	128
SELCOLUMN Pointer Network Dropout	0.5	0.5	0.5
SELVALUE Pointer Network Hidden Size	128	128	128
SELVALUE Pointer Network Dropout	0.5	0.5	0.5
Maximum Decode Step	25	25	30
LFM			
K	5	5	5
γ	1.0	1.0	1.0
γ decay rate	5e-5	5e-5	2e-4
Others			
Epsilon Greedy Rate for Sampling	0.15	0.15	0.0
Batch Size	16	64	64
Learning Rate	5e-4	5e-4	5e-4
Pre-Train LM Learning Rate	2e-5	2e-5	2e-5
Pre-Train LM Learning Rate Freeze Step	3000	1500	0
Pre-Train LM Learning Rate Warmup Step	1500	1500	518
Entropy Regularization	✗	✗	✓
Entropy Regularization Weight	-	-	0.1
Gradient Clip	5	5	5
Beam Size	5	5	5

Table 7: Hyper-Parameters of our neural semantic parser on three datasets.