Keep the Structure: A Latent Shift-Reduce Parser for Semantic Parsing

Yuntao Li[1]*, Bei Chen[2], Qian Liu[3]*, Yan Gao[2], Jian-Guang Lou[2], Yan Zhang[1], Dongmei Zhang[2]
[1]Peking University
[2]Microsoft Research
[3]Beihang University
{li.yt, zhyzhy001}@pku.edu.cn; qian.liu@buaa.edu.cn
{beichen, yan.gao, jlou, dongmeiz}@microsoft.com

Abstract

Traditional end-to-end semantic parsing models treat a natural language utterance as a holonomic structure. However, hierarchical structures exist in natural languages, which also align with the hierarchical structures of logical forms. In this paper, we propose a latent shift-reduce parser, called LASP, which decomposes both natural language queries and logical form expressions according to their hierarchical structures and finds local alignment between them to enhance semantic parsing. LASP consists of a base parser and a shift-reduce splitter. The splitter dynamically separates an NL query into several spans. The base parser converts the relevant simple spans into logical forms, which are further combined to obtain the final logical form. We conducted empirical studies on two datasets across different domains and different types of logical forms. The results demonstrate that the proposed method significantly improves the performance of semantic parsing, especially on unseen scenarios.

1 Introduction

Semantic parsing plays a key role in online querying systems. The purpose of semantic parsing is to convert a natural language (NL) query into a machine-interpretable meaning representation, which is also called logical form (LF). As a trend in the last few years, neural semantic parsers have been widely studied. The structural basis of a neural parser is a sequence-to-sequence model, which uses an encoder to comprehend NL sentences and a decoder to generate their corresponding LFs [Jia and Liang, 2016; Dong *et al.*, 2018; Shaw *et al.*, 2019]. Though these methods show high accuracies on simple queries, they tend to show lower performance on relatively complex queries [Talmor and Berant, 2018]. On the one hand, a complex query shows higher diversity on both meaning and sentence structure, which requires high generalization ability of encoder models to understand the structure and long-term dependencies correctly. On the other hand, generating long LFs is much harder than generating short LFs, which demands high performance of decoder models to produce complex LFs at once.

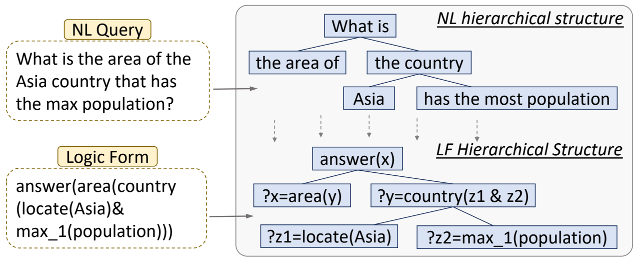
NL Query
What is the area of the Asia country that has the max population?
Logic Form
answer(area(country (locate(Asia)& max_1(population)))
NL hierarchical structure
What is
the area of
the country
Asia
has the most population
LF Hierarchical Structure
answer(x)
?x=area(y)
?y=country(z1 & z2)
?z1=locate(Asia)
?z2=max_1(population)

Figure 1: A motivating example to illustrate the local alignment between hierarchical structures of NL and LF, which is found by our model LASP to facilitate better semantic parsing.

On consideration of the well-defined hierarchical structure of logical form representations, prior studies explored the possibility of generating logical forms hierarchically, e.g. via tree-based decoding [Dong and Lapata, 2016] or coarse-to-fine decoding [Dong and Lapata, 2018]. These methods free semantic parsers from making long-sequence predictions at once, resulting in a simpler decoding procedure. However, the difficulty of understanding complex queries with long-term dependency still exists.

Similar to a logical form representation, the underlying structure of a natural language utterance is also hierarchically organized, i.e., some smaller spans are nested within larger spans. For example, as shown in Figure 1, the complex query "What is the area of the Asia country that has the max population?" can be split into several spans according to its hierarchical structure. By parsing each span into sub LF separately and composing them under the same hierarchical structure, a complex LF can be obtained. When such local alignment exists between the hierarchical structures of NL queries and LFs, it is possible to reduce the burden of semantic parsing by adding this kind of alignment to the parsing process.

In this paper, we propose a novel neural **La**tent **S**hift-Reduce **P**arser (LASP) for semantic parsing, which explicitly models the local alignment between hierarchical structures of NL queries and LFs. Given an NL query, LASP first uses a shift-reduce splitter to read tokens by SHIFT operations and detect the hierarchical structure. Once a *semantic span* is detected, LASP takes a REDUCE operation to convert it into a

*Work done during an internship at Microsoft Research.

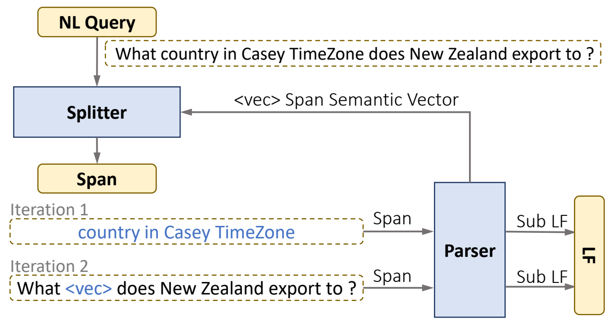
NL Query
What country in Casey TimeZone does New Zealand export to ?
Splitter
<vec> Span Semantic Vector
Span
Iteration 1
country in Casey TimeZone
Span
Iteration 2
What <vec> does New Zealand export to ?
Span
Parser
Sub LF
Sub LF
LF

Figure 2: Illustration of LASP with an example from ComplexWQ.

sub LF with a base parser and uses a *span semantic vector* to replace the original span. This process goes iteratively until the whole NL query is parsed. In contrast to traditional shift-reduce parsers which require expert-designed lexicons and grammars, our proposed LASP is a latent model that largely weakens the reliance on supervised training data or expert rules.

Our proposed LASP has three advantages: (1) It decomposes a complex NL query into several spans according to its hierarchical structure, and the structure also locally aligns with that of LF. This decomposition diminishes the demand for complex model designing and numerous training data. (2) It is a latent shift-reduce parsing model. The only supervision needed is the complete utterances, not the detailed splitting points. (3) The base parser and shift-reduce splitter are optimized iteratively during training. The two modules promote each other during the training process to achieve self-adaptation. We conduct experiments on two datasets that show improvement against the state-of-the-art parsers.

2 Methodology

2.1 Latent Shift-Reduce Parser

Latent **S**hift-Reduce **P**arser (**LASP**) is a compositional semantic parsing approach where a complex natural language (NL) query can be decomposed into multiple spans and solved separately. The LASP consists of two modules, i.e., a **shift-reduce splitter** transforms a sequence of NL tokens into several NL spans in a hierarchical tree structure, and a **base parser** converts an NL span into an LF representation. Figure 2 shows the illustration of LASP with a real example. Given an NL query, the shift-reduce splitter reads it token by token and detects whether a semantic-complete *span* is loaded. Once a span is detected out, it is fed into the base parser to produce the corresponding *sub logical form* (sub LF) and *span semantic vector*. The span semantic vector stands for the semantic meaning of the span and takes the place of the span in the original NL query. For example, in Figure 2, "<vec>" takes the place of the span "country in Casey TimeZone" in the next iteration. This process goes iteratively until the whole NL query is parsed. At last, all the sub LFs are combined to generate the complete LF of the original NL query. Next, we will introduce the simple base parser in Section 2.2 and the shift-reduce splitter in Section 2.3 in detial.

2.2 Base Parser

Let $\mathbf{x} = (x_1, x_2, \cdots, x_N)$ denote the sequence of an NL span. The element x_n can either be an NL token or a span semantic vector from the previous iteration. The base parser aims at converting $\mathbf{x}$ into its corresponding logical form $\mathbf{y}$ and generating a new span semantic vector $\mathbf{v}$ to represent $\mathbf{x}$.

For spans consisting of NL tokens only, we employ a simple sequence-to-sequence model as the base parser. At first, the span $\mathbf{x}$ passes by a token embedding layer and the output embedding vectors are denoted as $\mathbf{u} = (\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_N)$. On top of the token embedding layer, we use a bi-directional Recurrent Neural Network (RNN) as the encoder, whose basic unit is Gated Recurrent Units (GRU). The span semantic vector $\mathbf{v}$ is produced by the encoder as an abstract semantic representation of the span $\mathbf{x}$:

$$\mathbf{v} = \mathbf{BiGRU}_{\text{enc}}(\mathbf{u}). \tag{1}$$

Another GRU model is used as the decoder and generates the logical form $\mathbf{y}$:

$$\mathbf{y} = \mathbf{GRU}_{\text{dec}}(\mathbf{v}). \tag{2}$$

As mentioned, due to the compositional structure of LASP, the base parser is designed to support span semantic vector as input in addition to NL tokens. For the sake of simplicity, we leverage an MLP layer to transfer the span semantic vectors into a hidden vector, which has the same dimension as token embedding vectors.

2.3 Shift-Reduce Splitter

We employ a shift-reduce splitter to capture the compositionality of NL queries and decompose them into spans. Shift-reduce parsing algorithm is capable of transforming a sequence of NL tokens into a tree that represents its hierarchical structure. This structure aligns with the structure of its corresponding LF. The shift-reduce splitter splits a complex NL query into several spans according to its hierarchical structure, and each of them is parsed separately.

Our proposed shift-reduce splitter takes the form of a generative model, which predicts an action sequence given an NL query. Similar to a standard shift-reduce parsing algorithm, our presented shift-reduce splitter is initialized with an empty stack ($\mathbf{S}$) and a buffer ($\mathbf{B}$) containing all tokens of the NL query. In addition, we use a sub LF list ($\mathbf{L}$) to store all parsed sub LFs. At each step, the shift-reduce splitter chooses an action among the three ones:

- SHIFT pops out the first token from the buffer $\mathbf{B}$ and pushes it into the stack $\mathbf{S}$.
- REDUCE(k) pops out k tokens from the top of the stack $\mathbf{S}$ to form a span, where k is a parameter to be predicted. We set the maximum length of a span to be K. This span is parsed by the base parser to a sub LF, which is stored in the sub LF list $\mathbf{L}$.
- FIN is a shortcut to REDUCE, which pops out all elements from the stack $\mathbf{S}$ and finishes the shift-reduce process. It can break the span length limitation of REDUCE.

The shift-reduce splitter takes action step by step until the buffer is empty and only one vector is left in the stack. Then

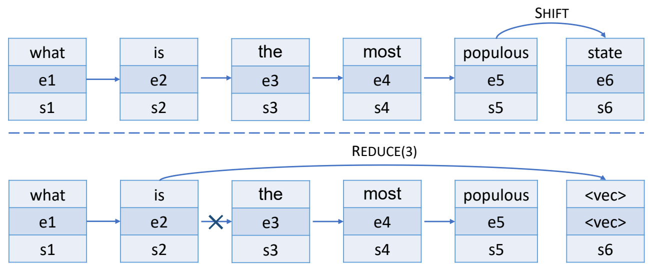

Figure 3: Stack's behaviours when taking SHIFT and REDUCE operations by the shift-reduce splitter, with the full query being "what is the most populous state in the US".

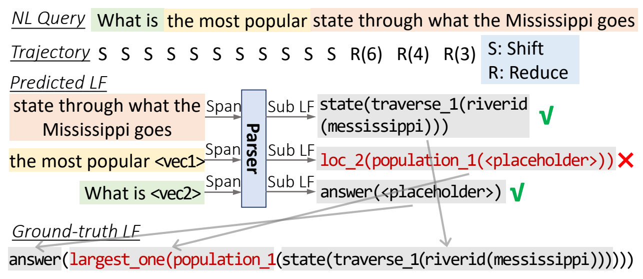
NL Query
What is the most popular state through what the Mississippi goes
Trajectory S S S S S S S S S S S R(6) R(4) R(3)
S: Shift
R: Reduce
Predicted LF
state through what the Mississippi goes
the most popular <vec1>
What is <vec2>
Span
Parser
Sub LF
state(traverse_1(riverid (messissippi)))
loc_2(population_1(<placeholder>))
answer(<placeholder>)
Ground-truth LF
answer(largest_one(population_1(state(traverse_1(riverid(messissippi))))))

Figure 4: An example of trajectory search.

we get a sequence of actions $\mathbf{a} = (a_1, a_2, \cdots, a_T)$, called a *trajectory*.

Besides NL tokens themselves, both stack $\mathbf{S}$ and buffer $\mathbf{B}$ stores a meaning representation vector for each token. Let $\mathbf{w} = (w_1, w_2, \cdots, w_M)$ denotes the token sequence of the NL query. We use a BiGRU model as encoder to convert NL tokens into their meaning representation vectors $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2, \cdots, \mathbf{e}_M)$:

$$\mathbf{e} = \mathbf{BiGRU}_{\text{splitter}}(\mathbf{w}), \tag{3}$$

which helps each token to absorb contextual information of the whole query.

Additionally, at step t, a state vector $\mathbf{s}_t$ is stored by the stack $\mathbf{S}$ to represent its current state. Due to the dynamics of $\mathbf{S}$, a StackGRU is adopted to compute this state vector. A StackGRU is an extension of a standard GRU model, which supports dynamic changes caused by the push and pop operations of a standard stack. Specifically, if a SHIFT operation is taken, the new hidden state $\mathbf{s}_t$ is updated by the last hidden state $\mathbf{s}_{t-1}$ in the stack (i.e., $\mathbf{s}_5$ in Figure 3 top) and the pushed NL token (i.e., $\mathbf{e}_6$ in Figure 3 top). If a REDUCE(k) is taken, we firstly pop out k tokens from the stack and summarize them into a span semantic vector $\mathbf{v}$. Then, StackGRU takes the last hidden state in the stack $\mathbf{s}_{-1}$ (i.e., $\mathbf{s}_2$ in Figure 3 bottom) and the span semantic vector $\mathbf{v}$ to update the new stack state $\mathbf{s}_t$, where the subscript -1 indicates the last (top) element of the stack.

$$\mathbf{s}_t = \begin{cases} \mathbf{StackGRU}\left(\mathbf{s}_{-1}, \mathbf{e}^t\right) & \text{if } a_t = \text{SHIFT} \\ \mathbf{StackGRU}\left(\mathbf{s}_{-1}, \mathbf{v}\right) & \text{if } a_t = \text{REDUCE} \end{cases}, \tag{4}$$

As a result, for a SHIFT action, we form a new stack element as (w^t, $\mathbf{e}^t$, $\mathbf{s}_t$), where w^t denotes the NL token that enters the stack at step t and $\mathbf{e}^t$ is its representation vector. For a REDUCE action, we form the new element as (<placeholder>, $\mathbf{v}$, $\mathbf{s}_t$). Then, the newly generated stack element is pushed into the top of the stack.

To decide which action to take at each step, we adopt a two-stage process for the shift-reduce splitter: firstly, the splitter selects an action among SHIFT, REDUCE and FIN; and secondly, the reducing length k is decided if the action REDUCE is selected in the first stage. We will introduce them in turn.

Stage 1. At step t, the action a_t is predicted based on the information from both the stack $\mathbf{S}$ and the buffer $\mathbf{B}$. We concatenate three vectors to obtain a state representation vector $\mathbf{z}_t$ that represents the current state of the shift-reduce splitter: the hidden state of the stack's last element ($\mathbf{s}_{-1}$), the token representation vector of the stack's last element ($\mathbf{e}_{-1}$) and the token representation vector of the buffer's first element ($\mathbf{e}'$). Then, the state representation vector is transformed with a multilayer perceptron ($\mathbf{MLP}$) to decide this high-level operation.

$$\begin{aligned} \mathbf{z}_t &= [\mathbf{s}_{-1}; \mathbf{e}_{-1}; \mathbf{e}'] , \\ a_t &= \text{argmax}\left(\mathbf{MLP}\left(\mathbf{z}_t\right)\right). \end{aligned} \tag{5}$$

Stage 2. The second stage is to decide the length of span to reduce if the REDUCE action is chosen in the first stage. We adopt an attention mechanism to decide the splitting position, whose key is the state representation vector $\mathbf{z}_t$ and values are the vectors related to each token stored in the stack $\mathbf{S}$:

$$\begin{aligned} \mathbf{r}_{-i} &= [\mathbf{e}_{-i}; \mathbf{s}_{-i}] , \\ \mathbf{p} &= (p_{-K}, \cdots, p_{-3}, p_{-2}) \\ &= \mathbf{Attn}\left(\mathbf{z}_t, [\mathbf{r}_{-K}, \cdots, \mathbf{r}_{-3}, \mathbf{r}_{-2}]\right), \end{aligned} \tag{6}$$

where K is the maximum length of the REDUCE operation, $\mathbf{e}_{-i}$ is the vector of the penultimate i-th token in the stack, $\mathbf{s}_{-i}$ is the corresponding state vector, $\mathbf{Attn}$ is the attention mechanism (dot attention in our experiments), and p_{-i} is the probability of reducing a span with length i. Notice that if K is larger than the length of $\mathbf{S}$, we set $K = |\mathbf{S}|$.

3 Training

The LASP consists of two sub modules, i.e., a base parser and a shift-reduce splitter. The hard decision made by shift-reduce splitter makes it hard to train these models together with back propagation directly. We propose an EM training approach to solve this problem, which trains these two modules separately and iteratively. Another difficulty of training these two modules is the lack of span-level supervised data. Our key solution to this obstacle is to find the correct trajectories with high confidence. We design a *trajectory search* approach to help find possible correct splitting and sub LFs for each NL query and use a trajectory memory to store them. We then use these stored trajectories to dump pseudo training data and optimize the two sub modules of LASP.

3.1 Trajectory Search

A trajectory is a sequence of actions produced by the shift-reduce splitter by running on an NL query. Taking the query shown in Figure 4 as an example, given a trajectory corresponded with an NL query, the query can be split into several parts with corresponding predict LF representations. By comparing these predicted sub LFs with the ground truth LF, we can determine if the trajectory has a high probability of being correct, as well as possibly fix parsing mistakes, as the sub LF in red in Figure 4.

Trajectory search aims to search for global-high-probability trajectories. On finding possible trajectories during trajectory decoding, we adopt a back-tracing-based searching algorithm. When predicting a single action at each step, apart from the most possible operation decided by the shift-reduce splitter, we also add some operations with less probability into possible candidate trajectories. Specifically, if the predicted operation is a SHIFT, we also search a trajectory with this step being REDUCE if the probability of REDUCE is above a threshold τ; if a REDUCE operation is predicted, we search for the top 2 possible span length to reduce and add them to possible trajectory list. These trajectories are stored in a trajectory memory, which is later used for the training process of the base parser and shift-reduce splitter.

3.2 Base Parser Training

Having those searched trajectories stored in the memory, we first extract training pairs of NL spans and corresponding sub LFs from them. For an NL query with its corresponding trajectories, we compare the predicted sub LFs with the ground truth LF and fix possible mistakes of each sub LF. Concretely, we compare the predicted sub LFs in the ground-truth LF from both inside-out and outside-in directions, as the example shown in Figure 4. If the ground-truth LF matches exactly with a combination of all sub LFs, we assume this NL query is correctly parsed and the predictions from NL spans to sub LFs are fully correct. Otherwise, the left unmatched NL span and LF are formed as a span-level pair.

The base parser is trained with above span-level pairs of NL queries and sub LFs, as well as each NL query without splitting. Note that we use a hierarchical dynamic training procedure, i.e. spans are dynamically split out during training, so that the loss from outsider LF prediction can also influence the span encoding vector generated by insider span through back propagation. Besides, the supervised span-level training data for the base parser is changed during training iterations w.r.t to the shift-reduce splitter and searched trajectories, so that the base parser training helps the parser to fit the splitting pattern of the shift-reduce splitter.

3.3 Shift-Reduce Splitter Training

The shift-reduce splitter is also trained with the searched trajectories. In this scenario, the selection of the best trajectory $\hat{\mathbf{a}} = (\hat{a}_1, \hat{a}_2, \cdots, \hat{a}_T)$ can be modeled as a latent variable. In a unsupervised setting where the ground-truth is not available, a maximum marginal likelihood (MML) estimation can be calculated. MML marginalizes the likelihood of each trajectory $\mathbf{a}$ given the NL query $\mathbf{w}$ with respect to the splitter parameters θ.

$$P(\mathbf{y}|\mathbf{w};\theta) = \sum_{\mathbf{a}\in\mathcal{A}} P(\mathbf{y}|\mathbf{a})P(\mathbf{a}|\mathbf{w};\theta) = \sum_{\mathbf{a}\in\mathcal{A}} P(\mathbf{a}|\mathbf{w};\theta),$$
$$J_{MML}(\theta|\mathbf{w},\mathcal{A}) = -\log \sum_{\mathbf{a}\in\mathcal{A}} P(\mathbf{a}|\mathbf{w};\theta), \tag{7}$$

where $\mathcal{A}$ denotes for all possible trajectories.

However, there are two major problems in the MML objective in training such a model. On the one hand, the searching space of possible action trajectories can be very large, which grows exponentially as sentence length grows. This issue makes it impossible to discover every possible trajectory. On the other hand, exploring only a subset of action trajectories cannot ensure the correct trajectory being selected, and the searching process is time consuming. In consideration of these drawbacks, we adopt a hard EM training strategy to train the model with only the most possible trajectory.

A heuristic ranking is applied for selecting the best trajectory. The principle of finding the best trajectory is to find the most possible and correct one for each NL query. To be specific, we categorize the parsing results of all trajectories into "correct", "partially correct" and "wrong". Then the trajectory with the best correctness of category and highest probability is selected as the most possible trajectory. In other words, if there are correctly parsed trajectories, we will select the most possible one from it; otherwise, we continue to select the most possible one from partially correct trajectories.

Having the searched most possible trajectory $\hat{\mathbf{a}}$, the model is optimized with a standard negative log likelihood objective with respect to $\hat{\mathbf{a}}$:

$$J(\theta|\mathbf{w},\hat{\mathbf{a}}) = -\log P(\hat{\mathbf{a}}|\mathbf{w};\theta) = -\max_{i=1}^{T} \log P(\hat{a}_i|\mathbf{w};\theta). \tag{8}$$

Pretraining. To solve the cold starting problem and boost the training process, both the base parser and the shift-reduce splitter need to be pre-trained. The base parser is pre-trained with a set of NL queries without splitting (e.g., simple queries from the training set), which ensures a basic ability of the parser to convert simple NL queries into correct LFs. Then, the pre-trained base parser is applied to find split points of each query. The principle of searching split points is that the predicted sub LFs of each split out NL span can be composed into the ground truth LF. Given each query's split points, a trajectory can be produced, which is used as pseudo-data for the shift-reduce splitter pre-training. Note that an NL query can have no split point, which means the trajectory is filled with a list of SHIFT operation and a FIN operation at last.

4 Experiments

4.1 Datasets

We conducted the experiments on two datasets, i.e. the Geoquery dataset and the Complex Web Questions (ComplexWQ) datasets. They have a large diversity in topic, query length, and type of LF representation, which challenges the flexibility and generality of our proposed LASP.

Geoquery. This dataset collects 880 NL queries about U.S. geography with the corresponding Functional Query Language (FunQL) meaning representations [Zettlemoyer and Collins, 2012]. 600 of them are split out as training set with the rest 280 examples as test set.

ComplexWQ. This dataset contains NL questions and their SPARQL logical forms on Freebase [Talmor and Berant, 2018]. All the questions in this dataset are much longer and more complex than the queries in the Geoquery dataset, and require multi-hop reasoning to solve. ComplexWQ is preprocessed in the same way as [Zhang *et al.*, 2019] to anonymize the entities in the SPARQL LF representations.

4.2 Implementation Details

The basic encoder-decoder model with attention is used for the base parser, each of whom has a single bi-directional GRU hidden layer with the hidden dimension being 512. For the shift-reduce splitter, we search the word embedding dimension in the range [50, 100, **300**, 512], hidden size in the range [128, 256, **512**] and select the best hyperparameters as marked in bold for experiments. For the ComplexWQ dataset, we leverage Stanford NLP to give POS tags to each token. During model processing, the POS tags are converted into tag embedding vectors with dimensions being 30, which is concatenated by the word embedding vector of each token as additional information. We do not adopt a more complicated base parser, since we focus on the structural compositionality of NLs and LFs in this paper. One of the advantages of our proposed LASP is to free the base parser from handling very complex NL queries. A complex query is divided into several shorter and easier parts, so that a simple seq2seq parser can handle most cases.

Geoquery and ComplexWQ adopt different pre-training strategies and sub LF representations based on their characteristics. On pre-training strategy, Geoquery dataset is relevant small with various NL query lengths, so we use the original training data to pre-train the base parser and use the pre-trained base parser to detect possible spans from complete NL queries. Those found spans are converted into trajectories and used for shift-reduce splitter pre-training. Unlike Geuquery, ComplexWQ is a synthetic dataset, where each NL query is composed of two simple queries. We randomly select 20% queries from the training set and split each query by decomposing it into the original two sub queries by heuristic rules and dump pseudo data for pre-training. Note that this heuristic splitting cannot ensure a high accuracy, which should be further fixed during model training. On sub LF representation, Geoquery uses the nested FunQL representation, where the return value of a sub LF becomes a sub part of the higher sub LF, and thus we use a "<placeholder>" to take the place of the inside sub LF in the outside LF. For ComplexWQ with SPARQL, simply connecting all predicted sub LFs can form the complete one.

4.3 Baseline Models

The simplest yet meaningful baseline of our model is a pure Seq2Seq model [Dong and Lapata, 2016], as we just add a shift-reduce splitting mechanism to it. Some common semantic parsing methods are also chosen as baseline models, including Seq2Tree [Dong and Lapata, 2016], Scanner [Cheng *et al.*, 2017], Coarse2Fine [Dong and Lapata, 2018] and UNIMER [Guo *et al.*, 2020]. Besides, for the Geoquery dataset, we add GNN model [Shaw *et al.*, 2019] as a baseline model, which uses GNN to model the correlation between tokens and is one of the state-of-the-art parsers. For the ComplexWQ dataset, we also compare our method with HSP [Zhang *et al.*, 2019], which decomposes NL queries but predicts LFs as a whole with a complex three-stage process.

Method	**Accuracy**
Compositional Semantic Parser	
DCS [Liang *et al.*, 2013]	87.9%
TIPS [Zhao and Huang, 2015]	88.9%
Neural Semantic Parser	
Seq2Seq [Dong and Lapata, 2016]	84.6%
Seq2Tree [Dong and Lapata, 2016]	87.1%
Scanner [Cheng *et al.*, 2017]	86.7%
Coarse2Fine [Dong and Lapata, 2018]	88.2%
GNN [Shaw *et al.*, 2019]	89.3%
GNN+BERT [Shaw *et al.*, 2019]	92.5%
UNIMER [Guo *et al.*, 2020]	86.2%
Latent Shift-Reduce Semantic Parser (Ours)	
LASP	90.3%
- splitter	85.5%

Table 1: Parsing accuracies on Geoquery dataset.

Method	**Accuracy**
Seq2Seq [Dong and Lapata, 2016]	47.3%
Seq2Tree [Dong and Lapata, 2016]	49.7%
Transformers [Vaswani *et al.*, 2017]	53.4%
Coarse2Fine [Dong and Lapata, 2018]	58.1%
HSP [Zhang *et al.*, 2019]	66.2%
LASP	68.3%

Table 2: Parsing accuracies on ComplexWQ dataset.

4.4 Results

The parsing accuracies on Geoquery is shown in Table 1. Compared with previous compositional semantic parsers which require complex lexicon and syntax designing, our proposed LASP achieves higher performance even with a simple Seq2Seq semantic parser without BERT. In comparison with neural semantic parsers, LASP shows the highest accuracy among all methods except the GNN+BERT, which takes advantage of big pre-trained BERT. Similar observations appear on the ComplexWQ dataset, as shown in Table 2. Our model outperforms the state-of-the-art method HSP as well as other baseline models.

These results verify that our proposed LASP can efficiently split an NL query into spans according to its hierarchical structure, which matches with the hierarchical structure of the corresponding LF representation. Besides, by splitting a complex NL query into several spans, the requirement of parser's capability is reduced so that a simple base parser can achieve state-of-the-art accuracies.

4.5 Closer Analysis

Recent studies pointed out that the compositionality of logical form matters more than that of NL queries, as generating a logical expression already seen in the training set does not exhibit the composition capabilities of the model [Herzig and Berant, 2020]. Bearing this principle in mind, we conduct another set of experiments to test the composition ability on logical forms of our proposed method. For the Geoquery dataset, We resplit it according to logical forms and set all NL queries related to the same logical form skeleton into either training set or test set. As shown in Table 3, by resplitting the

Method	Geoquery(resplit)	ComplexWQ
LASP	77.7%	68.3%
- trajectory train	68.2%	52.9%
- splitter	62.4%	47.3%
GECA [Andreas, 2020]	60.3%	N/A

Table 3: Ablation study of LASP.

1	What state border the state that borders the most states	Correct
2	What three college do Robert F. Kennedy attend, and the organization found date is before 1885-03-16	Correct
3	What language is spoken in the country that have the national anthem “ afghan national anthem ”	Correct
4	What is the holy book of the religion whose sacred site be kushinagar	Wrong
5	What is the profession of the person who write “ the world I see ”	Wrong

Table 4: Case study on NL splitting.

dataset, the pure Seq2Seq model without splitting drops to an accuracy of 62.4%, while our LASP remains an accuracy of 77.7%, which is much higher.

On ablation studies, we conduct experiments to test the effectiveness of NL splitting and trajectory-based training method on the Geoquery(resplit) and ComplexWQ dataset. LASP with only heuristic supervised data training achieves an accuracy of 68.2% on Geoquery(resplit), which is much lower than the one with trajectory search and training. A similar observation can be drawn from the results of the ComplexWQ dataset. These results demonstrate the effectiveness of span-level NL-LF supervision. Data augmentation could be another way to enhance the parser performance, hence we also compared our LASP with a data-augmentation-based method GECA [Andreas, 2020] on the Geoquery (resplit), and LASP also shows better performance than GECA.

The trajectory-search process, which generates span-level supervised NL-LF pairs, can split a long NL query into several short spans with the corresponding sub LFs. Some of these NL spans may not exist in the original training set along, but are searched out by cutting long NL queries. By adding those unseen NL spans and the corresponding LFs into the training set, the parser is more reliable in generating correct LF predictions. Besides, the trajectory search process can fix the wrong splits caused by heuristic rules. A cyclic training between the base parser and the shift-reduce splitter helps the two modules adapt to each other and work better together.

4.6 Case Study

Table 4 shows some cases. Case 1 comes from the Geoquery dataset. It can be observed that our LASP supports splitting an NL query into multiple spans dynamically and each span is correctly parsed into a sub logical form representation. An advantage of LASP is that it can support dynamic and arbitrary splitting of NL queries. LASP can decide whether an NL query needs to be decomposed, and into how many parts. The flexibility of our method in splitting NL queries ensures applicability in different scenarios and LF types.

Case 2-5 are from the ComplexWQ dataset. Case 2 and Case 3 are correctly split while the last two are wrongly split. Error splitting is one of the main reasons of error cases in the ComplexWQ dataset. We randomly picked out 30 NL queries with their splits and manually analyzed them. We found that 7 out of 30 are wrongly split, and among the correctly split queries there are 5 error parsed ones. Surprisingly, one error split NL query got a correct parsing result, which is case 4 in Table 4. Though this NL query is wrongly split, key tokens for predicting sub LFs are kept in each span, thus a correct LF is predicted.

5 Related Work

Semantic Parsing has been a focused topic for a long time. Neural semantic parsing uses an encoder-decoder framework as the basic pipeline, which frees semantic parsing from heavy expert work, but these models lack the ability to solve complex NL queries [Lake, 2019; Keysers *et al.*, 2020]. Compositional semantic parsing breaks an NL utterance into several sub spans, which are much easier to parse. However, the splitting operation requires for predefined lexicon and grammar [Zelle and Mooney, 1996; Zettlemoyer and Collins, 2012; Berant *et al.*, 2013; Pasupat and Liang, 2015], or complex model designing and supervised training [Zhang *et al.*, 2019; Herzig and Berant, 2020]. Our work strives to blend the strengths of compositional semantic parsing and neural semantic parsing.

On NL decomposition, dependency parsing and syntactic parsing explore possible solutions for decomposing an NL utterance into spans, but these spans are not suitable for semantic parsing [Nivre, 2003; McDonald *et al.*, 2005]. The key challenge to task-specific NL decomposition is the lack of supervised data [Shen *et al.*, 2019]. A common solution is to generate pseudo-data by heuristic rules or by data augmentation [Andreas, 2020]. However, the quality of rules impacts the overall performance greatly, and these rules are not extendable. Our proposed LASP is a latent model that finds the pseudo splitting supervision by the model itself.

6 Conclusion

In this paper, we propose LASP, a latent shift-reduce parser for semantic parsing. LASP splits an NL query according to its hierarchical structure, which also aligns with the hierarchical structure of LF. LASP does not require span-level supervision for NL query splitting but uses supervised pseudo data generated during training by a trajectory search process. Experimental results show that LASP consistently improves the performance of different datasets with different LF types.

Acknowledgments

We thank all the anonymous reviewers for their valuable comments. We also thank Zeqi Lin for helping us to revise this paper. This work was supported in part by National Key Research and Development Program of China under Grant No. 2018AAA0101902, and NSFC under Grant No. 61532001.

References

[Andreas, 2020] Jacob Andreas. Good-enough compositional data augmentation. In *ACL*, 2020.

[Berant *et al.*, 2013] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, pages 1533–1544, 2013.

[Cheng *et al.*, 2017] Jianpeng Cheng, Siva Reddy, Vijay Saraswat, and Mirella Lapata. Learning structured natural language representations for semantic parsing. In *ACL*, 2017.

[Dong and Lapata, 2016] Li Dong and Mirella Lapata. Language to logical form with neural attention. In *ACL*, 2016.

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

[Dong *et al.*, 2018] Li Dong, Chris Quirk, and Mirella Lapata. Confidence modeling for neural semantic parsing. In *ACL*, 2018.

[Guo *et al.*, 2020] Jiaqi Guo, Qian Liu, Jian-Guang Lou, Zhenwen Li, Xueqing Liu, Tao Xie, and Ting Liu. Benchmarking meaning representations in neural semantic parsing. In *EMNLP*, 2020.

[Herzig and Berant, 2020] Jonathan Herzig and Jonathan Berant. Span-based semantic parsing for compositional generalization. *arXiv preprint arXiv:2009.06040*, 2020.

[Jia and Liang, 2016] Robin Jia and Percy Liang. Data recombination for neural semantic parsing. In *ACL*, 2016.

[Keysers *et al.*, 2020] Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, et al. Measuring compositional generalization: A comprehensive method on realistic data. In *ICLR*, 2020.

[Lake, 2019] Brenden M Lake. Compositional generalization through meta sequence-to-sequence learning. In *NeurIPS*, pages 9791–9801, 2019.

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

[McDonald *et al.*, 2005] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. Non-projective dependency parsing using spanning tree algorithms. In *EMNLP-HLT*, pages 523–530, 2005.

[Nivre, 2003] Joakim Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the Eighth International Conference on Parsing Technologies*, pages 149–160, 2003.

[Pasupat and Liang, 2015] Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. In *ACL-IJCNLP*, 2015.

[Shaw *et al.*, 2019] Peter Shaw, Philip Massey, Angelica Chen, Francesco Piccinno, and Yasemin Altun. Generating logical forms from graph representations of text and entities. In *ACL*, 2019.

[Shen *et al.*, 2019] Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. Ordered neurons: Integrating tree structures into recurrent neural networks. In *ICLR*, 2019.

[Talmor and Berant, 2018] Alon Talmor and Jonathan Berant. The web as a knowledge-base for answering complex questions. In *NAACL*, 2018.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

[Zelle and Mooney, 1996] John M Zelle and Raymond J Mooney. Learning to parse database queries using inductive logic programming. In *AAAI*, pages 1050–1055, 1996.

[Zettlemoyer and Collins, 2012] Luke S Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *arXiv preprint arXiv:1207.1420*, 2012.

[Zhang *et al.*, 2019] Haoyu Zhang, Jingjing Cai, Jianjun Xu, and Ji Wang. Complex question decomposition for semantic parsing. In *ACL*, pages 4477–4486, 2019.

[Zhao and Huang, 2015] Kai Zhao and Liang Huang. Type-driven incremental semantic parsing with polymorphism. In *NAACL*, 2015.