On the Effects of Structural Modeling for Neural Semantic Parsing

Xiang Zhang[a] and **Shizhu He**[b] and **Kang Liu**[b] and **Jun Zhao**[b]
[a]School of Artificial Intelligence, University of Chinese Academy of Sciences
[b]The Laboratory of Cognition and Decision Intelligence for Complex Systems,
Institute of Automation, CAS
{xiang.zhang,shizhu.he,kliu,jzhao}@nlpr.ia.ac.cn

Abstract

Semantic parsing aims to map natural language sentences to predefined formal languages, such as logic forms and programming languages, as the semantic annotation. From the theoretic views of linguistic and programming language, structures play an important role in both languages, which had motivated semantic parsers since the task was proposed in the beginning. But in the neural era, semantic parsers treating both natural and formal language as sequences, such as Seq2Seq and LLMs, have got more attentions. On the other side, lots of neural progress have been made for grammar induction, which only focuses on natural languages. Although closely related in the sense of structural modeling, these techniques hadn't been jointly analyzed on the semantic parsing testbeds. To gain the better understanding on structures for semantic parsing, we design a taxonomy of structural modeling methods, and evaluate some representative techniques on semantic parsing, including both compositional and i.i.d. generalizations. In addition to the previous opinion that structures will help in general, we find that (1) structures must be designed for the specific dataset and generalization level, and (2) what really matters is not the structure choice of either source or target side, but the choice combination of both sides. Based on the finding, we further propose a metric that can evaluate the structure choice, which we believe can boost the automation of grammar designs for specific datasets and domains.

1 Introduction

Semantic parsing is the task to transduce source sentences in natural languages (NL), into the target representations, which are usually artificial formal languages (FL), such as Lisp, λ-calculus, and SQL. Theoretically natural languages are processed in structures (Chomsky, 2009), and the formal languages are also defined to have a context-free syntax (Linz and Rodger, 2022). Therefore inevitably semantic parsers such as the CCG-based are aware of source structures, and adopt the compositional semantics [1] of the targets. But they usually parse to λ-calculus (Venant and Koller, 2019) and do not support programming languages.

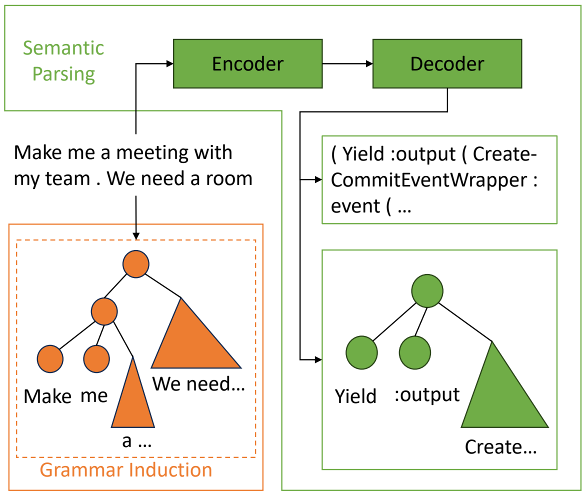
Semantic Parsing
Encoder
Decoder
Make me a meeting with my team . We need a room
(Yield :output (Create-CommitEventWrapper : event (...
Make me
a ...
We need...
Grammar Induction
Yield
:output
Create...

Figure 1: Structural modeling in two tasks. We're going to analyze how the progress in grammar induction could help neural semantic parsing.

In the neural era, Seq2Seq based parsers add supports to any sequential languages, but they can make grammar errors despite the effectiveness. Grammar-based parsers are proposed to ensure the grammatical correctness by decoding the rule sequences of the target AST. Recently, the development of the Text-to-SQL has motivated specialized parsers to support the SQL language. But the NL structures on the source side are seldom handled and left to pretrained large models.

On the contrary, NL structures are the key issues of treebanks like PTB and supervised parsers. The grammar induction field has also invented many methods to induce grammars with restricted forms from unsupervised training data. These parsers can

[1] Typical compositions are β-reductions in the λ-calculus and the unification in the functional grammar.

infer trees for new sentences, but don't process the semantic annotations obviously.

Unfortunately, no investigations had been conducted on the combination of the success of the two fields. Our research question (**RQ**) is thus as follows: **Is structural modeling of the natural language or the formal language useful for neural semantic parsing?** To answer the question, we use the encoder-decoder architecture with the attention mechanism to connect structures of two sides, due to its success of modeling token-level correlations. Our investigations are kept diverse in several important factors, such as the dataset variety, categories of structures, and generalization levels (I.I.D., compositional, or zero-shot). Under every possible combination of these factors, results are believed more faithful than single datasets (Finegan-Dollak et al., 2018).

Our evaluations add new knowledge to prior insights (Oren et al., 2020). We find it's not safe to claim the effectiveness for specific structural models for either NL or FL. The structures of NL and FL must be evaluated as a whole, and their effects even vary across datasets and generalization levels. Therefore, we make the conclusion that the combination of structural choices are more important than the structural choice on either the source or target side. The result is consistent with the one of the findings from Guo et al. (2020) in that different grammars, leading to different tree structures, have significantly different performance when keeping the same semantic representations and datasets.

These arguments in total suggest we can expect improvements from searching for better structural combinations on specific application domains. However, grammar search is not trivial but can be highly expensive. Inspired by the recent works in Large Language Models (LLMs) which can handle the code inputs well, we propose the metric, DisStruct, for evaluating the structural combination of the source and target sides based on the representations given by the LLMs and the optimal transport. The metric can be interpreted as the discrepancy between the specific training and testing splits under the structural choices. The metric is shown negatively correlated with the parser performance. It thus will help the automation of the grammar search theoretically.

In summary, we make three contributions as:

- We're the first to classify and compare representative structural models for neural semantic parsing, to our best knowledge.
- By evaluating the models against a few diverse testbeds, we find that structural combinations are more important than structural choice of either the natural or formal languages.
- We propose a metric of the structural combinations that is negatively correlated with the model performance which can speed up the structure searching.

2 Evaluation Framework

2.1 Datasets

As suggested by Finegan-Dollak et al. (2018), we conduct the experiments on a variety of datasets, which are different in sizes, anonymized query amounts, nested query depths, and involved SQL table amounts. We use the ATIS, GEO, Scholar, Advising (Oren et al., 2020), COGS (Kim and Linzen, 2020), and SMCalFlow-CS (Yin et al., 2021). The selection also covers several semantic representations. Table 1 gives the statistics. For the generalization levels, three have been proposed for the Question Answering task, i.e., the I.I.D., compositional, and zero-shot generalization(Gu et al., 2021). For semantic parsing, usually only the first two levels are considered. The I.I.D. generalization is just a uniformly random shuffle and

Dataset	**Split**	**# Examples** (train / dev / test)
ATIS (SQL)	I.I.D.	3014 / 405 / 402
ATIS (SQL)	Program	3061 / 375 / 373
Advising (SQL)	I.I.D.	3440 / 451 / 446
Advising (SQL)	Program	3492 / 421 / 414
Geo (SQL)	I.I.D.	409 / 103 / 95
Geo (SQL)	Program	424 / 91 / 91
Scholar (SQL)	I.I.D.	433 / 111 / 105
Scholar (SQL)	Program	454 / 97 / 98
COGS (λ-calculus)	I.I.D.	24160 / 3000 / 3000
COGS (λ-calculus)	Linguistic	24160 / 3000 / 21000
SMC16 (Lispress)	Domain	25424 / 1324 / 1325
SMC32 (Lispress)	Domain	25440 / 1324 / 1325
SMC64 (Lispress)	Domain	25472 / 1324 / 1325
SMC128 (Lispress)	I.I.D.	25536 / 1324 / 1325
SMC128 (Lispress)	Domain	25536 / 1324 / 1325

Table 1: The number of examples in each dataset. Different kinds of generalizations are explained in Section 2.1. SMCk denotes the SMCalFlow-CS dataset with k few-shot examples added into the training set. We manually shuffle the SMC-128 to build an I.I.D. split. The representation of each dataset is in the parenthesis.

split of the entire corpus. For the compositional generalization (CG), there isn't a standard split procedure currently. In our work, ATIS, GEO, Scholar, and Advising adopt the program-based split, which anonymize SQL queries as program templates and split the data at the template level. The COGS constructs CG examples in a linguistic view. The SMCalFlow-CS adopts the domain-based split, which uses single-domain questions for training, and questions requiring multi-domain knowledge for testing.[2]

2.2 Problem Formalization

We are focusing on encoder-decoder models to map a source sentence X into the target formal language Y. Basic forms of X, Y are provided as linear sequences, i.e. $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_m)$, where each x_i and y_j are tokens. Trees of source and target sides are denoted as S, T with respectively X and Y as their leaf nodes. For both S, T, three structural choices are available: **absent**, **latent**, and **given**. An absent structure is a pure sequence. Latent structure means the tree is not observed and jointly learned from the training data. Given structures rely on external parsers. The combination of choices of S, T yields a total of 9 probabilistic models as in Table 2.

Model Form	S Choices	T Choices
$P(Y \mid X)$	Absent	Absent
$P(Y, T \mid X)$	Absent	Latent
$P(T \mid X)$	Absent	Given
$P(S \mid X)P(Y \mid S, X)$	Latent	Absent
$P(S \mid X)P(Y, T \mid S, X)$	Latent	Latent
$P(S \mid X)P(T \mid S, X)$	Latent	Given
$P(Y \mid S, X)$	Given	Absent
$P(Y, T \mid S, X)$	Given	Latent
$P(T \mid S, X)$	Given	Given

Table 2: Probabilistic forms for all Seq2Seq-style models in comparison. Structures of both side can be one of three choices. If S is latent, training another model $P(S \mid X)$ is necessary to infer S.

Note we only consider the deterministic parsers instead of the generative ones. The models must predict at least one variable of the target side, given at least one variable of the source side. We've noticed several works using generative grammars (Qiu et al., 2021; Kim, 2021; Shaw et al., 2021) based on the notions of synchronized and quasi-synchronized CFGs. Due to the prevalence of deterministic semantic parsers, we leave generative models in the future work.

[2]Others like length-based and divergence-based splits (Shaw et al., 2021; Keysers et al., 2020) are not included for comprehensiveness due to limited computation resources.

2.3 Selected Structural Models

We briefly list the concrete models for structural choices in Table 3. The implementations and hyperparameters are left in the Appendix. Referring the original papers is also recommended for details.

S	Model
Absent	Bidirectional LSTM BERT (Devlin et al., 2019) Electra (Clark et al., 2020)
Latent	ON-LSTM (Shen et al., 2019) DIORA (Drozdov et al., 2019) PCFGs (Kim et al., 2019a; Yang et al., 2021) Perturb & Parse (PnP) (Corro and Titov, 2019b)
Given	Berkeley Parser + GCN
T	**Model**
Absent	LSTM
Latent	ON-LSTM (Shen et al., 2019)
Given	Handcrafted EBNF Grammars + LSTM

Table 3: Models for corresponding S and T choices.

Among the S choices, PnP gives a latent dependency tree, while others including the Berkeley Parser (Kitaev et al., 2019; Kitaev and Klein, 2018) produce constituency trees. For the T choices, all methods are focusing on constituency trees because formal languages have been defined with CFGs.

Note if T is given, we manually construct the grammar for COGS and SMCalFlow-CS, and use the grammar induced by Oren et al. (2020) for other datasets[3]. We use a parser generator to load grammars and follow the grammar-based parsing (Krishnamurthy et al., 2017; Yin and Neubig, 2018) to use LSTM to model the production rule sequence.

2.4 Evaluation Method

We use the Exact Match (EM) to measure accuracies. For absent and latent T choices, the generation target must be the same tokens as Y. When the oracle T is given, the model must similarly generate the same rule sequences of that T.

We have to report the aggregated results because the experiment number is proportional to #datasets $\times$ #generalization-levels $\times$ #S-models

[3]The CFG grammar of dataset are in the Appendix E to G.

× #T-models × #random-seeds[4]. The merit of results aggregation is its robustness. For example, once we find the ON-LSTM as the decoder useful, it is expected to generalize and work well under a variety of settings. Winning or losing on one setting is not critical.

For analysis, we assign each experiment result with factor labels, and the results will be aggregated under the perspective of factors. The factors we considered are representation types, S-choices, T-choices, and syntactic tree types. For example, when focusing on T-choices, we can compare accuracies of the 3 labels on a specific dataset and split. Each number is mean-aggregated over all S models, like the "GROUP BY" in SQL. The aggregation view will help us focus on what we're interested in and not get lost in enormous results.

3 Results Analysis

3.1 Lateral Structural Modeling

We first focus on aggregations for single factors on compositional generalization (CG). Each factor label corresponds to aggregated accuracies on 9 datasets, which are plotted as a single box.

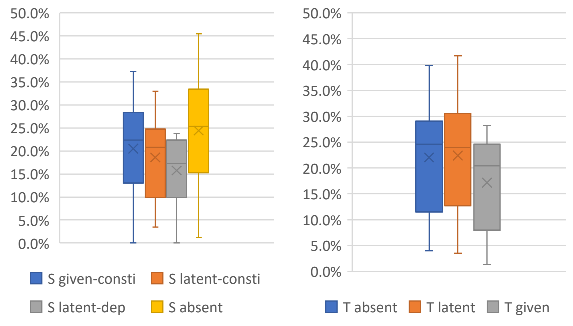

Figure 2: Accuracies viewed in S and T choices. Each bar is a distribution across all 9 CG datasets.

Figure 2 shows the absent S structure outperforms others, followed with given S then the latent. The constituency trees are also better than dependency trees. On the target side, the latent T is on a par with absent T, beating the given T by a large margin. Results on both sides suggest no structural bias is the best choice. Furthermore, when we zoom in the aggregation as in Figure 3, it's clearly the low performance of the latent S is caused by many poor latent models. Incredibly, among the

[4]Following Oren et al. (2020), we run experiments on SQL datasets with 5 random seeds because they're small. Raw accuracies without aggregation are listed in Appendix D.

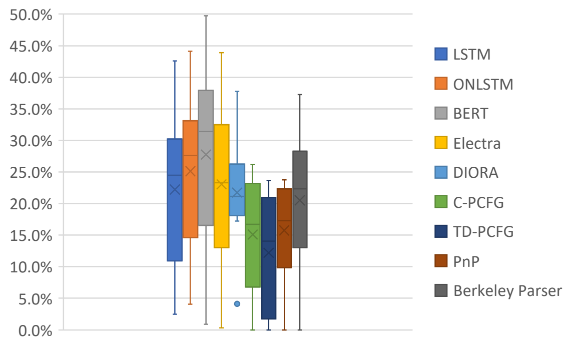

Figure 3: Accuracies viewed in S models. Each bar is the distribution of accuracies on 9 CG datasets.

latent S, the ON-LSTM works even as well as the Electra, and only falls behind BERT perhaps due to the parameter scales.

Takeaway Structural modeling CAN be useful. But finding a good discrete structure is not trivial. While handcrafted grammars of formal languages can be harmful, supervised parsers for natural languages are not that bad. Overall, a latent structural bias like ON-LSTM is the most promising.

3.2 Combinations of Source and Target

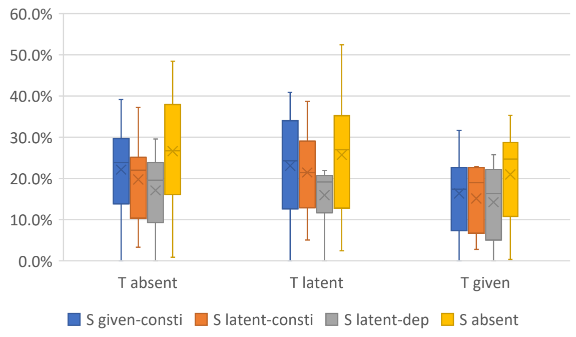

Figure 4: Accuracies viewed in combinations of each S and T choice, on 9 CG datasets.

We further analyze results of each S and T choice combination in Figure 4. The accuracy relations are similar to the S and T choices in Figure 2, with a few exceptions. First, when T structure is given, there's not much difference between the given and latent S choices. Therefore, the handcrafted grammars (the given T) are proven poor such that no trivial structural bias for the NL can be found to cooperate with it. Only with absent S structures can the performance be improved at this time. Second, when S is the latent dependency tree, the latent T is the worst, contrary to the right boxplot in Figure 2. This suggests that a latent dependency tree for S and a latent constituency tree for T are not compatible.

Takeaway Some incompatible combinations of the source and target choices of structural biases can lead to a performance below the average of any choice on its own.

3.3 Latent Source Structures

Section 3.1 shows that there're big discrepancies among the latent S models. We first compare the PCFGs in Figure 5. The Compound PCFG (Kim et al., 2019a) and TD-PCFG (Yang et al., 2021) are chosen as two basic PCFG variants. In addition, we build a reduced version for each of them by summing out the non-terminals at each cell in the parsing chart with a learnt prior, such that the cell will only store the representation of a span, instead of the representations of a span of every possible non-terminal. This trick can reduce the chart size from $O(n^2K)$ to $O(n^2)$, where K is the number of nonterminals. Appendix A lists more details.

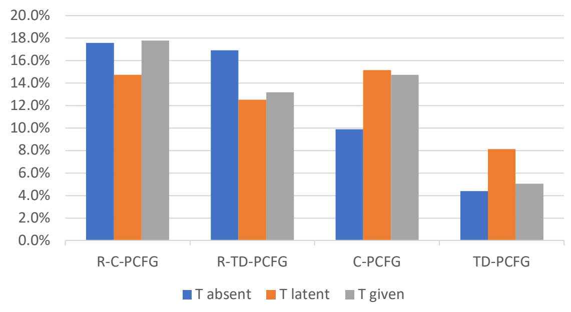

Figure 5: Accuracies for different PCFGs as encoders against different T choices on the GEO datasets with compositional generalization.

In general, the full rank C-PCFG performs better than its counterpart TD-PCFG with decomposed and less parameters. The reduced PCFGs can also outperform the basic ones. With latent and given T choices the C-PCFG works also well, but is not as good as the reduced version. This suggests a less constrained structural bias like the reduced PCFGs not storing non-terminals in the chart can be much better than the fully-fledged PCFGs. We therefore only evaluates the reduced PCFGs on other datasets because they have higher accuracies and less memory consumption.

Figure 6 shows only the performance of latent S models against different T choices. The ON-LSTM clearly beats other encoders, followed by

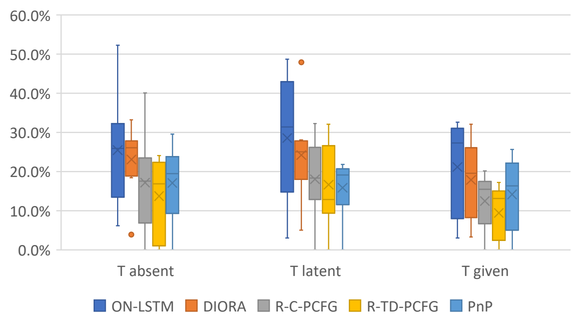

Figure 6: Accuracies for latent S models with different target T choices. Each bar is the distribution of accuracies on 9 CG datasets.

the DIORA encoder. Altogether with the Figure 5, we can make some interesting conclusions. First, by summing out non-terminals, reduced PCFGs have outperformed the basic models. Then, the DIORA discards non-terminals in its parameterization, and only considers compositions over spans with a chart-based parsing and an inside-outside algorithm. And it has beaten the PCFGs, Finally, the ON-LSTM which does not forcing syntactic trees being of Chomsky Normal Form, has achieved the best performance.

Takeaway Latent structural biases with less constraints would be better choices. Enforcing syntactic categories may not be suitable for neural semantic parsing.

3.4 Differences between Accuracies

The above findings tell us we have to find the compatible structural biases in general. In this section we compare the structural choices among different datasets. We focusing on the T choices and do not aggregate results of datasets and S choices. Specifically, we subtract the number of absent and latent T accuracies with the number of given T accuracies. As long as the differences are positive, the absent and latent T will be considered outperforming the given T that is constructed from handcrafted grammars. For the latent T, we only consider the best 3 models from previous analysis, i.e., the ON-LSTM, DIORA, and PnP. We consider both the I.I.D. and compositional generalizations, as shown in Figure 7.

The most intuitive result in Figure 7 is that among various datasets the given T is not consistently bad. On the SMCalFlow, the given T is outperformed by the absent and latent T, but the margins are not that large on other datasets in the

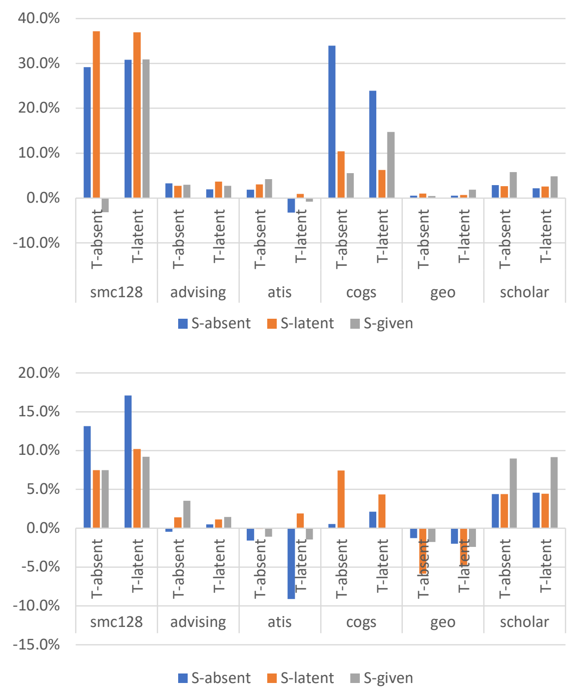

Figure 7: Differences subtracted the given T accuracies from the latent and absent T, under each dataset and each S-choice, with I.I.D. (Top) and compositional generalization (Down). Positive values mean that the latent or absent T outperforms the given T, while negative values suggest the given T is better.

I.I.D. setting. For the compositional generalization (the lower subfigure), we can even see the given T has not been outperformed on ATIS and GEO, but is poor on Advising and Scholar. Moreover, on the same dataset like ATIS and GEO, the handcrafted grammar is harmful on I.I.D. but useful on C.G. Also, the results on T choices are slightly different under different S choices, which again supports the compatibility argument in previous sections.

Takeaway Grammars of the formal languages can't be simply classified as useful or not. There must be an optimal grammar, depending on the datasets and generalization levels.

3.5 Discussions

After analyzing the structural modeling methods in different views, we're trying to answer our basic research question (RQ) based on the findings to make the answers and even the question itself much clearer.

RQ: Is structural modeling of the natural language or the formal language useful for neural semantic parsing? Yes AND no. It depends on the models. In general we find that models without structures (BERT) and with latent structures (ON-LSTM) are better for the natural language, but other structures are not useful. Specifically, the ON-LSTM is even better than the finetuned Electra as the encoder. For the formal languages, we find the latent structural model (ON-LSTM) is much better, but the handcrafted grammar-based decoding is poor (Section 3.1).

Why are the structural models that different? We hypothesize that the differences are rooted in the strictness of structural constraints of the models. For constituency trees, we find the more structural restrictions required by the model, the worse performance it would be (Section 3.3). Among these models, ON-LSTM neither differentiates syntactic categories, nor requires the Chomsky Norm Form tree, and has outperformed other models.

Since the ON-LSTM is proven effective, can we use it all the time? No. We're not recommending ON-LSTM for all situations. Because the compatibility of structural choices is more important. If the encoder is a structural model based on dependency trees, the ON-LSTM decoder will not perform well neither. What is really crucial is the encoder-decoder choices combined as a whole (Section 3.2).

Shall we use the best combination, the ON-LSTM for both the encoder and decoder? Not always. We further find the same structural combination could be not the same effective on all datasets and all generalization levels (Section 3.4). On the GEO with the compositional generalization, ON-LSTM performs worse than handcrafted grammars. In fact, the absent T can be seen a special structure, the right-branching tree with autoregressive decoders like RNNs. For example, an SQL query sequence is equivalent to the tree like (SELECT (* (FROM (tableA (WHERE (...))))). Therefore, the question is in fact asking what kind of trees are better, for the natural and formal languages, combined as a whole, under a specific dataset and a generalization level. We're going to handle this in Section 4. But, if the datasets and generalizations are not our concerns, the BERT or ON-LSTM as the encoder with the ON-LSTM decoder is recommended according to the above findings.

4 Metric for Structural Evaluation

Taking sequences as the right-branching trees, the models we've discussed can all parse an example (x, y) to its structures $(s(x), t(y))$. But the generalization performance is not only determined by some smart structural choices. It also depends on the dataset and the generalization level. However, it's expensive to manually design good structures, or to optimize a parameterized structural policy. Because on one hand we have to train and then evaluate a parser every time we need to confirm the effectiveness of that policy. On the other hand, even a parser jointly learning mappings and latent structures may work poorly according to above findings.

Inspired by the recent success on large language models (LLMs) (Sun et al., 2022) such as the Codex (Chen et al., 2021) which can read and write programming source codes well, we propose a learning-free metric for the structures based on the representations generated by LLMs, such that it's correlated with the performance.

Specifically, to evaluate a pair of structural models (s, t) for a dataset $D = (x, y)_i$, we first define the distance between a parallel sequence (x, y),

$$e_x, e_y = LLM(x), LLM(y) \tag{1}$$

$$e_s, e_t = f(s(x), e_x), f(t(y), e_y) \tag{2}$$

$$d_{s,t,D} = \mathbb{E}_{(x,y)\in D}[emd(u_s, u_t, cost(e_s, e_t))] \tag{3}$$

where $e_x \in \mathbb{R}^{n\times k}, e_y \in \mathbb{R}^{m\times k}$ are the k-dimensional representations generated by some LLM that can understand both natural and formal languages, $s(\cdot), t(\cdot)$ are the parsers or policies that output tree structures for x, y, and the f computes the representation of each tree node. We define the leaf nodes have the same representations in e_x, e_y, and internal nodes get their representations by mean-pooling of its children nodes. $u_s \in \mathbb{R}^l$ and $u_t \in \mathbb{R}^r$ are discrete uniform distributions, where l, r are node numbers of $s(x), t(y)$ respectively. The emd function returns the Earth Moving Distance (Peyré et al., 2019) of u_s, u_t under the cost matrix defined by euclidean distances of e_s, e_t. $d_{s,t,D}$ is the minimal transport cost from X to Y for the entire dataset D. We utilize the POT toolbox (Flamary et al., 2021) to compute the optimal transport. Then given the training and testing sets D_{train}, D_{test}, the DisStruct metric is defined as

$$M(s,t) = \frac{|\mathbb{E}[d_{s,t,D_{train}}] - \mathbb{E}[d_{s,t,D_{test}}]|}{\sigma[d_{s,t,D_{train}}]\sigma[d_{s,t,D_{test}}]} \tag{4}$$

where the expectation $\mathbb{E}$ and standard deviation σ are implemented by re-running with a few random seeds. In our evaluation, we sample 50 examples for the expectation in Eq.(3), and rerun 10 times for Eq.(4).

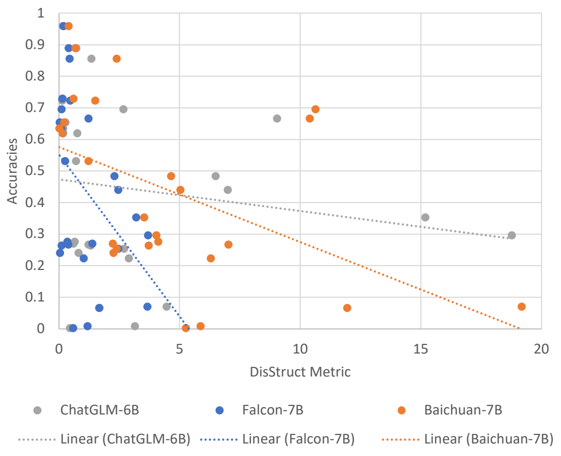

Figure 8: Fitting the metrics of different (S, T) choices to the accuracies on different datasets and generalizations. We include the absent S and both absent and given T, showing whether the metric can reflect the differences between the grammar-based and the sequence-based structures of the formal languages. Metrics computed with 3 chosen LLMs are all shown negatively correlated with the performance.

Intuitively, given structural choices (s, t), the DisStruct evaluates the distances of x and y of a single example, and computes the distance discrepancies between D_{train}, D_{test}. Therefore, we can expect higher performance by finding lower metric values from some (s, t) pair. Figure 8 illustrates the correlations. Although every (s, t) can yield a metric value, we plot only two kinds of pairs (absent, absent) and (absent, given) and investigate whether the metric can tell apart the differences between the grammar-based and the sequence-based structures. With three recent LLMs[5] that we can load with less than 24GB GPU, the metrics are shown all negatively correlated with the performance as expected.

Since each fitted linear model has a low R^2 value (i.e., plots far from the fitted line), we examine the results by datasets. As long as the metric can indicate performance for datasets, it'll be possible to probe or search structural choices for a specific dataset we're interested in. For each dataset under a generalization level, we only have 2 points. We computed the slope of the line determined by the

[5]ChatGLM-6B (Du et al., 2022): https://github.com/THUDM/ChatGLM-6B; Falcon-7B (Almazrouei et al., 2023): https://huggingface.co/tiiuae/falcon-7b; Baichuan-7B: https://huggingface.co/baichuan-inc/baichuan-7B.

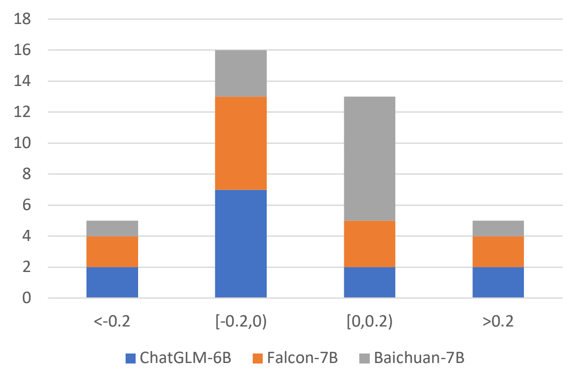

Figure 9: On each dataset and generalization level (totally 13 here), we compute metrics for two pairs, i.e. (absent, absent) and (absent, given), corresponding to two points in Figure 8. We plot the histogram for the slope of each line determined by the two points. The slopes are negative and are also low when positive, suggesting the metrics are possibly indicative for specific datasets and generalization level.

two points, and plot the histogram of the slopes in Figure 9. Hopefully, the slopes are negative at more than 50% times, and are also relatively small even it's positive. We also find the metrics based on ChatGLM-6B and Falcon-7B are more ideal than Baichuan-7B.

5 Related Works

Many representations have been used for semantic parsing. Popular representations include semantic roles, FOL or λ-calculus (Zettlemoyer and Collins, 2005, 2007; Wong and Mooney, 2007), λ-DCS (Liang et al., 2013), FunQL (Kate et al., 2005; Guo et al., 2020), application-specialized query graphs (Yih et al., 2015; Chen et al., 2018; Hu et al., 2018), and programming languages like SQL (Xu et al., 2018), Java (Iyer et al., 2018; Alon et al., 2020), and Python (Yin and Neubig, 2017; Rabinovich et al., 2017). Linguists also design meaning representations such as AMR (Banarescu et al., 2013), ERS (Flickinger et al., 2014), and UMR (Van Gysel et al., 2021). Abend and Rappoport (2017) had reviewed many semantic representations in a linguistic-centric perspective, and Li et al. (2022) had proposed a metric to evaluate different representations. Our discussions are not at representation level (only the lispress, λ-calculus, and SQL are used), but on structure effects under maybe a fixed representation.

Classic semantic parsers used to assign categories to linguistic or semantic fragments, and compose them in a bottom-up fashion. Some typical implementations are based on CCG (Zettlemoyer and Collins, 2005), SCFG (Wong and Mooney, 2006), Hyperedge Replacement Grammar (Chiang et al., 2013), and AM Algebra (Groschwitz et al., 2017, 2018; Weißenhorn et al., 2022). Other parsers do not define linguistic categories, but use feature engineering or types to guide composing algorithms (Liang et al., 2013; Pasupat and Liang, 2015).

Neural parsers like Seq2Seq (Xiao et al., 2016) adopt end-to-end mappings but can make grammar errors. Seq2Tree (Dong and Lapata, 2016) is then proposed to generate grammatically valid trees for untyped λ-calculus. Grammar-based decoding (Krishnamurthy et al., 2017; Yin and Neubig, 2018) turns to generate rule sequences converted from the target AST. Some parsers design intermediate patterns for an easier abstraction over the targets (Zhang et al., 2017; Dong and Lapata, 2018; Guo et al., 2019; Ding et al., 2019; Iyer et al., 2019; Choi et al., 2021; Chen et al., 2020). The abstraction layer can be seen as handcrafted structures for the targets. We only consider CFG-based structures due to their generality. Similarly, graph-based targets and parsers are also beyond our discussing. LLMs as semantic parsers (Qiu et al., 2022; Zhuo et al., 2023) are found not performing well on the COGS dataset before structural discussions. We leave some results and discussions in Appendix C.

Recently the compositional generalization has attracted much focus (Jambor and Bahdanau, 2022; Liu et al., 2021; Herzig and Berant, 2021). But they either devise special parsers other than the encoder-decoder architecture, or handle representations like FunQL, therefore not direct applicable to other general parsers. Zheng and Lapata (2022) reports the entanglement problem where Seq2Seq models entangle irrelevant semantic factors during generation. Yin et al. (2021) induces token and span level alignments. Our structural discussions are orthogonal to their model improving works.

6 Conclusion

By evaluations on a variety of settings, we find the structural modeling is not guaranteed to give better performance. We conclude that structural biases for sources and targets must be chosen as a whole, and that choices also depend on the specific dataset and generalization level. We propose the DisStruct metric to facilitate structure finding, which is negatively correlated with the performance.

Limitations

We've discussed a variety of structural models, but may lack the tuning of hyperparameters for each model to work at its best. For example, the number of nonterminals and preterminals are important for PCFGs, but we use a small number compared with the grammar induction task on PTB due to our small dataset size. Also, it is a reasonable guess that BERT and ELECTRA as encoders are inferior than large language models such as T5, Falcon, and ChatGPT. We have not conduct experiments on datasets simply because of limited computation resources. Also we note that LLMs can be used as the decoder-only models, and generate targets via in-context learning or zero-shot prompts. We left the results in the Appendix C because structural models or representations we concerning are not involved in the paradigm.

Furthermore, our study is all English-based datasets. Considering the large differences between language families, the structure model of constituency and dependency trees in our study may have different effects. Universal structures such as the Universal Dependencies (de Marneffe et al., 2021) may be considered for future research.

Finally, DIORA and PCFGs in our study require approximately 4 times more GPU memories than other encoders (excluding the BERT and ELECTRA of course). This may be caused by the CKY-style computation which is $O(n^3s^2)$ in time where n is the sentence length and s is the number of syntactic categories. This will leads to more GPU consumption to compute the tensor graph. We're also wondering if a sample-based learning algorithm could work instead of the inside algorithm.

Ethics Statement

Since our study is objective, we have reviewed our datasets. The contents of the datasets are publicly available for years and obtainable without checking the membership of any group. In addition, some datasets had adopted careful preprocessing such as anonymization which replaced real-world entity names with placeholders. The dependent code resources are managed in public repositories. And so will ours. So far we believe our work does not have ethical concerns.

Acknowledgements

This work was supported by National Key R&D Program of China (No.2022ZD0118501) and the National Natural Science Foundation of China (No.62376270, No.U1936207, No.61976211). This work was supported by the Strategic Priority Research Program of Chinese Academy of Sciences (No.XDA27020100), Youth Innovation Promotion Association CAS, Yunnan Provincial Major Science and Technology Special Plan Projects (No.202202AD080004).

References

Omri Abend and Ari Rappoport. 2017. The state of the art in semantic representation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 77–89, Vancouver, Canada. Association for Computational Linguistics.

Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Merouane Debbah, Etienne Goffinet, Daniel Heslow, Julien Launay, Quentin Malartic, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. 2023. Falcon-40B: an open large language model with state-of-the-art performance.

Uri Alon, Roy Sadaka, Omer Levy, and Eran Yahav. 2020. Structural language models of code. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 245–256. PMLR.

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

Bo Chen, Xianpei Han, Ben He, and Le Sun. 2020. Learning to map frequent phrases to sub-structures of meaning representation for neural semantic parsing. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(0505):7546–7553.

Bo Chen, Le Sun, and Xianpei Han. 2018. Sequence-to-action: End-to-end semantic graph generation for semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 766–777, Melbourne, Australia. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz

Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code.

David Chiang, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, Bevan Jones, and Kevin Knight. 2013. Parsing graphs with hyperedge replacement grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 924–932, Sofia, Bulgaria. Association for Computational Linguistics.

DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. 2021. RYANSQL: Recursively applying sketch-based slot fillings for complex text-to-SQL in cross-domain databases. *Computational Linguistics*, 47(2):309–332.

Noam Chomsky. 2009. Syntactic structures. In *Syntactic Structures*. De Gruyter Mouton.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pretraining text encoders as discriminators rather than generators. In *ICLR*.

Caio Corro and Ivan Titov. 2019a. Differentiable perturb-and-parse: Semi-supervised parsing with a structured variational autoencoder.

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

Marie-Catherine de Marneffe, Christopher D. Manning, Joakim Nivre, and Daniel Zeman. 2021. Universal Dependencies. *Computational Linguistics*, 47(2):255–308.

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.

Jiwei Ding, Wei Hu, Qixin Xu, and Yuzhong Qu. 2019. Leveraging frequent query substructures to generate formal queries for complex 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 2614–2622, Hong Kong, China. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742, Melbourne, Australia. Association for Computational Linguistics.

Andrew Drozdov, Patrick Verga, Mohit Yadav, Mohit Iyyer, and Andrew McCallum. 2019. Unsupervised latent tree induction with deep inside-outside recursive auto-encoders. 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 1129–1141, Minneapolis, Minnesota. Association for Computational Linguistics.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. Glm: General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California. Association for Computational Linguistics.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*.

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

Rémi Flamary, Nicolas Courty, Alexandre Gramfort, Mokhtar Z. Alaya, Aurélie Boisbunon, Stanislas Chambon, Laetitia Chapel, Adrien Corenflos, Kilian Fatras, Nemo Fournier, Léo Gautheron, Nathalie

T. H. Gayraud, Hicham Janati, Alain Rakotomamonjy, Ievgen Redko, Antoine Rolet, Antony Schutz, Vivien Seguy, Danica J. Sutherland, Romain Tavenard, Alexander Tong, and Titouan Vayer. 2021. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8.

Dan Flickinger, Emily M. Bender, and Stephan Oepen. 2014. Towards an encyclopedia of compositional semantics: Documenting the interface of the English Resource Grammar. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 875–881, Reykjavik, Iceland. European Language Resources Association (ELRA).

Jonas Groschwitz, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2017. A constrained graph algebra for semantic parsing with AMRs. In *IWCS 2017 - 12th International Conference on Computational Semantics - Long papers*.

Jonas Groschwitz, Matthias Lindemann, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2018. AMR dependency parsing with a typed semantic algebra. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1831–1841, Melbourne, Australia. Association for Computational Linguistics.

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond i.i.d.: Three levels of generalization for question answering on knowledge bases. In *Proceedings of the Web Conference 2021*, WWW '21, page 3477–3488, New York, NY, USA. Association for Computing Machinery.

Jiaqi Guo, Qian Liu, Jian-Guang Lou, Zhenwen Li, Xueqing Liu, Tao Xie, and Ting Liu. 2020. Benchmarking meaning representations in neural semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1520–1540, Online. Association for Computational Linguistics.

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

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

S. Hu, L. Zou, J. X. Yu, H. Wang, and D. Zhao. 2018. Answering natural language questions by subgraph matching over knowledge graphs. *IEEE Transactions on Knowledge and Data Engineering*, 30(5):824–837.

Srinivasan Iyer, Alvin Cheung, and Luke Zettlemoyer. 2019. Learning programmatic idioms for scalable semantic parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5426–5435, Hong Kong, China. Association for Computational Linguistics.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2018. Mapping language to code in programmatic context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1652, Brussels, Belgium. Association for Computational Linguistics.

Dora Jambor and Dzmitry Bahdanau. 2022. LAGr: Label aligned graphs for better systematic generalization in semantic parsing. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3295–3308, Dublin, Ireland. Association for Computational Linguistics.

Rohit J. Kate, Yuk Wah Wong, and Raymond J. Mooney. 2005. Learning to transform natural to formal languages. In *Proceedings of the 20th national conference on Artificial intelligence - Volume 3*, AAAI'05, page 1062–1068, Pittsburgh, Pennsylvania. AAAI Press.

Daniel Keysers, Nathanael Schärli, rli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. 2020. Measuring compositional generalization: A comprehensive method on realistic data. In *Proceedings of ICLR*.

Najoung Kim and Tal Linzen. 2020. COGS: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9087–9105, Online. Association for Computational Linguistics.

Yoon Kim. 2021. Sequence-to-sequence learning with latent neural grammars. In *Advances in Neural Information Processing Systems*, volume 34, page 26302–26317. Curran Associates, Inc.

Yoon Kim, Chris Dyer, and Alexander Rush. 2019a. Compound probabilistic context-free grammars for grammar induction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2369–2385, Florence, Italy. Association for Computational Linguistics.

Yoon Kim, Alexander Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. 2019b. Unsupervised recurrent neural network grammars. 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 1105–1117, Minneapolis, Minnesota. Association for Computational Linguistics.

Nikita Kitaev, Steven Cao, and Dan Klein. 2019. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy. Association for Computational Linguistics.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia. Association for Computational Linguistics.

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.

Zhenwen Li, Jiaqi Guo, Qian Liu, Jian-Guang Lou, and Tao Xie. 2022. Exploring the secrets behind the learning difficulty of meaning representations for semantic parsing. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3616–3625, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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

Peter Linz and Susan H Rodger. 2022. *An introduction to formal languages and automata*. Jones & Bartlett Learning.

Chenyao Liu, Shengnan An, Zeqi Lin, Qian Liu, Bei Chen, Jian-Guang Lou, Lijie Wen, Nanning Zheng, and Dongmei Zhang. 2021. Learning algebraic recombination for compositional generalization. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1129–1144, Online. Association for Computational Linguistics.

Inbar Oren, Jonathan Herzig, Nitish Gupta, Matt Gardner, and Jonathan Berant. 2020. Improving compositional generalization in semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2482–2495, Online. Association for Computational Linguistics.

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.

Gabriel Peyré, Marco Cuturi, et al. 2019. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Pawel Krzysztof Nowak, Tal Linzen, Fei Sha, and Kristina Toutanova. 2021. Improving compositional generalization with latent structure and data augmentation. *CoRR*, abs/2112.07610.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Tianze Shi, Jonathan Herzig, Emily Pitler, Fei Sha, and Kristina Toutanova. 2022. Evaluating the impact of model scale for compositional generalization in semantic parsing. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9157–9179, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1139–1149, Vancouver, Canada. Association for Computational Linguistics.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? 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 922–938, Online. Association for Computational Linguistics.

Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. 2019. Ordered neurons: Integrating tree structures into recurrent neural networks. In *International Conference on Learning Representations*.

Tian-Xiang Sun, Xiang-Yang Liu, Xi-Peng Qiu, and Xuan-Jing Huang. 2022. Paradigm shift in natural language processing. *Machine Intelligence Research*, 19(3):169–183.

Jens E. L. Van Gysel, Meagan Vigus, Jayeol Chun, Kenneth Lai, Sarah Moeller, Jiarui Yao, Tim O'Gorman, Andrew Cowell, William Croft, Chu-Ren Huang, Jan Hajič, James H. Martin, Stephan Oepen, Martha Palmer, James Pustejovsky, Rosa Vallejos, and Nianwen Xue. 2021. Designing a uniform meaning representation for natural language processing. *KI - Künstliche Intelligenz*, 35(3):343–360.

Antoine Venant and Alexander Koller. 2019. Semantic expressive capacity with bounded memory. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 65–79, Florence, Italy. Association for Computational Linguistics.

Pia Weißenhorn, Yuekun Yao, Lucia Donatelli, and Alexander Koller. 2022. Compositional generalization requires compositional parsers.

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.

Yuk Wah Wong and Raymond Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 439–446, New York City, USA. Association for Computational Linguistics.

Yuk Wah Wong and Raymond Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 960–967, Prague, Czech Republic. Association for Computational Linguistics.

Zhaofeng Wu. 2022. Learning with latent structures in natural language processing: A survey. *CoRR*, abs/2201.00490.

Chunyang Xiao, Marc Dymetman, and Claire Gardent. 2016. Sequence-based structured prediction for semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1341–1350, Berlin, Germany. Association for Computational Linguistics.

Xiaojun Xu, Chang Liu, and Dawn Song. 2018. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv:1711.04436 [cs]*.

Songlin Yang, Yanpeng Zhao, and Kewei Tu. 2021. PCFGs can do better: Inducing probabilistic context-free grammars with many symbols. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1487–1498, Online. Association for Computational Linguistics.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1321–1331, Beijing, China. Association for Computational Linguistics.

Pengcheng Yin, Hao Fang, Graham Neubig, Adam Pauls, Emmanouil Antonios Platanios, Yu Su, Sam Thomson, and Jacob Andreas. 2021. Compositional generalization for neural semantic parsing via span-level supervised attention. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2810–2823, 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 and Graham Neubig. 2018. TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 7–12, Brussels, Belgium. Association for Computational Linguistics.

Luke Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 678–687, Prague, Czech Republic. Association for Computational Linguistics.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, page 658–666, Arlington, Virginia, United States. AUAI Press. Event-place: Edinburgh, Scotland.

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.

Hao Zheng and Mirella Lapata. 2022. Disentangled sequence to sequence learning for compositional generalization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4256–4268, Dublin, Ireland. Association for Computational Linguistics.

Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar C Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James Duncan. 2020. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *Advances in Neural Information Processing Systems*, 33.

Terry Yue Zhuo, Zhuang Li, Yujin Huang, Fatemeh Shiri, Weiqing Wang, Gholamreza Haffari, and Yuan-Fang Li. 2023. On robustness of prompt-based semantic parsing with large pre-trained language model: An empirical study on codex. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 1090–1102, Dubrovnik, Croatia. Association for Computational Linguistics.

A Structure Modeling

We'll make extensions for Seq2Seq models. In the classical Seq2Seq, the encoder module is in charge of encoding source input $X = \{x_i\}_{i=1}^n$ and prepares for the attention mechanism a memory $H = \{h_i\}_{i=1}^n$ of states, where each h_i are usually aligned to each input token x_i. The decoder is obliged to generate tokens $Y = \{y_j\}_{j=1}^m$ by referring the memory H for each y_j. The last state h_n in memory is usually chosen to initialize autoregressive decoders. We will explain how H is constructed for encoders, and how Y is chosen for decoders.

A.1 Encoders

If the source structure is Absent, we take the input X as a plain sequence and choose the BiLSTM as the encoder. Due to their impressive performance, we also use the BERT (Devlin et al., 2019) and ELECTRA (Clark et al., 2020) language models from the Transformers library (Wolf et al., 2020). The encoder memory H is then the encoder outputs of each word in X.

If the source structure is Given, we use Berkeley Parser to get the constituency tree T of X. After removing the POS tags, T consists of words x_i as leaf nodes and the syntactic categories as internal nodes, such as NP, PP, and WHNP. We use two-layer GCN to encode nodes following the structure, and collect all the node hidden states as the attention memory H.

For latent structures, we choose representative grammar induction methods, namely ONLSTM (Shen et al., 2019), DIORA (Drozdov et al., 2019), PCFGs (Kim et al., 2019a; Yang et al., 2021), and Perturb-and-Parse (Corro and Titov, 2019b) Both constituency and dependency trees are considered. And most latent structures are learnt in two ways, by relaxation or sampling (Wu, 2022), where the former is usually optimized by maximizing the marginal probability of X as Eq.5, and the latter is optimized by sampling a structure S and passing to the downstream decoders (Eq.6).

$$\max_\theta P_\theta(X) = \sum_S P_\theta(S, X) \tag{5}$$

$$\max_\theta P_\theta(Y \mid X) = \mathbb{E}_{S \in P(S|X)} P(Y \mid S, X) \tag{6}$$

To wrap these up, the Perturb-and-Parse will give a sampling-based dependency trees, while others are the relaxation-based constituency trees.

ONLSTM Specifically, ONLSTM[6] shares the interface with classical RNNs, and invents the ordered neuron that can be interpreted as hierarchical structures. So we use it just as the replacement for BiLSTM. The memory H is also the states of sequence X, and the optimization only uses gradients from the decoders.

DIORA DIORA[7] aims to learn latent binary trees following the inside-outside algorithm. Embeddings of X are composed bottom-up for filling the inside chart with inside states. The composition c_{ijk} of two sub-span states $h(x_{i:j})$ and $h(x_{j:k})$ is parameterized by an MLP f_h. Every possible composition is scored with another MLP f_s. As DIORA falls into the relaxation-based category, each span state is a summation (Eq. 7) of all possible compositions with the normalized scores (Eq. 8).

$$h^{in}(x_{i:k}) = \sum_j s_j^{ik} f_h \left(h^{in}(x_{i:j}), h^{in}(x_{j:k})\right) \tag{7}$$

$$s_j^{ik} = softmax\left(f_s\left(h^{in}(x_{i:j}), h^{in}(x_{j:k})\right)\right)_j \tag{8}$$

where $softmax(\cdot)_j$ means the j-th normalized score after the softmax function. Similarly, the outside pass will fill the outside chart to given any span $x_{j:k}$ an outside state $h^{out}(x_{i:k})$, which is composed by each possible parent and sibling span and summed up with the normalized score. The outside composer and scorer are different MLPs. The attention memory H for DIORA encoders is full of representations of X, where each word is represented by the concatenation of inside and outside states as $h_i = [h^{in}(x_{i:i+1}); h^{out}(x_{i:i+1})]$, where the inside state of one-word spans $h(x_{i:i+1})$ are actually the word embeddings. Note that DIORA comes up with its own training objective, which

[6] `https://github.com/yikangshen/Ordered-Neurons`

[7] We use the original DIORA model from S-DIORA repo. `https://github.com/iesl/s-diora`

maximizes the reconstruction probabilities from each one-word span as Eq.9.

$$\max_\theta L^{diora} = \sum_i \log P_\theta(x_i | h^{out}(x_{i:i+1})) \quad (9)$$

PCFGs Two notable modern PCFGs[8] are C-PCFG (Kim et al., 2019a) and TD-PCFG (Yang et al., 2021). Rules are restricted to Chomsky normal form, including $S \to A, A \to BC$, and $P \to x$, where S is the fixed start token, A is a nonterminal, P generating a single terminal word x is called a preterminal, and B, C can be either nonterminal or preterminal. Embeddings and neural networks are used to parameterize the rule distributions as $\pi_{S\to A}, \pi_{P\to x}, \pi_{A\to BC}$.

C-PCFG adopted a novel variational model to infer a global state z of X, and let the neural nets predict π by concatenation z to each symbol embeddings. We use BiLSTM for the variational model. And TD-PCFG decomposed the large tensor of $\pi_{A\to BC}$ into the sum of products of lower rank tensors, largely extending the number of nonterminals and preterminals.

To use PCFGs as encoders, we first build up the PCFG models on the source sequence X. Since C-PCFG is built with a variational inference model, the loss involves a reconstruction loss as Eq.5 and a KL divergence. The former with a summation can be computed efficiently by the inside algorithm, and the latter is easy to obtain because the prior of z is kept Gaussian.

We choose to include all the span representations $h_{i:k}$ in the attention memory H. The representations are computed similar to the bottom-up inside algorithm. The algorithm fills an inside chart with probability scores $s_{ikA} \doteq P(x_{i:k} \mid A)$ for every span $x_{i:k}$ with each nonterminal A.

$$s_{ikA} = \sum_B \sum_C \sum_j w_{ijkABC} \quad (10)$$

$$w_{ijkABC} = \pi_{A\to BC} \cdot s_{ijB} \cdot s_{jkA} \quad (11)$$

Similarly, the span representation $h_{i:k}$ is also a weighted sum (Eq.12) of all h_{ijkABC}, which means the compositional representation for span $x_{i:k}$ as the category A, split at the point j, with left and right sub-spans being categories B and C.

$$h_{i:k} = \sum_{A,B,C,j} h_{ijkABC} \cdot w_{ijkABC} \cdot \pi_s(A) \quad (12)$$

[8]https://github.com/sustcsonglin/TN-PCFG

Note that we uses $\pi_s, s \in N$ as a prior to sum over A, which can be interpreted as treating the span $x_{i:k}$ as a valid sentence.

To compute compositions of span $h_{i:j}$ and $h_{j:k}$, Instead of concatenating embeddings of $A, B, C, h_{i:j}, h_{j:k}$ and transforming with an MLP, we factorize the computation of h_{ijkABC} with different MLPs to avoid broadcasting to the unrelated dimensions as Eq.13.

$$\begin{aligned} h_{ijkABC} &= f_h(A) + f_{ls}(B) + f_{rs}(C) \\ &\quad + f_l(h_{i:j}) + f_r(h_{j:k}) \end{aligned} \quad (13)$$

Note that we can rearrange Eq.12 and Eq.13 jointly to save up space, by moving items together and summing out irrelevant dimensions in advance. And for TD-PCFG which decomposes the tensor $\pi_{A\to BC} = \sum_l u_A^l \cdot v_B^l \cdot w_C^l$, the similar form of Eq.12 and Eq.13 and the efficiency trick can also be adopted. Formulae related to TD-PCFG are omitted here to save up space.

Perturb-and-Parse The model (abbr. PnP) focuses on sampling trees from the distribution of dependency structures. Words embeddings $e(X) \in \mathbb{R}^{n\times d}$ of $X \in \mathbb{R}^n$ are transformed to arc weights as Eq. 14, from which the Eisner's algorithm (Eisner, 1996) infers the tree S. The gumbel-softmax trick is adopted for differentiable sampling (Eq.15), and the argmax operation in Eisner's algorithm is replaced with the softmax following Corro and Titov (2019a). In this way, during training the output of Eisner's algorithm is not yet a valid but *soft* dependency tree, indicating the probabilities that there's an arc between two words x_i and x_j. But we switch to the default argmax during testing.

$$W = f_{head}(e(X)) \cdot f_{tail}(e(X))^T \quad (14)$$

$$Z \sim \mathcal{G}(0, 1) \quad (15)$$

$$S = Eisner(W + Z) \quad (16)$$

After the source tree S is inferred, we use two GCN layers to pass messages among nodes following the structures, where each node is a word in X. We use all the node representations to build the attention memory H. The PnP model is simply trained with the downstream tasks (Corro and Titov, 2019b).

A.2 Decoders

If the target structure is Absent, we simply model it with an LSTM. And we do not use any pretrained language model as the decoder. For datasets with very long targets and slow for training, such as the ATIS and Advising, we use the Transformer decoder instead of LSTM.

For latent target structures, we only use ONLSTM as the source side because it shares the same interface with RNNs. Other extensive works are not tested, because the SQLs are usually much longer than the natural language, and the grammar induction works are seldom evaluated on such long sentences (Drozdov et al., 2019). Furthermore, semantic representations are born with well-defined structures, it's not intuitive to learn latent structures from data.

For target structures that are given, we use the grammar induced by Oren et al. (2020) as discussed in Section 2. We manually convert the grammar into ENBF form and use the parser generator Lark to parse SQLs in the dataset. After that, we follow the order of left-most derivation to traverse the AST parses of SQLs as in TranX (Yin and Neubig, 2018), and the rule sequences are modeled by an LSTM. We denoted this method as Grammar-based Decoders as Oren et al. (2020).

Although the models above are enough to fulfill the taxonomy in Section 2, we've also tried but failed to use C-PCFG and RNNG (Dyer et al., 2016) as decoders. The generative RNNG is such expressive that make SQL grammar errors often, like a WHERE clause followed by another. URNNG (Kim et al., 2019b) requires an external (UCB Parser specifically) inference model to constrain the expressive power of RNNG. For C-PCFG, we hypothesize lacking of attention mechanism is crucial. We hypothesize the execution guided decoding might be helpful and necessary, but it's beyond our discussion in structures.

B Experiment Hyperparameters

We explain the details of models and hyperparameters here. We use the same setting for all datasets, and keep most parameters the same across models.

For hyperparameters applicable to all models, we use AdaBelief optimizer (Zhuang et al., 2020), and set the learning rate to 1e-3, and betas to 0.9 and 0.999. We do not use weight decays for all models. We fix the batch size to 16. The learning rate scheduler is based on NoamLR from the AllenNLP package, with the model size set to 400 and warmup steps to 50. We use the pretrained GloVe embeddings of 100 dimensions for the source side. For BERT hyperparameters, the learning rate is set 1e-5 and no LR schedulers.

Model	#examples	Accuracy
ChatGLM-6B	3000	6.27%
text-davinci-003	3000	0.83%
gpt-3.5-turbo	300	31.33%

Table 4: The in-context learning results of LLMs on the I.I.D. generalization of COGS. The testing set has the size 3000. The text-davinci-003 and gpt-3.5-turbo are evaluate on their May-15 2023 version. We didn't conduct a complete testing due to the accumulated accuracy and the cost.

We set the encoder hidden size to 300 for most models, except 150 for Diora and PnP, and 200 for PCFGs and Tree encoders. Sequence encoders and the inference model of C-PCFG are bidirectional (BiLSTM and ONLSTM). All encoders are 1-layer except the 2-layer GCN used for Tree and PnP encoders. Decoder is fixed to LSTM but Transformer for PCFGs/BERT/Electra. LSTM decoder is 1-layer and the hidden size is 200 for PCFGs models and 300 for others.The attention scores are computed by dot products. Transformer decoders are 2-layers and uses 300 for hidden size, and 10 for attention heads. All encoder dropout is 0 and decoder dropout is 0.5.

Training on GEO and Scholar uses 150 epochs for PCFGs and Tree encoders, 300 for tree encoders and 400 for others. All models trained for ATIS and Advising uses 30 epochs. On COGS and SMCalFlow-CS datasets, the models are trained for 15 epoches because of the large size. In practice, most models are trained in 4 to 12 hours, with an Xeon E5-2680 CPU and a single GeForce RTX 3090 GPU.

C Few-shot Parsing with LLMs

We just use the LLMs on the I.I.D. generalization of COGS dataset. We first build an index on the natural language of the training set, and then search for the closest 10 examples (x', y'), with each testing x. The prompt is typically built as "Input: x'. Output: y'." for each example (x', y'), appended by the testing example as "Input: x. Output:". In this way we're trying to utilize the in-context learning ability of LLMs for semantic parsing. and the accuracy is evaluated by Exact Match (EM) of the outputs against the gold targets. However, the performance is not ideal.

The lower two LLMs with the similar scale even have a pretty much performance difference. Note a plain Seq2Seq model can generalize well in the

I.I.D. setting, we find this performance not acceptable. We have sampled and analyzed the errors of ChatGLM, and there're some typical errors, such as (1) missing declarations of a variable; (2) output too long sequences which can be over ten times than the gold target; (3) inventing undefined the neo-davidsonian predicates; (4) misunderstanding the passive and active roles. We hypothesize that LLMs must be finetuned on these unseen representations like neo-davidsonian λ-calculus. And at least there're still much study to do before discussing the structural biases for LLMs.

D Accuracies for Model Combinations

We list the complete accuracies for each encoder and decoder combinations in Table 5 and Table 6. For the encoders, rcpcfg and rtdpcfg are the reduced version of C-PCFG and TD-PCFG respectively. The pnp is the Perturb-and-Parse model. The syn-parser is the supervised Berkeley Parser with a GCN to encode. For the decoders, the seq denotes an LSTM as the decoder, and the prod denotes the grammar-based decoding of rule sequences modeled by an LSTM. Please refer to Appendix A and Section 2 for an introduction.

We've defined several S and T choices. For encoders, the bilstm, bert, and electra are **absent** S. The ON-LSTM, DIORA, R-C-PCFG, R-TD-PCFG, and PnP are **latent** S. And only the syn-parser belongs to **given** S. For decoders, the seq, ON-LSTM, and prod represent the **absent**, **latent**, and **given** T, respectively.

E EBNF Grammar for SQL

For grammar-based decoding, AST parses of SQLs are required. We use the Lark Python package which is a parser generator like the classical flex and bison. We use the grammar induced by Oren et al. (2020) and manually convert it to the Lark format, which is an implementation of EBNF. Other grammars from MySQL and SQLite are not used in this work.

The lexer definitions we use are as follows.

```
SPACES: /[\u000B\x09\x0d\x0a\x20]/
SINGLE_LINE_COMMENT: "--"
    (/[^\x0d\x0a]/)* ("\x0D")? "\x0A"
WS: SINGLE_LINE_COMMENT | SPACES
%ignore WS

SCOL: ";"
COMMA: ","
STAR: "*"
WHERE: "WHERE"
SELECT: "SELECT"
DISTINCT: "DISTINCT"
LIMIT: "LIMIT"
GROUP: "GROUP"
ORDER: "ORDER"
BY: "BY"
HAVING: "HAVING"
AS: "AS"
AND: "AND"
OR: "OR"
DOT: "."
ASC: "ASC"
DESC: "DESC"
LPAR: "("
RPAR: ")"
LIKE: "LIKE"
NOT: "NOT"i
IN: "IN"
BETWEEN: "BETWEEN"
NULL: "NULL"
IS: "IS"
PLUS: "+"
MINUS: "-"
DIV: "/"
EQUAL: "="
NEQ: "<>"
GTE: ">="
LTE: "<="
GT: ">"
LT: "<"
UPPER: "UPPER"
LOWER: "LOWER"
FROM: "FROM"
```

The parser definitions are as follows.

```
statement: query SCOL | query
query: select_core groupby_clause
       orderby_clause limit
     | select_core groupby_clause
       orderby_clause
     | select_core groupby_clause limit
     | select_core orderby_clause limit
     | select_core groupby_clause
     | select_core orderby_clause
     | select_core
select_core: select_with_distinct
    select_results from_clause
    WHERE where_clause
  | select_with_distinct
    select_results from_clause
select_with_distinct: SELECT DISTINCT
                    | SELECT
select_results: select_result COMMA
  select_results
  | select_result
  | function binaryop non_literal_number
select_result: STAR
            | TABLE_NAME DOT STAR
            | col_ref
            | function AS COL_ALIAS
            | function
            | col_ref AS COL_ALIAS
from_clause: FROM source
source: single_source COMMA source
     | single_source
single_source: source_table
            | source_subq
source_table: "TABLE_PLACEHOLDER" AS TABLE_NAME
source_subq: LPAR query RPAR AS SUBQ_ALIAS
          | LPAR query RPAR
limit: LIMIT non_literal_number
```

encoder	decoder	smc16	smc32	smc64	smc128	advising	atis	cogs	geo	scholar
	seq	28.4	19.8	40.0	52.6	5.9	15.1	0.0	26.2	26.1
bilstm	onlstm	28.2	26.2	34.7	48.2	5.2	15.3	7.4	22.9	25.7
	prod	14.1	27.6	31.1	26.9	7.8	16.3	0.0	26.6	21.6
	seq	32.1	32.5	20.2	52.3	6.8	22.8	6.2	25.9	31.0
onlstm	onlstm	31.4	39.7	46.3	48.8	5.0	24.7	3.1	26.2	32.4
	prod	9.7	27.3	32.7	31.3	6.3	22.2	3.0	30.8	27.4
	seq	29.2	37.9	42.2	51.1	9.1	29.8	2.6	29.5	33.1
bert	onlstm	27.3	42.0	44.8	55.8	9.8	19.3	0.0	35.8	33.3
	prod	16.2	28.3	32.4	42.3	7.6	31.2	0.0	31.0	27.8
	seq	29.4	37.7	50.0	41.7	4.7	29.0	0.0	23.7	21.0
electra	onlstm	27.5	31.8	32.0	53.3	7.0	18.6	0.0	18.5	21.8
	prod	13.1	18.2	25.4	36.7	6.0	30.9	0.9	25.5	17.5
	seq	26.9	19.3	28.5	33.3	3.9	18.5	27.3	24.2	26.1
diora	onlstm	28.1	18.2	27.6	47.9	5.1	17.9	21.1	25.1	27.3
	prod	8.5	21.8	22.5	32.1	3.3	15.4	8.2	29.7	19.6
	seq	23.2	21.4	23.7	40.2	2.8	11.0	0.0	17.6	14.9
rcpcfg	onlstm	22.2	18.3	32.3	26.2			0.0	14.7	12.9
	prod	17.3	16.2	20.2	12.2	1.7	11.8	0.0	17.8	15.5
	seq	21.5	24.1	19.9	23.2	0.7	1.4	0.0	16.9	16.1
rtdpcfg	onlstm	9.4	23.3	26.6	32.1			0.0	12.5	12.9
	prod	6.3	17.3	14.5	15.7	1.5	3.4	0.0	13.2	13.2
	seq	19.5	20.1	29.6	24.8	6.3	12.3	0.0	18.5	22.9
pnp	onlstm	17.1	19.2	20.5	21.9	6.2	17.1	0.0	20.9	20.4
	prod	6.8	12.5	18.9	24.5	3.3	16.4	0.0	25.7	19.8
	seq	23.8	27.2	28.8	39.2	11.4	16.4	0.0	22.0	30.4
syn-parser	onlstm	24.3	27.6	37.4	40.9	9.3	16.0	0.0	21.3	30.6
	prod	6.8	17.1	21.1	31.7	7.8	17.4	0.0	23.7	21.4

Table 5: The accuracies of each datasets on their compositional generalization levels. For the ATIS, GEO, Scholar and Advising, average results of 5 random seeds are reported.

```
    | LIMIT value
where_clause: LPAR where_clause RPAR where_conj
  | LPAR where_clause RPAR where_or
  | LPAR where_clause RPAR
  | unaryop where_clause
  | expr where_conj
  | expr where_or
  | expr
  | source_subq binaryop non_literal_number
where_conj: AND where_clause
where_or: OR where_clause
groupby_clause: GROUP BY group_clause
    HAVING expr
  | GROUP BY group_clause
group_clause: expr COMMA group_clause
          | expr
orderby_clause: ORDER BY order_clause
order_clause: ordering_term COMMA order_clause
          | ordering_term
ordering_term: expr ordering
            | expr
            | COL_ALIAS ordering
ordering: ASC
      | DESC
col_ref: SUBQ_ALIAS DOT COLUMN_NAME
      | TABLE_NAME DOT COLUMN_NAME
      | SUBQ_ALIAS DOT COL_ALIAS
      | TABLE_NAME DOT COL_ALIAS
expr: in_expr
  | value LIKE value
```

```
  | value NOT LIKE value
  | value BETWEEN value AND value
  | value NOT BETWEEN value AND value
  | value binaryop expr
  | unaryop expr
  | col_ref IS NOT NULL
  | col_ref IS NULL
  | source_subq
  | value
in_expr: value NOT IN string_set
      | value IN string_set
      | value NOT IN expr
      | value IN expr
      | value IN LPAR arg_list RPAR
string_function: string_fname
    LPAR col_ref RPAR
string_fname: LOWER | UPPER
parenval: LPAR expr RPAR
function: fname LPAR DISTINCT
      arg_list_or_star RPAR
  | fname LPAR arg_list_or_star RPAR
  | "YEAR(CURDATE())"
arg_list_or_star: arg_list
              | STAR
              | "1"
arg_list: expr COMMA arg_list
      | expr
non_literal_number: "1"
                | "2"
                | "3"
```

encoder	decoder	smc128	advising	atis	cogs	geo	scholar
bilstm	seq	57.8	86.2	61.8	94.3	70.7	67.8
	onlstm	61.5	86.2	60.9	98.5	70.9	67.2
	prod	24.1	82.3	57.5	56.4	71.5	66.0
onlstm	seq	62.9	82.1	63.6	99.3	71.2	66.3
	onlstm	63.2	82.3	61.9	96.3	72.2	65.5
	prod	19.8	80.5	58.9	95.2	71.3	61.0
bert	seq	51.8	89.9	67.0	97.4	75.8	69.3
	onlstm	54.9	88.7	62.0	66.7	75.8	70.3
	prod	25.1	87.1	65.3	46.7	75.7	68.5
electra	seq	50.0	90.1	66.7	96.2	72.2	71.6
	onlstm	48.2	87.7	58.2	92.6	71.8	69.0
	prod	23.2	86.8	66.4	83.0	69.8	65.6
diora	seq	55.6	66.3	52.0	85.1	70.7	64.4
	onlstm	54.3	68.2	50.4	78.2	68.8	65.0
	prod	16.3	61.5	50.9	56.5	68.9	62.7
rcpcfg	seq	50.0	81.7	58.0	96.2	48.2	57.1
	onlstm	51.2			95.9	60.8	56.8
	prod	18.0	80.1	58.3	88.3	59.6	52.1
rtdpcfg	seq	42.1	77.0	55.0	96.7	54.7	55.0
	onlstm	45.6			94.5	61.5	56.2
	prod	16.0	59.6	53.5	85.5	54.5	50.4
pnp	seq	43.9	83.6	56.5	60.4	67.6	66.7
	onlstm	44.2	84.3	53.5	57.8	67.4	66.7
	prod	14.9	81.9	53.2	61.9	66.2	65.8
syn-parser	seq	15.3	75.2	57.7	83.0	60.6	57.1
	onlstm	49.4	75.0	52.7	92.2	62.1	56.2
	prod	18.5	72.2	53.5	77.5	60.2	51.3

Table 6: The accuracies of each datasets with the I.I.D. generalization. Similar to the CG level, average results of 5 random seeds are reported for the ATIS, GEO, Scholar, and Advising datasets

```
                 | "4"
                 | "0"
                 | "5"
                 | "100"
string_set: "'" string_set_vals "'"
string_set_vals: value COMMA string_set_vals
               | value
fname: "COUNT"
     | "SUM"
     | "MAX"
     | "MIN"
     | "AVG"
     | "ALL"
boolean: "true"
       | "false"
binaryop: PLUS
        | MINUS
        | STAR
        | DIV
        | EQUAL
        | NEQ
        | GTE
        | LTE
        | GT
        | LT
        | LIKE
unaryop: PLUS
       | MINUS
       | NOT
```

We put values in the grammar definition following Oren et al. (2020). This good enough for our usage. Note in a formal SQL grammar, the values for entities, tables, and columns are usually included in the lexer definition and defined with regular expressions. We leave the other definitions in our code release because it's too long (hundreds of lines), including the nonterminals of *value*, *COL_ALIAS*, *SUBQ_ALIAS*, *TABLE_NAME*, and *COLUMN_NAME*.

F EBNF Grammar for COGS

We list our handcrafted grammar for COGS here.

```
start: preludes formulas
     | formulas
     | lambdas formulas
     | PROPER_NOUN
lambdas: lambda DOT lambdas?
lambda: LAMBDA var
preludes: prelude SEMICOLON
        | prelude SEMICOLON preludes
prelude: ASTERISK? NOUN LPAR var RPAR
var: LETTER
   | LETTER UNDERSCORE NUMBER
formulas: formula
        | formula AND formulas
formula: predicate LPAR params RPAR
```

```
predicate: NOUN
         | NOUN DOT NOUN
         | NOUN DOT NOUN DOT NOUN
params: param
      | param COMMA params
param: var
     | PROPER_NOUN
AND: "AND"
NOUN: WORD
LETTER: /[a-z]/
NUMBER: /\d+/
LPAR: "("
RPAR: ")"
WORD: /[a-z]+/
PROPER_NOUN: /[A-Z][a-z]+/
DOT: "."
COMMA: ","
SEMICOLON: ";"
ASTERISK: "*"
UNDERSCORE: "_"
LAMBDA: "LAMBDA"
```

G EBNF Grammar for Lispress

We list our handcrafted grammar for SMCalFlow-CS, which uses the Lispress language. Although the Lispress has an official parser in Python, we still use a handcrafted grammar for consistency with our work.

```
VALID_CHAR: /[a-zA-Z\d\"\#\(\)\+/
          | /\.\:\<\>\=\?\[\]\~]/
QUOTE: "\""
LPAR: "("
RPAR: ")"
LBRA: "["
RBRA: "]"
COLON: ":"
DOT: "."
LET: "let"
DO: "do"
META: "^"
MACRO: "#"

SYMBOL_CHAR: /[a-zA-Z0-9\+\<\>\=\?\~]/
CAP_CHAR: /[A-Z]/
NONCAP_CHAR: /[a-z0-9\+\<\>\=\?\~]/

CAP_SYMBOL: CAP_CHAR SYMBOL_CHAR*
NONCAP_SYMBOL: NONCAP_CHAR SYMBOL_CHAR*
ANY_SYMBOL: SYMBOL_CHAR+

PLAIN_STRING: /(\\.|[^\\\"])+/

COMP_SYMBOL: /\?[^ ]+/
REAL_NUMBER: /\d+(\.\d+)/
INT_NUMBER:  /\d+/
LONG_NUMBER: /\d+L/
TYPE_CONSTRUCTION: "apply"
STRING_TYPENAME: "String"
NUMBER_TYPENAME: "Number"
BOOLEAN_TYPENAME: "Boolean"

start: s_exp
s_exp: LPAR type_args? fn_call RPAR
     | LPAR value RPAR
type_args: META LPAR (NUMBER_TYPENAME
    | BOOLEAN_TYPENAME
    | STRING_TYPENAME) RPAR
```

```
fn_call: kwarg_fn | arg_fn
kwarg_fn: kwarg_fn_name kwarg*
arg_fn: arg_fn_name arg*
kwarg_fn_name: CAP_SYMBOL
             | CAP_SYMBOL LBRA ANY_SYMBOL RBRA
arg_fn_name: kw_name
           | LET
           | DO
           | NONCAP_SYMBOL
           | type_name DOT attribute
kwarg: kw_name arg
kw_name: COLON ANY_SYMBOL
arg: s_exp
   | value
   | variable
variable: NONCAP_SYMBOL
value: typed_literal | old_typed_literal
old_typed_literal: MACRO LPAR STRING_TYPENAME
  QUOTE string_literal QUOTE RPAR
 | MACRO LPAR STRING_TYPENAME QUOTE QUOTE RPAR
 | MACRO LPAR NUMBER_TYPENAME number_literal RPAR
 | MACRO LPAR BOOLEAN_TYPENAME boolean_literal RPAR
 | MACRO LPAR type_name QUOTE string_literal QUOTE RPAR
typed_literal: boolean_literal
 | META BOOLEAN_TYPENAME boolean_literal
 | QUOTE string_literal QUOTE
 | QUOTE QUOTE
 | META STRING_TYPENAME QUOTE string_literal QUOTE
 | META STRING_TYPENAME QUOTE QUOTE
 | number_literal
 | META NUMBER_TYPENAME number_literal
 | type_name DOT TYPE_CONSTRUCTION
string_literal: PLAIN_STRING
number_literal: REAL_NUMBER | INT_NUMBER | LONG_NUMBER
boolean_literal: "true" | "false"
type_name: CAP_SYMBOL
attribute: NONCAP_SYMBOL
```