Grammar-based Decoding for Improved Compositional Generalization in Semantic Parsing

Jing Zheng and **Jyh-Herng Chow** and **Zhongnan Shen** and **Peng Xu**
Ant Technologies U.S., Inc
{jing.zheng, jyhherngchow, zhongnan.shen, peng.x}@antgroup.com

Abstract

Sequence-to-sequence (seq2seq) models have achieved great success in semantic parsing tasks, but they tend to struggle on out-of-distribution (OOD) data. Despite recent progress, robust semantic parsing on large-scale tasks that combine challenges from both compositional generalization and natural language variations remains an unsolved issue. To encourage research in this area, this work introduces CUDON, a large-scale dialogue dataset in the Chinese language, specifically created to evaluate the compositional generalization of semantic parsing. The dataset contains about ten thousand multi-turn complex queries, and provides multiple splits with different degrees of train-test distribution divergence. We have investigated improving compositional generalization through grammar-based decoding on this dataset. With specially designed grammars that leverage program schema, we are able to significantly improve the accuracy of seq2seq semantic parsers on OOD splits: a LSTM-based parser using a Context-free Grammar (CFG) achieves over 25% higher accuracy than a standard seq2seq baseline; a parser using Tree-Substitution Grammar (TSG) improves parsing speed by five to seven times over the CFG parser with only a small accuracy loss. The grammar-based LSTM parsers also outperforms BART- and T5-based seq2seq parsers on the OOD splits, despite having less than one tenth of the parameters and no pretraining. We also validated our approach on the SMCalflow-CS dataset, specifically on the zero-shot learning task.

1 Introduction

The task of semantic parsing is to translate a natural language utterance into an executable program in certain meaning representation (MR). In recent years, sequence-to-sequence (seq2seq) models have become the dominant approach for this task (Jia and Liang, 2016; Dong and Lapata, 2016), mainly due to their excellent ability to handle natural language variations. However, evaluations on synthetic datasets such as SCAN (Lake and Baroni, 2018) have shown that seq2seq models do not reliably generalize to those out-of-distribution (OOD) utterances with element combinations not seen in the training data. The ability to generalize to novel element combinations is commonly known as *compositional generalization*.

To improve compositional generalization, many recent studies propose specially-designed model architectures targeting SCAN-like synthetic datasets, and some have achieved perfect accuracy on some of these datasets (Li et al., 2019; Russin et al., 2019; Gordon et al., 2020; Lake, 2019; Liu et al., 2020a; Nye et al., 2020; Chen et al., 2020). However, most of these approaches have only been tested on synthetic datasets, which are constructed for a specific purpose and cannot represent the full range of challenges that a real-world semantic parser must address.

This motivates us to study the compositional generalization problem in a more realistic setting, specifically, in the context of task-oriented dialogue systems, where semantic parsers must handle greater task complexity and navigate through a wider range of natural language variations and ambiguities. The contributions of this work are two-fold: first, on the dataset front, we have created CUDON[1], a Chinese dialogUe Dataset for compOsitional geNeralization research. This is a new large-scale task-oriented dialogue dataset, which contains complex multi-turn cross-domain dialogues, created in a semi-synthetic manner, and specially tailored for studying the compositional generalization problem. It also provides 8 train/dev/test splits with different level of distribution divergence. To the best of our knowledge, this is the first large-scale Chinese dataset designed for this purpose. And the dataset being in Chinese makes it even more valuable since the similar resources are scarce.

Second, on the modeling front, we find that by

[1]https://github.com/ant-research/dialog-dataset-for-compositional-semantic-parsing

Findings of the Association for Computational Linguistics: ACL 2023, pages 1399–1418
July 9-14, 2023

exploiting schema of the target program, including the syntax and function argument lists, in the paradigm of grammar based decoding (Krishnamurthy et al., 2017; Yin and Neubig, 2017; Scholak et al., 2021), we can substantially improve compositional generalization of semantic parsers in the seq2seq framework. Both the parser using a context free grammar (CFG) and the one using a tree-substitution grammar (TSG) (Cohn et al., 2010) exceed a standard seq2seq baseline using Pointer Generator Network (See et al., 2017), by over 20 percent on the OOD splits, with the latter being 5-7 times as fast as the former. And they even compare favorably with parsers using large pretrained language models with more than 10 times of parameters. This level of improvement was not reported before in similar works. We also verified that the proposed approach also works in the SMCalflow-CS (Yin et al., 2021) dataset without much tuning, achieving significant improvement in the zero-shot setting over the baseline, and exceeds previously reported results with pretrained language models in low-resource settings. In addition, we have not seen the TSG formalism used in grammar-based decoding before, although we adopt a model architecture similar to Platanios et al. (2021).

2 Dataset

We have constructed a large-scale multi-turn cross-domain task-oriented dialogue dataset in the financial assistant scenario over seven domains and 52 intents. There is also a *common* domain for intents such as *goodbye* and *thankyou*, which can occur in all dialogues. The raw material of entities and properties, such as stock prices and gains, or the experience and best return of fund managers, was crawled from various websites. Throughout the course of dialogue generation, the system may carry out the user-requested tasks by consulting this data to produce realistic and coherent results.

The MR we use is called *function expressions*, which allows arbitrarily nesting of function invocations. There are two types of functions: one with only named arguments, each associated with an intent in the dialogue system, and its arguments associated with the slots of the intent; the other can have both positional and named arguments, and perform non-interactive operations like data manipulation.

Overall, the dataset has 9996 dialogues, with an average of 41 turns per dialogue and about 44% of the turns involve multiple domains. There are 3876 different delexicalized patterns of the target function expressions with highly unbalanced frequencies. We use the function *refer* to reference slots filled in previous turns, aided with *constraint* functions to limit search matching. This idea, borrowed from Andreas et al. (2020a), makes utterances seemingly independent to each other in a multi-turn dialogue. Figure 1 shows an example of a multi-turn dialogue.

U1	我想查基金。找评级是至少是【4】星的 Show me some funds that are at least 4-star rated **FE:** fund/search_fund(em_rating='4', op='>=')
S2	找到多个，包括【中欧短债债券C, 创金合信货币C, 渤海汇金量化汇盈灵活配置混合】 Found several, including 【China-Europe Short-Term Bonds/C, Chuangjin Hexin Currency/C, Bohai Huijin Quant Fund Hybrid】
U3	第3个的基金公司叫什么？近一季度的涨跌幅怎样? The third one. Tell me the fund company, and its performance in the last quarter. **FE:** do(common/describe_prop(obj=user_select(fund/search_fund/result, '3'), prop='fund_company_name'), common/describe_prop(obj=common/describe_prop/obj, prop='pct_3m'))
S4	基金公司是【渤海汇金证券资产管理有限公司】。最近3个月年涨跌幅是【9.49】% The company is 【Bohai Huijin Securities Asset Management Co.】。It was up 【9.49】% in the last quarter.
U5	【中高风险】这个风险等级的有哪些基金? How about funds with medium to high risk type? **FE:** fund/search_fund(fund_risk_type='中高风险')
S6	找到多个，包括【华泰柏瑞量化创优灵活配置混合, 华泰柏瑞价值增长混合A, 银华和谐主题混合】 Found several, including【Huatai Borui Quant Fund Hybrid, Huatai Borui Value Growth Hybrid/A, Yinhua Harmony Theme Mix】
U7	最近半年华泰价值的涨跌幅是多少? What's the performance of Huatai Value Growth in the last 6 months? **FE:** common/describe_prop(obj=user_select(fund/search_fund/result, '华泰价值'), prop='pct_6m')
S8	最近6个月涨跌幅是【13.71】% The fund was up 【13.71】% in the last 6 months.
U9	股票评测。【之前基金的重仓股】股票 How's the evaluation of the stocks that it invests in heavily ? **FE:** stock/evaluate_stock(stock_name=get_property(refer(constraint[fund]()), 'heavy_holdings'))
S10	着重在化工行业，标准普尔，富时罗素板块。今日共有几篇资讯和该公司相关，总体情绪偏正面，总体影响度为★★☆☆☆ They focus on chemical industry, S&P, and Russell. There are a few related news today, and overall mood is positive but not very influential
U11	谢谢你啦 Thank you **FE:** common/thankyou()

Figure 1: Example of a multi-turn dialogue. User turns are annotated with function expression (FE).

```
do(
   hotel/reserve_hotel(
      hotel_name = user_select(
        hotel/find_hotel(location = 'West Lake')),
      occupancy = '2',
      date = 'Sunday'),
   restaurant/find_restaurant(
      type = 'fastfood',
      location = hotel/find_hotel/address))
```

Figure 2: A function expression example

2.1 Function Expression as the Goal

A common approach to task-oriented multi-turn dialogue generation is to use an agenda-based user simulator (Schatzmann et al., 2007). This approach, using a flat-structured goal representation defined by multiple tuples of intent, act, and slot-value pairs, is incapable of describing dataflows across multiple intents in a single dialogue turn.

Instead of the flat-structured goal, we propose specifying a goal as a *function expression*, that can explicitly express the composition of intents and complex dataflow in a dialogue. For example, the goal *"find a hotel near West Lake, reserve a room for two people for Sunday, and check if there is a fast food restaurant nearby"*, can be expressed as shown in Figure 2, where the `user_select` function triggers the system to ask the user to select one result from the `find_hotel` intent.

Similar to Wang et al. (2015); Rastogi et al. (2020), we take a two-step approach. A goal is first generated from a set of production rules to create the function expression and its corresponding canonical utterance simultaneously. However, instead of simply paraphrasing this canonical utterance, we use the goal to drive a user simulator to generate a multi-turn dialogue where each user turn has its own function expression and canonical utterance. In the second step, we paraphrase the user utterances in this multi-turn dialogue to make them more natural and fluent.

Appendix A describes the production rules used for goal generation in more detail. Appendix B describes how function expressions are used to generate multi-turn dialogues, and shows the goal that generates the dialogue shown in Figure 1.

2.2 Paraphrasing user turns

The generated canonical utterances are then paraphrased by crowd workers and a fine-tuned mBART model (Liu et al., 2020b). Our work focuses on the user-side modeling, so we only paraphrase the user utterances. We instruct the crowd workers to paraphrase every user turn of 500 dialogues, and paraphrase only three longest user turns of a dialogue for the remaining. We ask them to rewrite the utterances in accordance with the context and encourage them to express the same meaning but with different sentence structures. A dialogue is paraphrased by a worker, cross checked and possibly revised by a different worker, and finally verified and possibly modified by a third person.

Then we collect all the human paraphrased data (a total of 38895 samples) to train an mBART-based paraphrasing model, which is then used to paraphrase all remaining data. Inspired by recent prompt-based approaches (e.g., Liu et al. (2021)), during training we utilize the first word of each training sample as the prompt to initiate the paraphrase 90% of the time, while randomly selecting a word from anywhere in the sentence for the remaining 10%. This methodology ensures that the model is likely to begin the generated sentence with the prompt word, while still allowing for flexibility to use the prompt word in other positions when necessary. This approach improves the diversity and robustness of the model by generating a wider range of paraphrased outputs. During inference, for each utterance we provide the model with three different prompts based on the pattern of its function expression. We then filter out those results that have modified entity values. In the end, using the mBART model, we are able to paraphrase 94.8% of the remaining data.

2.3 Train/Test splits

The primary goal of this dataset is to help research of compositional generalization, so we want to split the data into train and test in multiple ways for experimentation. To facilitate data splitting, we pair each utterance with its context to form a data sample to make them mutually independent, allowing arbitrary way of splitting. In this work we use the user utterance from the previous round as context. Though this may not be the optimal setting for training the most accurate model, we think it is a reasonable compromise between flexibility and quality. For each split, we reserve a portion of randomly selected test samples as the validation data for checkpoint selection, and the rest for evaluation, on which we report results in the experiments. It is important to note that the validation data must not be used for model training, as it shares similarity with the evaluation data.

IID Split We provide a standard random split of data samples, where train and test set have similar data distribution. We dedupe each set to get unique context-utterance pairs. We also remove samples from validation and evaluation data if they appear in the training set.

TMCD Splits Shaw et al. (2021) proposes Target Maximum Compound Divergence (TMCD) for data splitting. Each target expression is repre-

sented as a sequence of atoms, where a compound describes the composition of the atoms. Thus, compound divergence measures the difference in the composition between the train and test sets.

In this study, we define atoms as the tokens in function expressions and compounds as the parent-child relationship in function invocations. The compounds of a concrete example can be found in Appendix C.

To study compositional generalization, following the MCD principle (Keysers et al., 2020), we aim to generate splits that have low atom divergence between the train and test sets, while having a significant difference in the representation of compounds in the train and test sets. To experiment with TMCD splits, we fix atom divergence at 0.1, and run 200 different splits to get their compound divergence, and then select six splits with varying compound divergence for model experiments.

Length Split Length split is also often used in evaluating compositional generalization (e.g., in SCAN). Here we define the length of a function expression to be the max number of arguments among all the functions in the expression. The training set contains all the samples whose expression length is smaller than or equal to two. Those samples with expression length larger than two are divided into validation and test set.

Appendix C describes these splits in more detail and contains plots that show the atom and compound distributions in the train/test set.

3 Approach

To improve compositional generalization, we exploit the program schema to enhance seq2seq models in a grammar-based decoding framework. The model takes in the user utterance and the context as input, and outputs a sequence of rules and associated values, which can be used to construct the program. Intuitively, if the parser is aware of the argument list of each function, it can avoid predicting arguments not belonging to this function. And grammar can also help avoid generating ill-structured outputs.

3.1 Context Free Grammar (CFG)

We design a context free grammar that defines all valid function expressions. Figure 3 shows an example derivation tree for the below function expression: `f1(a1=f2(b1="abc"), a2=123)`, where `f1` and `f2` are function names; `a1`, `a2`, and `b1` are

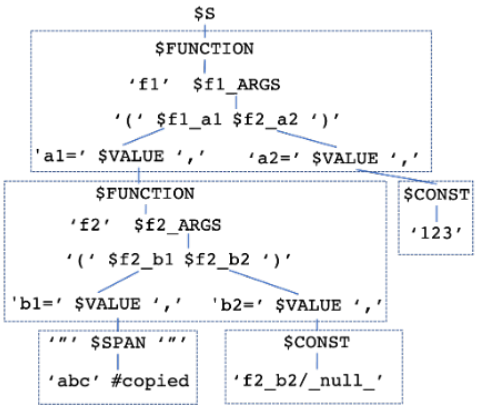


Figure 3: Derivation tree for function expression `f1(a1=f2(b1="abc"), a2=123)`. Nonterminal symbols start with `$`, and terminal symbol values are single quoted. Function `f2` has an optional argument `b2`. Each solid line represents a CFG rule; each box represents an elementary tree in TSG.

argument names; `"abc"` is a string literal copied from the input utterance, and `123` is a constant. Assuming that based on schema function `f2` has an optional argument `b2` that is not included in this expression. We use specific nonterminals to represent function argument lists and each of the arguments. The output rule sequence for this example can be found in Appendix D. The grammar induction process is described in and Appendix E.

Some of the grammar design choices are made for greater generalizability. First, we introduce the nonterminal `$VALUE` to represent all types of argument values. An argument nonterminal must first rewrite as `$VALUE`, then as more specific value types. This ensures at grammar level every argument can be assigned to any value type even if it does not exist in the training data.

Second, we always use a function's full named argument list in the rules, but allow each argument to have argument specific null values, e.g., `f2_b2/_null_` in Figure 3. This way, the grammar is immune to novel combinations of function arguments in test data. However, this requires generation of uninvolved arguments and can negatively impact inference speed.

Model Architecture

As shown in Figure 4, we design an encoder-decoder model with span-copying mechanism for the CFG-based semantic parsers. The decoder takes rule id as input, and uses model output to cross attend to the encoder output, and pass the result to an output layer, which uses a softmax function to produce rule id distributions, from which

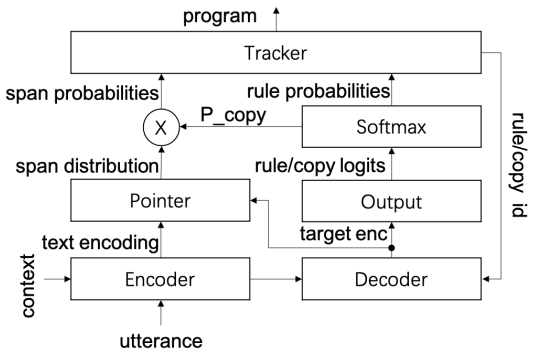


Figure 4: CFG parser model architecture

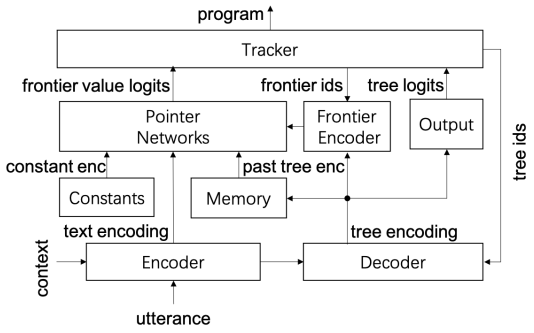


Figure 5: TSG parser model architecture

the tracker decides the best next rule or span ids. Appendix D explains in detail how rules and spans are indexed into integer ids.

The output vocabulary includes all rule ids plus a special id (i.e. copy id) representing span copying, whose probability P_copy is used as a multiplier for span probabilities to ensure proper distribution. The decoder output and the projected encoder output are fed into a pointer network (Vinyals et al., 2015), which produces distribution for all input spans with an attention layer. Note, each input span can be viewed as a dynamic rule that rewrites `$SPAN` as the actual content, therefore span ids are used for represent these rules.

The rule distribution (excluding the copy id) and the scaled span distribution are fed to the tracker, which generates the target program and ensure its validity. At each step, the tracker creates a mask based on its internal state to filter out illegal rules, picks the most probable legal rule id to update the internal state, and feeds the next rule id to the decoder to continue the loop, until a special end id is predicted. Note if a span id is selected at a certain step, the copy id is sent to the decoder, so that the decoder is agnostic of the content being copied. The prediction is performed in the order of top-down pre-order traversal, resulting in the leftmost derivation.

To train the model, in addition to the encoder input, the trainer needs the sequence of rule ids and span ids as labels, which are obtained during grammar induction. We train the model using the negative log likelihood (NLL) loss in the extended vocabulary space with both rule ids and span ids, and use the exact match metric to pick the checkpoint. More details about the model can be found in Appendix F.

Our approach is similar to Yin and Neubig (2017), but we use very different grammar design, which targets greater compositional generalization. We also use span-level copy, like in Platanios et al. (2021), instead of token-level copy. We also don't use an abstract syntax tree, but a domain-specific grammar induced from the training data.

3.2 Tree-Substitution Grammar (TSG)

For a typical seq2seq model, inference time is closely related to the number of steps to produce a complete output. An obvious approach to boosting speed is to reduce necessary output steps, i.e., the total number of rules required to construct the whole derivation tree in our case. This motivates us to explore TSGs, where each production rule rewrites a frontier nonterminal as an entire elementary tree, and therefore requires significantly fewer generation steps.

In fact, we can convert the aforementioned CFG into a TSG in a straightforward manner by packing rules responsible for generating a tree fragment into an elementary tree, as illustrated by the boxes in Figure 3. This way, we can create a TSG out of the original CFG, with much fewer generation steps required in the entire derivation. More details and examples can be found in G, which also describes the grammar induction process.

Model Architecture

At each generation step the model first predicts the rewrite rules for all the frontiers of the last predicted tree in parallel, and then the next elementary tree id. Note, predicting a frontier rewrite rule is equivalent to finding the right-hand-side elementary tree. Because of the parallel nature of frontier prediction, this approach greatly improves parsing speed over the CFG parser, especially for trees with many null-valued frontiers. Predicting frontiers independently may have advantage in compositional generalization as well, though it could also lead to errors caused by conflicting argument predictions.

The model architecture is illustrated in Figure 5. Similar to the CFG parser, the TSG parser also uses an encoder-decoder model to predict tree ids, where the decoder takes input from the encoder and the past tree ids. We mix each tree's embedding with the max pooling of related frontier embeddings before feeding to the decoder. This leads to a small improvement over using embedding purely based tree ids.

Frontier prediction is performed separately, by matching the frontier encoding vector against value encoding vectors from three different sources: previous tree instances, constants, and input text spans. Details can be found in Appendix H.

The tracker then finds the best match for each frontier, and the next tree id, which is fed back to the decoder for the next prediction. It accumulates logits of frontier values and tree ids to find the derivation hypothesis with the highest combined score from both frontier and tree predictions. The tracker also checks potential conflicts between frontier values, and penalizes the violating hypotheses. This provides a slight accuracy improvement in beam search settings. Different from the CFG parser, the TSG parser makes prediction in a bottom-up post-order manner, so that the are trees predicted at a later step can attend to tree encoding vectors produced at earlier steps.

Model training requires target labels as a sequence of elementary tree instances, each includes the tree id, frontier kind and value id information. We combine the negative log-likelihood (NLL) loss from tree id prediction and cross entropy (CE) loss from frontier value prediction for optimization. At inference time, the tracker assembles prediction outputs into the tree instances, from which the program is constructed. More details about the model can be found in Appendix H.

4 Experiments

In this section we compare parsing accuracy and speed performance of all parsers. We use exact match accuracy as the metric, with minor normalization to filter out non-significant variations and to sort named arguments (order insensitive). Table 1 summarizes the sample sizes of train/validation/evaluation data of all the splits, along with compound divergence values.

Baseline We train and evaluate baseline parsers using the OpenNMT-py toolkit (version 1.2.0). The baseline model is configured as a source-to-target text translator, using a pointer generator network with two layers of bidirectional LSTMs as encoder and two layers of LSTMs as decoder. We found a small gain by using 300-dimension pretrained embeddings from FastText[2] for initialization. Since we observe large accuracy variations between models trained with different random seeds, to have a clear picture of the degree of variations, we run training 55 times with different seeds to collect the medians and other statistics. We find on TMCD splits, the standard deviation of accuracies range from 6.4 to 11.3 percent. For this reason we want to use large evaluation sets to reduce random effects. More details can be found in Appendix I.

	train	**valid**	**eval**	**div**
IID	132800	32402	41329	9e-5
Tmcd1	154586	21045	31662	0.1008
Tmcd2	150437	22808	34048	0.1360
Tmcd3	154840	20846	31607	0.2055
Tmcd4	151011	22767	33515	0.2350
Tmcd5	153169	21690	32434	0.3422
Tmcd6	155732	20771	30790	0.3792
Length	196397	1074	9822	0.5021

Table 1: Statistics of sample sizes and compound divergence values.

Grammar-based Parsers To ensure meaningful comparison against the baseline, grammar-based parsers also use bi-directional LSTM (Hochreiter and Schmidhuber, 1997) as the encoder, unidirectional LSTM as the decoder, and the same pretrained word embeddings for initialization. We make sure that grammar-based models have similar parameter sizes as the baseline (around 11M).

Pretrained Models Since LSTM is not today's state of the art (SOTA) in modeling, to verify if our approach is still competitive compared against stronger models, we also train two additional parsers using T5 (Raffel et al., 2020) and BART (Lewis et al., 2019) models. Both were pretrained on a subset of the CLUE corpus (Xu et al., 2020) using UER-py (Zhao et al., 2019). The parameter sizes for the T5 [3] model is 214M, and the BART [4] model 117M. We fine tune these models on our data as text-to-text translators. Considering the differences in training data and parameter

[2]https://fasttext.cc/docs/en/crawl-vectors.html

[3]https://huggingface.co/uer/t5-small-chinese-cluecorpussmall

[4]https://huggingface.co/uer/bart-base-chinese-cluecorpussmall

ONMT	BART	T5	CFG	TSG
96.9	97.7	98.3	97.6	97.2

Table 2: Accuracy (%) on the IID split.

	ONMT	BART	T5	CFG	TSG
Tmcd1	51.8	71.7	**73.4**	52.3	49.1
Tmcd2	51.0	60.5	48.9	**77.5**	77.1
Tmcd3	31.5	**52.8**	41.3	47.9	49.1
Tmcd4	28.8	52.3	37.6	48.7	**54.4**
Tmcd5	24.9	41.9	42.1	**57.1**	54.4
Tmcd6	27.3	35.8	32.0	55.8	**61.0**
Length	0	0	0	**55.9**	39.9
Average	30.8	45.0	39.3	**56.4**	54.3
Tmcds	35.9	52.5	45.9	**56.5**	**56.7**

Table 3: Accuracy (%) on all the OOD splits. Average refers to equal-weight accuracy average on all splits; Tmcds to the average accuracy on TMCD splits only.

sizes, the comparison is not strictly fair, but can help evaluate the competitiveness of our approach.

We observe that besides the baseline, other parsers also have accuracy variations, but in much lighter degrees. For these parsers, we run model training five times with different random seeds, and report median accuracy, which appears reasonably stable in multiple experiments.

4.1 Results on IID Split

We first evaluate the models on the in-distribution data. As Table 2 shows, T5 has slightly higher accuracy than other parsers, which is expected given it has the largest parameter size and is pretrained on extra data. The rest are in the same ballpark, showing that the grammar-based parsers are competitive on the IID split.

4.2 Results on OOD Splits

Results of all the parsers on the OOD data are summarized in Table 3, from which we have the following observations: First, both CFG and TSG beat the baseline by very large margins: 25.6% for CFG and 23.5% for TSG. They also outperform BART and T5, especially clear on the Length split, where ONMT, BART and T5 all have zero accuracy. CFG and TSG parsers have about the same accuracy on the TMCD splits, but the former has a 16% lead on the Length split, probably because the CFG parser detects null-valued arguments more reliably with serial argument prediction.

T5 and BART lead the ONMT baseline almost everywhere, confirming the advantages of large pretrained models over LSTM. They are also the leaders on Tmcd1, which has the smallest divergence among all the OOD splits. However, on splits with higher divergence, they tend to perform much worse, especially for the T5 parser. This indicates large pretrained language models do not necessarily have sufficient compositional generalizabilites.

	IID	TMCDs	Length	OOD Avg.
CFG-f	**97.6**	39.9	2.5	39.9
CFG	**97.6**	**56.5**	**55.9**	**56.4**

Table 4: Comparison between CFG-f (no schema) and CFG (with schema).

Batch	ONMT	BART	T5	CFG	TSG
1	93.9	462.3	1179.8	96.5	19.5
32	17.0	29.9	65.8	22.2	3.2

Table 5: Average parsing time in millisecond per utterance on Tesla P100 GPU.

4.3 Importance of Schema

To examine the importance of schema information, we experimented with a different CFG design, which uses a right-recursion rule to generate arbitrary argument sequence, until the underlying model predicts the 'end-of-argument-list' rule. This is similar to the strategy used in Yin and Neubig (2018). This grammar, denoted CFG-f, still guarantees valid target syntax, but does not have knowledge of function schema. Table 4 shows that without using schema, CFG-f still matches CFG on the IID split, but falls far behind on all OOD splits, especially on the 'length' split. We believe that using the schema info is the key for achieving large gain on OOD data, and this is one of the main differences between our approach and many prior works.

4.4 Parsing Speed

We test parsing speed of all the parsers on the first 4000 examples of the IID test set on a Tesla-P100 GPU. For fair comparison all parsers use greedy search inference mode, whose accuracy is very close to beam search on this dataset. As shown in the Table 5, the TSG parser is the clear winner in parsing speed: in both single and batch inference mode, it is five to seven times as fast as the CFG parser, five times as ONMT. BART and T5 models are much slower because of much larger parameter sizes, especially in the single inference mode, where GPU computation power cannot be fully utilized.

Task	0	8	16	32
ONMT	30.6	38.2	42.3	45.0
TSG	37.2	39.1	39.6	45.7
T5	-	34.7	44.7	59.0
BERT2SEQ	-	-	33.6	53.5

Table 6: Few-shot learning result on SMCalFlow-CS. The first row lists number of shots.

4.5 Few-Shot Learning

SMCalFlow (Andreas et al., 2020b) is a large-scale English dialogue dataset tailored for semantic parsing research. SMCalflow-CS is a subset of SMCalflow containing only single-turn utterances that cover two skills, calendar events and org chart. The training set mostly contains single-skill utterances, plus a limited number of composite samples involving both skills. The test set has composite samples only. Multiple few-shot learning tasks are created based on the number of composite samples (or shots) in the training set. The zero-shot task is the most challenging one to evaluate a model's generalizability, and we did not find results reported from prior works.

For this experiment, we compare the TSG parser against the ONMT baseline, and two other parsers using pretrained models, whose results are cited from previous work. In order to use our current implementation on this dataset, we convert the programs in SMCalflow-CS to our MR in a lossless manner. We use exactly the same model configurations and parser setup from experiments mentioned earlier. As Table 6 shows, on the zero-shot task, the TSG parser outperforms the ONMT baseline significantly, and even beats the 8-shot T5 (Qiu et al., 2021) and the 16-shot BERT2SEQ result (Yin et al., 2021) despite using much smaller models without pretraining. When the number of shots increases, the gap between ONMT and TSG diminishes, similar to what we observe on our own dataset when the divergence decreases. T5 and BERT2SEQ become significantly better, motivating us to investigate more sophisticated models to catch up on higher-shot tasks.

4.6 Error Analysis

We performed error analysis and find that the majority of errors are related to argument prediction errors, as shown in Table 7. The two parsers have similar overall accuracies on the TMCD splits, though the TSG parser tends to miss more arguments, which explains the large accuracy gap between the two on the Length split, where errors overwhelmingly come from argument missing. Currently we use a simple argument-specific vector to model null-valued arguments, which has large room for improvement.

Looking more closely into details, we found some individual errors, e.g., confusing one specific function with another, can occur very frequently on some splits, accounting for a significant portion of the total errors. This is most likely caused by a highly frequent pattern in the test data but rare in training, resulting large numbers of errors.

We notice some errors can be fixed by leveraging additional schema information. For example, sometimes arguments expecting numeric values are assigned non-numeric strings. By incorporating type checking, like in Platanios et al. (2021) and Krishnamurthy et al. (2017), this problem can be effectively addressed. Also, boundary errors of copied entities can be largely resolved by applying fuzzy match against a dictionary, for many entities with known categorical values.

Error Type	CFG	TSG
Function prediction	16.9	19.5
Argument missing	4.0	9.6
Argument insertion	8.0	4.8
Argument confusion	9.4	5.5
Argument value	9.3	10.8
Overall	44.0	45.6

Table 7: Average percentages of utterances with each error type on the TMCD splits. Overall refers to the average utterance error rate. Since a single utterance can have multiple types of errors, the sum of error rates of all categories is larger than the over error rate.

5 Related Work

Dataset Compositional generalization research used to focus on fully synthetic datasets such as SCAN (Lake and Baroni, 2018), COGS (Kim and Linzen, 2020), NACS (Bastings et al., 2018), CFQ (Keysers et al., 2020), and so on. Recently researchers have also conducted experiments on non-synthetic datasets, such as GeoQuery (Zelle and Mooney, 1996), and SPIDER (Yu et al., 2018), trying to attack both compositional generalization and natural language variation challenge together. The most similar dataset to ours is SMCalFlow-CS(Yin et al., 2021), which is extracted from SMCalflow(Andreas et al., 2020b), a large scale dialogue data set, for the purpose of compositional

generalization research. However, our dataset has many more domains, multi-turn dialogues, more complex conversations, and different ways of splitting data. More importantly, our dataset is in Chinese with few similar resources.

Modeling For small-scale synthetic data, many specialized model architectures proved to be effective on SCAN like tasks (Li et al., 2019; Russin et al., 2019; Gordon et al., 2020; Lake, 2019; Liu et al., 2020a; Nye et al., 2020; Chen et al., 2020). To also address natural language variations in non-synthetic tasks, some recent works exploit structure of the source input and its relation to the target side (Herzig and Berant, 2021; Shaw et al., 2021; Weißenhorn et al., 2022), and employ source-side parsing that can be computationally demanding for long sentences, and may have coverage challenge and not available in all languages; while we try to exploit target-side structure only for higher efficiency. Some other works leverage source-side structure for data augmentation to overcome distribution divergence (Yang et al., 2022b; Qiu et al., 2022), which can clearly help but is not the focus of this paper. Grammar-based decoding has shown to help semantic parsing on in-distribution data (Krishnamurthy et al., 2017; Yin and Neubig, 2017). Oren et al. (2020) also look into compositional generalization, and find the accuracy gain from grammar-based decoding is small and inconsistent across all datasets. We think one of the main reasons for our improvement lies in different grammar design, which can leverage program schema to counter distribution divergence. Yang et al. (2022a) decompose a text-to-sql task into a sequence of subtasks, each to fill slots of a subclause by prompting preptrained language model. This in spirit is remotely similar to our approach, which predicts one function and its argument at each step, but with very different model architectures and on very different domains. The model architectures used in this paper bear resemblance to some of prior works. The model of the CFG parser is similar to Yin and Neubig (2017, 2018), except that we use span-level copying instead of token-by-token copying, and different grammar. And the span copying approach is previously used by Platanios et al. (2021), which also inspired us on the design of the TSG parser. Lastly, although LSTMs are not viewed state of the art, they can perform surprisingly well in some semantic parsing tasks comparing to large pretrained models, for example in Yang et al. (2021).

6 Conclusions

Compositional generalization remains an unsolved problem for real-world semantic parsers. We propose a semi-synthetic large-scale task-oriented dialogue dataset intended to promote research in this area. We find that by designing appropriate grammars, leveraging schema information, and choosing right model architecture we can substantially improve both compositional generalization and parsing efficiency. We will investigate applying grammar-based decoding approach to large pretrained language models on more challenging tasks.

Limitations

The schema information and grammar design are domain specific. We have tested our approach mainly on our own dataset, though we believe the similar approach can be applied to other tasks, as long as the meaning representation involves functions and arguments. Also, we have not explored all approaches to obtain the highest possible accuracy on this dataset, because our main goal is to show the effectiveness of the proposed approach, which we believe is clearly demonstrated by the current result. At this stage, the difference between in-distribution and out-of-distribution accuracies remain very large, and there is a large room for further improvements. We hope by releasing this dataset we can help promote research in related areas.

Ethics Statement

This work proposes to release a new dataset to research community for improving compositional generalization in semantic parsing. The goal of the research topic is to make model work better on underrepresented language without relying on large training data and therefore long training time. The models proposed in this paper are very small in today's standard and do not require large computing resources for training and evaluation. We particularly pay attention to efficiency of models to save computing resource after deployment.

Acknowledgments

We want to thank Kuan Xu, Zujie Wen and Yongliang Wang for their supports in dataset generation, and Yi Su for helpful discussions.

References

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

Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, et al. 2020b. Task-oriented dialogue as dataflow synthesis. *Transactions of the Association for Computational Linguistics*, 8:556–571.

Jasmijn Bastings, Marco Baroni, Jason Weston, Kyunghyun Cho, and Douwe Kiela. 2018. Jump to better conclusions: SCAN both left and right. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 47–55, Brussels, Belgium. Association for Computational Linguistics.

Xinyun Chen, Chen Liang, Adams Wei Yu, Dawn Song, and Denny Zhou. 2020. Compositional generalization via neural-symbolic stack machines. In *Advances in Neural Information Processing Systems*, volume 33, pages 1690–1701. Curran Associates, Inc.

Jianpeng Cheng, Devang Agrawal, Héctor Martínez Alonso, Shruti Bhargava, Joris Driesen, Federico Flego, Dain Kaplan, Dimitri Kartsaklis, Lin Li, Dhivya Piraviperumal, Jason D. Williams, Hong Yu, Diarmuid Ó Séaghdha, and Anders Johannsen. 2020. Conversational semantic parsing for dialog state tracking. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8107–8117, Online. Association for Computational Linguistics.

Trevor Cohn, Phil Blunsom, and Sharon Goldwater. 2010. Inducing tree-substitution grammars. *Journal of Machine Learning Research*, 11:3053–3096.

Róbert Csordás, Kazuki Irie, and Juergen Schmidhuber. 2021. The devil is in the detail: Simple tricks improve systematic generalization of transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 619–634, Online and Punta Cana, Dominican Republic. 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.

Jonathan Gordon, David Lopez-Paz, Marco Baroni, and Diane Bouchacourt. 2020. Permutation equivariant models for compositional generalization in language. In *International Conference on Learning Representations*.

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

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

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

Mihir Kale and Abhinav Rastogi. 2020. Template guided text generation for task-oriented dialogue. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6505–6520, Online. Association for Computational Linguistics.

Daniel Keysers, Nathanael Schärli, 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 *International Conference on Learning Representations*.

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.

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1516–1526, Copenhagen, Denmark. Association for Computational Linguistics.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills

of sequence-to-sequence recurrent networks. In *International conference on machine learning*, pages 2873–2882. PMLR.

Brenden M. Lake. 2019. Compositional generalization through meta sequence-to-sequence learning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 9788–9798.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Yuanpeng Li, Liang Zhao, Jianyu Wang, and Joel Hestness. 2019. Compositional generalization for primitive substitutions. 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 4293–4302, Hong Kong, China. Association for Computational Linguistics.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586*.

Qian Liu, Shengnan An, Jian-Guang Lou, Bei Chen, Zeqi Lin, Yan Gao, Bin Zhou, Nanning Zheng, and Dongmei Zhang. 2020a. Compositional generalization by learning analytical expressions. *Advances in Neural Information Processing Systems*, 33:11416–11427.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020b. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742.

Maxwell Nye, Armando Solar-Lezama, Josh Tenenbaum, and Brenden M Lake. 2020. Learning compositional rules via neural program synthesis. *Advances in Neural Information Processing Systems*, 33:10832–10842.

Inbar Oren, Jonathan Herzig, Nitish Gupta, Matt Gardner, and Jonathan Berant. 2020. Improving compositional generalization in semantic parsing. *arXiv preprint arXiv:2010.05647*.

Emmanouil Antonios Platanios, Adam Pauls, Subhro Roy, Yuchen Zhang, Alexander Kyte, Alan Guo, Sam Thomson, Jayant Krishnamurthy, Jason Wolfe, Jacob Andreas, and Dan Klein. 2021. Value-agnostic conversational semantic parsing. 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 3666–3681, Online. Association for Computational Linguistics.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Pawel Nowak, Tal Linzen, Fei Sha, and Kristina Toutanova. 2022. Improving compositional generalization with latent structure and data augmentation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4341–4362, Seattle, United States. Association for Computational Linguistics.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Paweł Krzysztof Nowak, Tal Linzen, Fei Sha, and Kristina Toutanova. 2021. Improving compositional generalization with latent structure and data augmentation. *arXiv preprint arXiv:2112.07610*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. 2020. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8689–8696.

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

Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. 2007. Agenda-based user simulation for bootstrapping a POMDP dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 149–152, Rochester, New York. Association for Computational Linguistics.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, 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.

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

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1332–1342, Beijing, China. Association for Computational Linguistics.

Pia Weißenhorn, Lucia Donatelli, and Alexander Koller. 2022. Compositional generalization with a broad-coverage semantic parser. In *Proceedings of the 11th Joint Conference on Lexical and Computational Semantics*, pages 44–54, Seattle, Washington. Association for Computational Linguistics.

Liang Xu, Xuanwei Zhang, and Qianqian Dong. 2020. Cluecorpus2020: A large-scale chinese corpus for pre-training language model. *arXiv preprint arXiv:2003.01355*.

Jingfeng Yang, Federico Fancellu, Bonnie Webber, and Diyi Yang. 2021. Frustratingly simple but surprisingly strong: Using language-independent features for zero-shot cross-lingual semantic parsing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5848–5856, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jingfeng Yang, Haoming Jiang, Qingyu Yin, Danqing Zhang, Bing Yin, and Diyi Yang. 2022a. SEQZERO: Few-shot compositional semantic parsing with sequential prompts and zero-shot models. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 49–60, Seattle, United States. Association for Computational Linguistics.

Jingfeng Yang, Le Zhang, and Diyi Yang. 2022b. SUBS: Subtree substitution for compositional semantic parsing. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 169–174, Seattle, United States. 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.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. 2019. CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1962–1979, Hong Kong, China. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055.

Zhe Zhao, Hui Chen, Jinbin Zhang, Xin Zhao, Tao Liu, Wei Lu, Xi Chen, Haotang Deng, Qi Ju, and Xiaoyong Du. 2019. Uer: An open-source toolkit for pre-training models. *EMNLP-IJCNLP 2019*, page 241.

Appendices

A Production Rules for Goal Generation

Figure 6 illustrates our production rules to generate a goal. The right hand side of a rule can be associated with a `[weight]` and a `[canonical`

utterance]. The [weight] is used to influence the sampling probability. This handy mechanism allows the designer to focus differently on certain generation. The [canonical utterance] is synchronously generated as the rule expands, where it can use #i to refer to the text generated by i-th $rhs symbol.

The outermost production rule also allows the && connector for sequential composition, which will be described below. To control the complexity of generated expressions, we use a parameter *nested_rate* to control the depth of nesting, and *rate_decay* to indicate how fast the probability decays when the generation goes deeper.

Composition of intents mainly come from the following:

- dataflow through object property (slot): for example, at Figure 6 line 6, $FundCompanyName may come from a property of a $FundManager (at line 15), which could come from search_fund_manager for fund_manager_name='xxx', resulting in this expression: search_fund(fund_company_name = get_property(search_fund_manager (fund_manager_name = 'xxx'), 'current_company'), i.e., "*tell me some funds under the company where fund manager 'xxx' works for*".

- through the && operator: the production rule S1 && S2 denotes the sequential composition of S1 and S2. S2 and S1 are not necessarily independent, as S2 can refer to the result of S1 (as in the Figure 2 find_restaurant example).

B Function Expression to Dialog

We have described how goals and their canonical utterances are generated. Although a goal itself can be a valid single-turn user request to the system, and the system is expected to execute the whole function expression, but a single turn with such complexity is very unlikely to happen in reality.

Similar to the agenda-based simulator, we use this goal to drive multi-turn dialogues. In our work, a goal is divided into subgoals, and the dialogue generation proceeds with each subgoal one-by-one, respecting their dependency. For this dataset, we choose to let each subgoal comprise one intent function, so that given a subgoal, multi-turns can be generated with an existing traditional agenda-based approach. This does not mean the resulting turns in the dialogue will become simply slot filling, as they can still contain nested simple functions, and multi-intent interactions occur through direct slot references, the refer function, or sequential composition

To generate canonical utterance for each turn during dialogue generation, we use a templated approach similar to the T2G2 approach (Kale and Rastogi, 2020).

User turns are annotated with function expressions. As an example, for the goal in Figure 7, which means *"find some funds with at least 4-star rating and medium-to-high risk level, and I want to know how their heavy holdings perform"*, the tool generates the dialogue in Figure 1. Notice that the search_fund has 3 arguments, but in the dialogue the user may not inform all 3 slots in a single turn. In this example, the user informs the em_rating first and the system does a search_fund on rating. Then the user later informs the fund_risk_type; note that even though the function expression of this turn contains only the fund_risk_type argument, the system does a search_fund with not only fund_risk_type but also the slots known already from the context (to be more precisely, when these slots are not dependent of each other). Also notice that to create more realistic dialogues, the tool automatically create turns where the user may randomly request further information about the system's results to obtain the request type of dialogue acts.

C Atom and Compound Distribution

Keysers et al. (2020) defines the similarity of two weighted distributions P and Q, using the Chernoff coefficient as

$$C_\alpha(P||Q) = \sum_k p_k^\alpha q_k^{1-\alpha} \in [0, 1]$$

and the divergence of the two sets is 1 minus this similarity. They use α=0.5 for atom divergence and α=0.1 for compound divergence.

To compute atom and compound divergence, we first delexicalize the expression, i.e., mask all quoted constants with placeholders; then following TMCD (Shaw et al., 2021), we define atoms to be the tokens in the function expressions, and compounds are the parent-child relation in function calls. For the example in Figure 7, it consists of the following compounds:

```
fund/search_fund( fund_risk_type, - , - )
fund/search_fund( - , em_rating , - )
fund/search_fund( - , - , op )
```

```
FundIntents = [5] $Funds [#1]
            | $GetFundPrediction [#1]
            | ...
            ;
Funds =  ...
       | search_fund(fund_company_name=$FundCompanyName)  [【#1】旗下的基金]       # funds under【#1】
       ;
GetFundPrediction = fund_prediction(fund_name=$FundName) [【#1】的走势预测]       # trend prediction for【#1】
       | ... ;
Fund = user_select($Funds) 【#1】
     | ... ;
FundCompanyName = @Dictionary(fund_company_names_both.txt)
     | [5] get_property($Fund, 'fund_company_name')         [【#1】的公司]        # the fund company of【#1】
     | [5] get_property($FundCompany, 'fund_company_name')  [【#1】的公司]        # the fund company name of【#1】
     | [5] get_property($FundManager, 'current_company')    [【#1】的公司]        # the fund company【#1】works at
     ;
```

Figure 6: Examples of production rules for goal generation

```
stock/evaluate_stock(
  stock_name = get_property(
    user_select(fund/search_fund(
      fund_risk_type = '中高风险',
      em_rating = '4',
      op = '>=')),
    'heavy_holdings'))
```

Figure 7: An example goal as function expression

```
user_select( fund/search_fund )
get_property( select , - )
get_property( - , ENTITY)
stock/evaluate_stock( stock_name )
```

In Figure 8, we plot the atom and compound distribution in train (purple) and test (green) set for these splits.

For IID split, as shown in Figure 8(a), both atom distribution (top) and compound distribution (bottom) show nearly identical distribution in the train set and the test set. X-axis are the atoms (or compounds), sorted by their frequency in the set and y-axis is the log(frequency), where atom divergence is 0.00003932, and compound divergence is 0.00008713.

For TMCD splits, Figure 8(b) shows that atoms in the test set are all in the training set; on the other hand, Figure 8(c) shows some compounds in the test set are not in the training set.

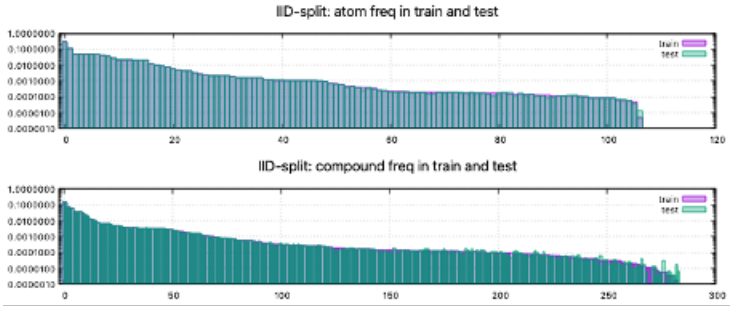
For the Length split, as shown in Figure 8(d), the atom and compound divergence are quite high: atom divergence is 0.2470, and compound divergence is 0.5021. The distribution shows that while all the atoms in the test set do appear in the training set, many compounds in the test set do not appear in the training set.

D CFG used by Semantic Parser

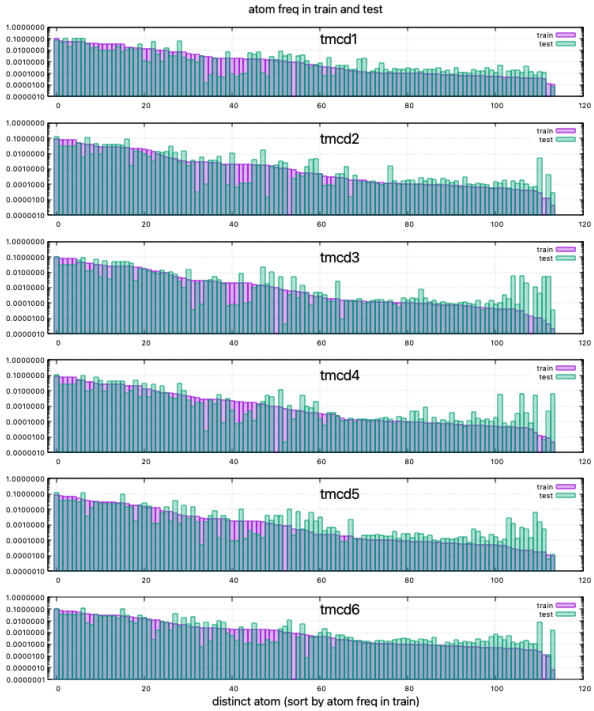
The CFG grammar used for semantic parsing in this paper is particularly designed for function expressions. In a derivation process, the start nonterminal (NT) symbol `$S` first rewrites `$FUNCTION` representing generic function type, which further rewrites as a function name followed by a NT representing the invocation argument list. This NT then rewrites as a list of NTs representing all related arguments. As explained in the paper, we elect to include all named arguments declared in the function's schema, no matter whether they are actually used in the invocation. Then all the argument NTs will first rewrite as a generic NT `$VALUE`. If the argument was not used in the invocation, `$VALUE` will rewrite as a specific terminal symbol representing a "null" value, which will be filtered out in postprocessing; otherwise, `$VALUE` will rewrite as one of four value types: 1) `$CONSTANT`, which next rewrites as a terminal symbol for a unique string literal; 2) `$SPAN`, next as span of text in the input; 3) `$FUNCTION`, next as a corresponding function name and argument list NT as described above; 4) `$SLOT_REF`, next as reference to a slot, which can be either a leaf reference, represented by a terminal symbol with content in the form of `<domain>/<intent>/<slot>`; or a recursive reference, including a `$FUNCTION` followed by a terminal symbol with content `/<slot>`.

Taking the example in Figure 3, the sequence of the rules used in the derivation are as follows:

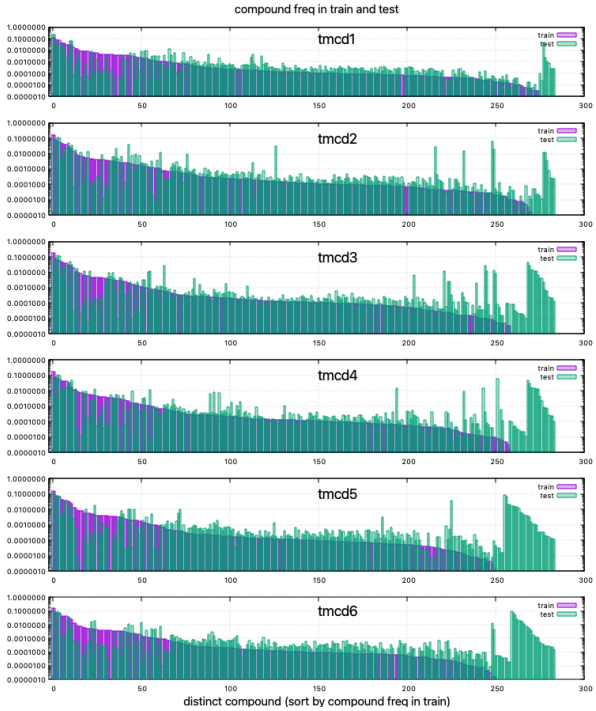
```
1. $S -> $FUNCTION  # start rule
2. $FUNCTION-> 'f1' $f1_ARGS
3. $f1_ARGS -> '(' $f1_a1  $f2_a2 ')'
4. $f1_a1 -> 'a1=' $VALUE ','
5. $VALUE -> $FUNCTION
6. $FUNCTION -> 'f2' $f2_ARGS
7. $f2_ARGS -> '(' $f2_b1 $f2_b2 ')'
8. $f2_b1 -> 'b1=' $VALUE ','
9. $VALUE -> '"' $SPAN '"'
10. $SPAN -> 'abc'  # copied from input
11. $f2_b2 -> 'b2=' $VALUE ','
12. $VALUE -> $CONSTANT
13. $CONSTANT -> 'f2_b2/_null_' # null val
14. $VALUE -> $CONSTANT
15. $f2_a2 -> 'a2=' $VALUE
16. $VALUE -> $CONSTANT
```



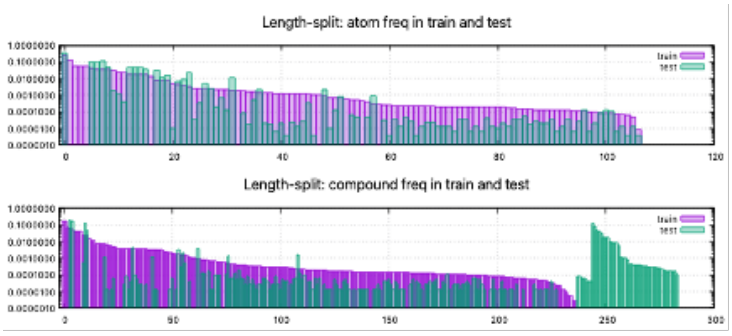
(a) Atom and compound distribution on IID.



(b) Atom distribution on TMCD splits



(c) Compound distribution on TMCD splits



(d) Atom and compound distribution on Length.

Figure 8: Atom & compound distributions on all data splits: train set in purple and test set in green

```
17. $CONSTANT -> '123'  # constant
```

As can be easily seen the yield of the derivation is

```
f1(a1=f2(b1="xyz", b2=f2_b2/_null_), a2=123",)
```

which can be transformed to the correct form with a regex in the post processing stage. During inference, given input text, the model needs to predict the above derivation rules, which yield the program. In order to manage these rules, we assign each of them an integer id, ranging from 0 to $V-1$, where V is the number of rules excluding the copy rule. We also assign each span an index starting from V, in the order of the span's end position and then start position. Assuming the input utterance has U tokens, there are $N = (U+1) * U/2$ total spans. The decoder predicts a sequence of above index ids, which are used by the tracker to retrieve rules and spans, and generate the program as yield.

E CFG Induction

In order to obtain a CFG for semantic parsing, we need to perform grammar induction from the training data. The process is straightforward: First we implement a parser given the syntax of function expression language. Then we parse all the function expressions in the training data, extract the rules based on the parse, and save them in a grammar file. For string literals in the function expressions, if they appear in the utterance, we use span copy rule, otherwise, we add a constant rule. If the string literal appears multiple times in the source data, we always use the left most one, and masked span out to prevent duplicated copies from the same text. This simple strategy may encounter occasional bad cases, but overall, it is working as expected. During grammar induction, we also process the data and generate additional information needed by trainer, basically the rule id sequences, plus the span indexes. For each derivation, we always start from the rule `$S -> $FUNCTION` with a fixed id, and always append a special id marking the end of sequence, which the parsing model is trained to predict.

F Modeling Details of CFG Parser

The entire model architecture is illustrated in Figure 4. Below we describe the implementation details of each of the boxes.

Encoder For encoder input, for convenience to use BERT model, we pack tokens of the context and the utterance together in the following way, which also works for other type of encoders:

```
[CLS] <context tokens> [SEP] <utt tokens> [SEP]
```

After encoding, we pass the vectors to both the decoder and the pointer, but with different masks. For the decoder input, we allow both context and utterance vectors being passed, but for the pointer input, we mask out the context part, only allow the utterance vectors to be used to avoid copying from context. This way we can apply different types of encoders, such as LSTMs and transformers, in a unified manner.

Decoder The decoder can have multiple layers, each layer applies the decoder model layer to the decoder input, and then cross-attends to the encoder output vectors using a multihead attention layer:

$$\mathbf{h}_m = \mathbf{model}(\mathbf{h}_i)$$
$$\mathbf{h}_o = \mathbf{h}_m + \mathbf{MHA}(\mathbf{h}_m, \mathbf{h}_{enc}, \mathbf{h}_{enc})$$

where $\mathbf{model}$ can be RNN, transformers etc., $\mathbf{MHA}$ stands for multi-head attention, $\mathbf{h}_i$ is the input to the decoder layer, can be either the output vector from the last decoder layer, or for the first decoder layer, the embedding vector of the current rule id.

Pointer We use span copying instead of token copying. Each span is encoded as the concatenation of the encoding vector of the begin and end position of the span. The logit of a span [b, e] is computed as:

$$logit_{b,e} = \mathbf{v}^\intercal \mathbf{tanh}(\mathbf{W}_s[\mathbf{h}^b_{enc}; \mathbf{h}^e_{enc}] + \mathbf{W}_t \mathbf{h}_{dec})$$

where b and e are the span's begin and end positions; $\mathbf{h}^b_{enc}$ and $\mathbf{h}^e_{enc}$ are the corresponding encoding vectors from the encoder; ";" means vector concatenation; $\mathbf{h}_{dec}$ is the decoder output; Matrix $\mathbf{W}_s$ and $\mathbf{W}_t$ and vector $\mathbf{v}$ are trainable parameters. Applying a *softmax* to the logits gives us the span distributions.

Tracker The tracker searches the derivation with the highest probability, whose yield is the target program, i.e., function expression. In order to ensure validity of derivation, the tracker maintains a stack, containing unprocessed rule's right-hand-side elements. At the top of the stack is always a NT, which must match the predicted rule's left-hand-side NT at next step, so that the rule can be applied. To ensure the predicted rules are valid, the tracker applies a pre-computed mask, according to the stack-top NT, on the prediction logits, so that invalid predictions are filtered out.

At each step, after receiving a valid rule id, the tracker pops out the NT at the top of the stack, and pushes in the right-hand-side symbols of the rule in the reverse order. Then it pops out all terminal symbols (including the copied span) one by one to generate the target output, until the top symbol is an NT or the stack is empty. Should it be the latter case, the tracker will only accept the special end symbol at the next step, and then stop the derivation and return the result; Otherwise the tracker will feed the decoder with predicted rule id, or the copy id if a span id is predicted.

G Converting CFG to TSG

In the example in Appendix D, we can pack the rule 2, 3, 4, 15 together into a TSG rule:

```
$FUNCTION -> 'f1'  $f1_ARGS ( '('
  ( $f1_a1  ( 'a1=' $VALUE  ',' )
    $f2_a2 (  'a2=' $VALUE ','  )
  ) ')'
)
```

and pack rule 6, 7, 8, 13 into another TSG rule:

```
$FUNCTION -> 'f2' $f2_ARGS ( '('
  ( $f2_b1 ( 'b1=' $VALUE ',' )
    $f2_b2 ( 'b2=' $VALUE ',' )
  ) ')'
)
```

These two rules each have two leaf NTs, `$VALUE`, which are called frontier NTs, or simply frontiers. In addition, the rule 9, 13, 17 can be viewed as degenerated TSG rules with height 1. To save the extra derivation step to rewrite `$VALUE` as `$FUNCTION`, we combine it with the rules above to create new rules so that `$VALUE` can directly rewrite as the same right hand side elementary tree, thus can save one step of derivation. This way, we convert a CFG to a TSG, and at the same time convert the CFG parser training data to TSG parser training data, as sequences of tree instances. Each tree instance contains both tree ids and associated frontier kind and value ids.

During inference, the TSG parser sequentially predicts the sequence of tree indices - in the above example, two instances - plus an end symbol terminating the inference process. The tracker then uses these tree instances to construct the entire derivation tree, and its yield as the output program.

H Modeling Details of TSG Parser

In Figure 5, the encoder and decoder in the TSG parser are the same as in the CFG parser shown in Figure 4. The frontier encoder computes the frontier encoding vectors from the tree encoding from the decoder, and the embeddings retrieved from the tracker with frontier ids:

$$\mathbf{h}_{frontier} = \mathbf{h}_{tree} \odot \mathbf{tanh}(\mathbf{e}_{frontier})$$

where $\mathbf{h}_{tree}$ is the tree encoding vector, $\mathbf{e}_{frontier}$ is the frontier embedding vector, and $\odot$ is element-wise multiplication operator. Inside the decoder, we mix the tree embedding with max pooling of the embedings of the tree's frontiers, and use the mixed embedding to compute tree encoding vectors:

$$\hat{\mathbf{e}}_{\mathbf{tree}} = \mathbf{e}_{tree} + \max_{tree's frontiers} \{\mathbf{e}_{frontier}\}$$

where $\mathbf{e}_{tree}$ is the input tree embedding, $\hat{\mathbf{e}}_{\mathbf{tree}}$ is the output mixed embedding used to compute the tree encoding vector in the decoder.

Then the pointer networks match the frontier encoding vector against frontier value vectors obtained from three kinds of sources: 1) previous tree instances, whose encodings are stored in a memory; 2) constants, with id-based embedding vectors; 3) text spans of input, with encoding from projected encoder output. Each of these corresponds to a "Pointer Networks" box in Figure 5:

- *Span pointer* matches the frontier encoding vector from decoder output against spans in the input, and is similar to the pointer in the CFG parser, except that we use a multiplicative attention layer and use a bias term:

  $$logit_{b,e} = [\mathbf{h}^b_{enc}; \mathbf{h}^e_{enc}]^\intercal \mathbf{W}_{span} \mathbf{h}_{frontier} + b_{span}$$

  where matrix $\mathbf{W}_{span}$ and scalar b_{span} are trainable parameters.

- *Constant pointer* matches the frontier encoding against precomputed constant embeddings:

  $$logit_c = \mathbf{e}_c^\intercal \mathbf{W}_{const} \mathbf{h}_{frontier} + b_{const}$$

  where c is the constant index, matrix W_{const} and scalar b_{const} are trainable parameters.

- *Past tree pointer* matches the frontier encoding against past tree encodings stored in a memory:

  $$logit_t = \mathbf{h}_t^\intercal \mathbf{W}_{tree} \mathbf{h}_{frontier} + b_{tree}$$

  where t is the time step, matrix $\mathbf{W}_{tree}$ and scalar b_{tree} are trainable parameters.

The tracker performs search of the derivation hypothesis with highest combined tree and frontier probabilities:

$$\pi^* = \operatorname*{argmax}_{\pi} \prod_{t \in T(\pi)} p(t) \prod_{f \in F(t)} \max_{v} p(v|f)$$

where π is a derivation hypothesis; $T(\pi)$ is the set of right-hand-side trees in π's rules; $F(t)$is the frontier set of tree t; v is a value of frontier f.

In addition to ensuring validity of derivation, the matcher can perform other checks to alleviate the problems caused by parallel frontier prediction. As mentioned earlier, checking overlapped text copying does provide small improvement in beam search. Should there be a typing system, the tracker could also apply type check to make sure each frontier has appropriate value by penalizing type violations.

I Accuracy Variations of Baseline Parser

We find the baseline parser's accuracy varies greatly with different random seeds. This phenomenon is also observed by Csordás et al. (2021). To study the degree of variation, we run training 55 times with different random seeds on all the splits. Table 8 and Figure 9 summarize the result.

As we can see, the accuracy variations among different runs are very large for those TMCD splits, ranging from 6.4 to 11.3, probably because the model is sensitive to randomness in initialization. This also shows the importance of using large test splits, and reporting results from multiple runs. From the plot we also notice quite a few outlier points that are far from clusters, therefore we prefer median over mean when reporting results.

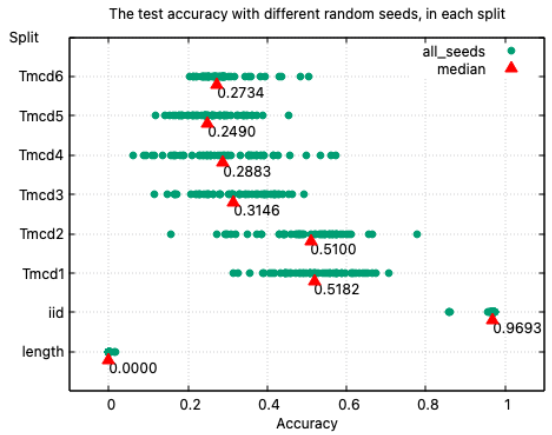


Figure 9: Baseline accuracies from 55 training runs on all splits.

	median%	**mean%**	**stddev%**
IID	96.9	96.2	2.6
Tmcd1	51.8	52.0	9.0
Tmcd2	51.0	49.0	11.3
Tmcd3	31.5	31.7	9.2
Tmcd4	28.8	29.3	11.9
Tmcd5	24.9	26.0	6.9
Tmcd6	27.3	28.8	6.4
Length	0	0.1	0.3

Table 8: Baseline accuracy median, mean, standard deviation on all splits.

ACL 2023 Responsible NLP Checklist

A For every submission:

☐ A1. Did you describe the limitations of your work?
Left blank.

☐ A2. Did you discuss any potential risks of your work?
Left blank.

☐ A3. Do the abstract and introduction summarize the paper's main claims?
Left blank.

☐ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☐ Did you use or create scientific artifacts?

Left blank.

☐ B1. Did you cite the creators of artifacts you used?
Left blank.

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Left blank.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Left blank.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Left blank.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Left blank.

☐ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Left blank.

C ☐ Did you run computational experiments?

Left blank.

☐ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Left blank.

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☐ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Left blank.

☐ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Left blank.

☐ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Left blank.

D ☐ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
Left blank.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
Left blank.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
Left blank.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
Left blank.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
Left blank.