Measuring and Improving Compositional Generalization in Text-to-SQL via Component Alignment

Anonymous ACL submission

Abstract

Recently, the challenge of *compositional generalization* in NLP has attracted more and more attention. Specifically, many prior works show that neural networks struggle with compositional generalization where training and test distributions differ. However, most of these works are based on word-level synthetic data or a specific data split method to generate compositional biases. In this work, we propose a clause-level compositional example generation method, and we focus on text-to-SQL tasks. We first split the sentences in the Spider text-to-SQL dataset into several sub-sentences, then we annotate each sub-sentence with its corresponding SQL clause, resulting in our new dataset Spider-SS. Building upon Spider-SS, we further construct a new dataset named Spider-CG, by composing Spider-SS sub-sentences to test the ability of models to generalize compositionally. Experiments show that previous models suffer significant performance degradation when evaluated on Spider-CG, even though every sub-sentence is seen during training. To deal with this problem, we modify the RATSQL+GAP model to fit the segmented data of Spider-SS, and we show that this method improves the generalization performance.[1]

1 Introduction

Neural models in supervised learning settings show good performance on data drawn from the training distribution. However, the generalization performance can be poor on out-of-distribution (OOD) samples (Finegan-Dollak et al., 2018; Suhr et al., 2020; Kaushik et al., 2020; Sagawa et al., 2020). This observation might hold even when the new samples are composed of known constituents; e.g., on SCAN dataset (Lake and Baroni, 2018), many models provide the wrong predictions for the input "jump twice and walk", even when "jump twice", "walk", and "walk twice" are seen during training. This (often lacking) ability to generalize to novel combinations of elements observed during training is referred to as *compositional generalization*.

Previous work on compositional generalization in text-to-SQL focuses on data split and word substitution (Finegan-Dollak et al., 2018; Shaw et al., 2021). However, data split methods are limited by the dataset content, making it difficult to construct a challenging benchmark while ensuring that every compound appears in the training set. Ensuring a reasonable data split may also lead to a reduction in the size of the dataset, e.g., the training set drop from 7000 to 3282 in the Spider TCMD split (Yu et al., 2018b; Shaw et al., 2021).

Word substitution is often used to generate compositional variations for data augmentation (Yu et al., 2021; Andreas, 2020). However, this method cannot generate more complex sentences, i.e., cannot generate "jump twice and walk" from the data only containing "jump twice", "walk", and "walk twice". In addition, in the *cross-domain* text-to-SQL setting, examples generated by word substitution are supposed to be considered as in-distribution data, since cross-domain models are expected to generalize to unseen domains with unseen utterances, including examples generated by word substitution.

In this work, we first introduce our Spider-SS dataset (SS stands for sub-sentence) derived from the Spider benchmark (Yu et al., 2018b). Figure 1 presents a comparison between Spider and Spider-SS. To build Spider-SS, we design a sentence split algorithm to split every Spider sentence into several sub-sentences until indivisible. Next, we annotate every sub-sentence with its corresponding SQL clause. In order to reduce the difficulty of annotation, we annotate the query in an intermediate representation called NatSQL (Gan et al., 2021b), because it is simpler and syntactically aligns better with natural language (NL).

[1] We will release code and dataset upon publication.

Spider Example:

Sentence: What type of pet is the youngest animal, and how much does it weigh?

SQL: SELECT PetType , Weight FROM Pets ORDER BY Pet_Age LIMIT 1

Spider-SS Example:

SubSentence: What type of pet
NatSQL: SELECT Pets.Pettype

SubSentence: is the youngest animal
NatSQL: ORDER BY Pets.Pet_Age LIMIT 1

SubSentence: , and how much does it weigh?
NatSQL: SELECT Pets.Weight

Figure 1: A natural language sentence in the original Spider benchmark is split into three sub-sentences in Spider-SS, where each sub-sentence has a corresponding NatSQL clause.

Spider-SS :

Example-1:

SubSentence: What is the name and nation of the singer
NatSQL: SELECT Singer.Name SELECT Singer.Country

SubSentence: who have a song having 'Hey' in its name?
NatSQL: WHERE Concert.Song_Name like '%Hey%'

Example-2:

SubSentence: What are the names of the singers
NatSQL: SELECT Singer.Name

SubSentence: who performed in a concert in 2014?
NatSQL: WHERE Concert.Year = 2014

Spider-CG :

Subset-1: sub-sentence substitution in Example 1 and 2

Sentence: What is the name and nation of the singer who performed in a concert in 2014?
NatSQL: SELECT Singer.Name, Singer.Country WHERE Concert.Year = 2014

Subset-2: Example-1 append a sub-sentence from Example-2

Sentence: What is the name and nation of the singer who have a song having 'Hey' in its name and who performed in a concert in 2014?
NatSQL: SELECT Singer.Name, Singer.Country WHERE Concert.Song_Name like '%Hey%' AND Concert.Year = 2014

Figure 2: Two Spider-CG samples generated by: (1) substituting the sub-sentence with one from another example; or (2) composing sub-sentences from 2 examples in Spider-SS.

Our annotated Spider-SS provides us with sub-sentences paired with NatSQL clauses, which serve as our compounds. Based on Spider-SS, we further construct the dataset Spider-CG (CG stands for compositional generalization), by substituting the sub-sentences with those from other samples, or composing two sub-sentences to form a more complicated sample. Spider-CG contains two subsets, and we present one example for each subset in Figure 2. The first subset includes 24,134 examples generated by substituting sub-sentences, and we consider most data in this subset as in-distribution. The second subset contains 22,531 examples generated by appending sub-sentences, which increases the length and complexity of the sentence and the SQL query compared to the original samples, and we consider most examples in this subset as OOD. We demonstrate that when models are only trained on the original Spider dataset, they suffer a significant performance drop on the second OOD subset of Spider-CG, even though the domain appears in the training set.

To improve the generalization performance of text-to-SQL models, we modify the RAT-SQL+GAP+NatSQL (Wang et al., 2020; Shi et al., 2021; Gan et al., 2021b) model so that it can be applied to the Spider-SS dataset, with the model trained sub-sentence by sub-sentence. This modification obtains more than 7.8% accuracy improvement evaluated on the second subset of Spider-CG. To our knowledge, this is the first sub-sentence-based text-to-SQL model.

In short, we make the following contributions:

- Besides the sentence split algorithm, we introduce Spider-SS, a human-curated sub-sentence-based text-to-SQL dataset built upon the Spider benchmark, by splitting its NL questions into sub-sentences.
- We construct the Spider-CG benchmark for measuring the compositional generalization performance of text-to-SQL models.
- We show that the RATSQL+GAP+NatSQL model can be adapted to sub-sentence-based text-to-SQL data, and that this improves its generalization performance.

2 Spider-SS

2.1 Overview

Figure 1 presents a comparison between Spider and Spider-SS. Unlike Spider, which annotates a whole SQL query to an entire sentence, Spider-SS annotates the SQL clauses to sub-sentences. Spider-SS uses NatSQL (Gan et al., 2021b) instead of SQL because some examples are difficult to annotate using SQL. The Spider-SS provides a combination algorithm that collects all NatSQL clauses and then generates the NatSQL query, where the NatSQL query can be converted into an SQL query.

The purpose of building Spider-SS is to attain clause-level text-to-SQL data to generate more

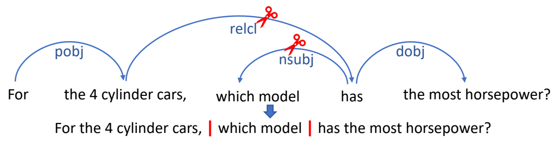

Figure 3: Dependency structure of a sentence and how to split this sentence into three sub-sentences.

complex examples through the combination of clauses. Besides, it is easier to build models with better performance based on more finely labeled data, especially in areas with relatively few resources, i.e., there are only 7000 training samples in the Spider dataset. Consistent with Spider, Spider-SS contains 7000 training and 1034 development examples, but Spider-SS does not contain a test set since the Spider test set is not public. There are two steps to build Spider-SS. First, design a sentence split algorithm to cut the sentence into sub-sentences, and then manually annotate the NatSQL clause corresponding to each sub-sentence.

2.2 Sentence Split Algorithm

We build our sentence split algorithm upon the NL dependency parser spaCy [2], which provides the grammatical structure of a sentence. Basically, we split the sentence with the following dependencies: *prep, relcl, advcl, acl, nsubj, npadvmod, csubj, nsubjpass* and *conj*. According to (de Marnee and Manning, 2016), these dependencies help us separate the main clause, subordinate clauses, and modifiers. Figure 3 shows the dependency structure of a sentence and how to split this sentence into three sub-sentences. However, not every sentence would be split since there are some non-splittable sentences, such as the third example in Figure 4, with the same annotation as the Spider dataset. Although this method can separate sentences well in most cases, due to the variability of natural language, some examples cannot be perfectly split.

To address the remaining issues in sentence split, we design some refinement steps tailored to text-to-SQL applications. For example, when the phase of a schema column or table is accidentally divided into two sub-sentences, these two sub-sentences are automatically concatenated. Besides, when there is only one word in a sub-sentence, the corresponding split should also be undone.

We sampled 500 examples from the Spider-SS development set to evaluate the acceptability

[2]https://github.com/explosion/spaCy

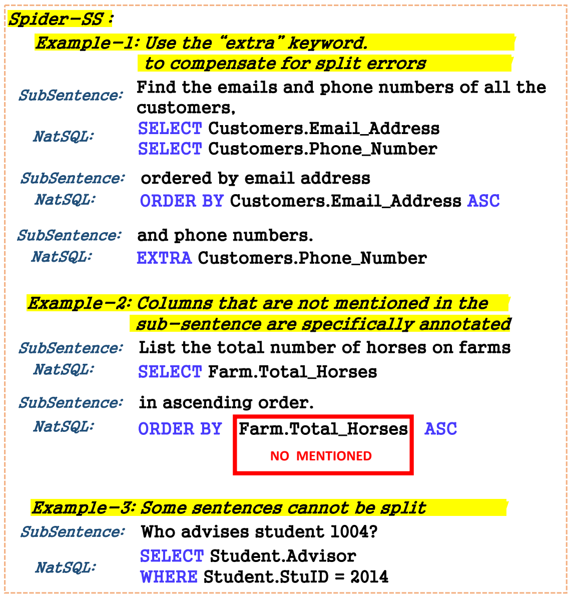

Figure 4: Spider-SS examples in three special cases.

of splitting results manually, and only $< 3\%$ of the splitting results are unsatisfactory. For example, in the splitting results of the first example in Figure 4, the last two sub-sentence should be combined to correspond to "**ORDER BY** Customer.Email_Address, Customer.Phone_Number **ASC** ". In this example, we did not simply give an "**ORDER BY** Customer.Phone_Number **ASC** " to the last sub-sentence, because it does not mention anything related to "**ORDER BY** ". Here, we introduce "*extra*", a new NatSQL keyword designed for the Spider-SS dataset, indicating that this sub-sentence mentions a column that temporarily does not fit in any other NatSQL clauses. When combining NatSQL clauses into the final NatSQL query, the combining algorithm determines the final position for the "*extra*" column based on the clauses before and after. Note that even if there is a small proportion of unsatisfactory splitting results, as long as the model trained on Spider-SS can give the correct output according to the input sub-sentence, the quality of the sub-sentences itself does not strongly affect the model utility.

2.3 Data Annotation

When we get the split results from the last step, we can start data annotation. We give precise annotations based on the sub-sentence content, such as the "*extra*" column annotation discussed in the last subsection. Besides, if the description of the schema column is missing in the sub-sentence, we will give the schema column an additional "*NO*

MENTIONED" mark. For example, in the second example of Figure 4, the "*in ascending order*" sub-sentence does not mention the "*Farm.Total_Horses*" column. Therefore, we add a "*NO MENTIONED*" mark for it. For those sub-sentences that do not mention anything related to the query, we give a "*NONE* mark, representing there are no NatSQL clauses.

Since the annotation is carried out according to the sub-sentence content, the equivalent SQL that is more consistent with the sub-sentence will be preferred to the original SQL. Similarly, if the original SQL annotation is wrong, we correct it according to the content.

We annotate the sub-sentence using NatSQL instead of SQL, where NatSQL is an intermediate representation of SQL, only keeping the *SELECT, WHERE, and ORDER BY* clauses from SQL. Since some sub-sentences need to be annotated with *GROUP BY* clause, we choose the version of NatSQL augmented with *GROUP BY*. We did not use SQL directly because it is difficult to annotate in some cases, such as the SQL in Figure 5. The difficulty is that there are two *SELECT* clauses in this SQL query, but none of the sub-sentences seem to correspond to two *SELECT* clauses. In addition, considering that the two *WHERE* conditions correspond to different *SELECT* clauses, the annotation work based on SQL is far more difficult to complete. As shown in Figure 5, we can use NatSQL to complete the annotation quickly, while the NatSQL can be converted back to the target SQL.

3 Spider-CG

3.1 Overview

Spider-CG is a synthetic dataset, which is generated by recombining the sub-sentences of Spider-SS. There are two recombination methods. The first is sub-sentence substitution between different examples, and the second is to append a sub-sentence into another sentence. To facilitate the follow-up discussion, we named the Spider-CG subset generated by the sub-sentence substitution method **CG-SUB**, and the other named **CG-APP**.

In CG-SUB, there are 21,168 examples generated from the Spider-SS training set, while 2,966 examples are generated from the development set. In CG-APP, examples generated from training and development sets are 19,241 and 3,290, respectively. Therefore, the whole Spider-CG contains 46,665 examples, which is about six times the Spider dataset. If need more data, we can append sub-sentences to the CG-SUB examples.

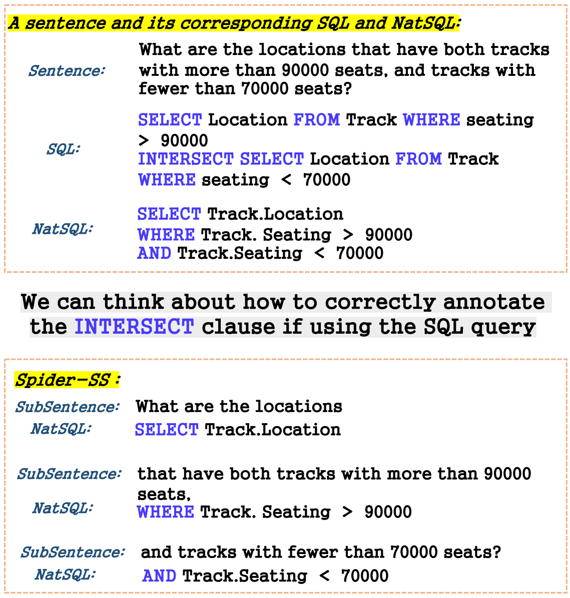

Figure 5: It is difficult to annotate if using the SQL instead of NatSQL.

3.2 Generation Algorithm

Before generating the Spider-CG, we need to generate the compositional element. Considering that there are some unsatisfactory splitting results in spider-SS, in order to ensure the quality of generated sentences, compositional elements abandoned the examples with "*NO MENTIONED*" and "*extra*" column, such as the first and second examples in Figure 4. Each element contains one or more sub-sentences, but these sub-sentences must be annotated as *WHERE or ORDER BY* clauses. We collect the sub-sentences for compositional elements by scanning all sub-sentence from top to bottom or from bottom to top and stopping when encountering clauses except *WHERE and ORDER BY*. For example, we generate a compositional element containing the last two sub-sentences of the Spider-SS example in Figure 5 In contrast, we do not generate any element from the example in Figure 1.

According to Algorithm 1, we can generate the CG-SUB and CG-APP based on compositional elements. It should be noted that elements in a domain cannot be used in another because the schema items are different. So as many domains as there are, it needs to run Algorithm 1 how many times.

Algorithm 1 Generate CG-SUB and CG-APP dataset in a certain domain

Input: e_list ▷ All compositional elements in a domain
Output: cg_sub and cg_app ▷ CG-SUB and CG-APP dataset in a certain domain

for Every $element_1$ in e_list **do**
  for Every $element_2$ in e_list **do**
    if $element_1$!= $element_2$ **then**
      if $element_1$.can_be_substituted_by($element_2$) **then**
        cg_sub.append($element_1$.generate_substitution_example($element_2$))
      if $element_1$.can_append($element_2$) **then**
        cg_app.append($element_1$.generate_appending_example($element_2$))
return cg_sub, cg_app

Ques	List the name and age of the heads of departments are older than 56 ?
SQL	**SELECT** name, age **FROM** head **WHERE** age > 56
Ques	Show the name of employees named Mark Young ?
SQL	**SELECT** name **FROM** employee **WHERE** name = 'Mark Young'

Table 1: Two acceptable but not perfect examples in the Spider-CG.

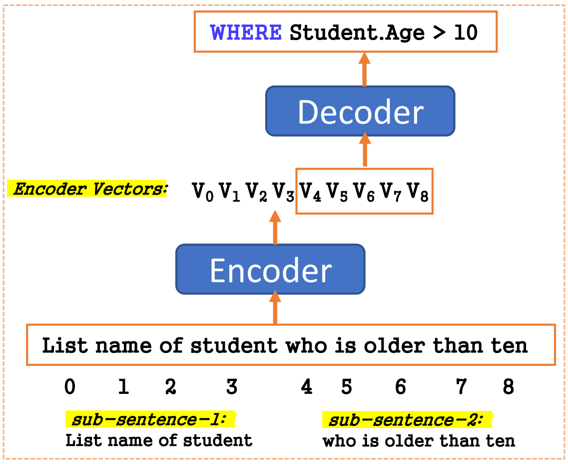

Figure 6: A example of encoding the whole sentence but decoding only the sub-sentence.

3.3 Quality Assurance

We randomly sampled 2000 examples from the Spider-CG dataset, around 99% of which are acceptable. Acceptable does not mean that there are no grammatical errors; it means that the sentence has been clearly expressed and does not cause ambiguity. Besides, acceptable examples may be meaningless. We give two acceptable but not perfect examples in Table 1, where the first sentence contains the grammatical error. At the same time, the other is meaningless because the content it wants to query is the condition it gave. We define an acceptable example as an example where you may not find a problem without studying it carefully.

The '*can_be_substituted_by*' and '*can_append*' function in Algorithm 1 prevent mistakes while generating the Spider-CG. These two methods check whether the sub-sentences can be appropriately connected, e.g., whether the previous word is the same and whether the part of speech of the first word in the sub-sentence is the same. Besides, these two methods also prevent generating overly complex or contradictory SQL, e.g., 'give me the name of the youngest student who is the oldest'.

4 Model

Existing text-to-SQL models input a sentence and output the corresponding SQL query. So the easiest way to think of using the Spider-SS dataset is to train the model where inputting sub-sentence and outputting the corresponding NatSQL clauses. However, this method is not workable because it will lose some essential schema information. For example, if you only look at the third sub-sentence in Figure 1, you do not know whether he ask about the weight of pets or people.

In order to take into account the context and the sub-sentence data of Spider-SS, we propose that a seq2seq model can encode the whole sentence but decode only the sub-sentence. Figure 6 presents the workflow of encoding the whole sentence but only decoding the sub-sentence of '*who is older than ten*' and outputting the corresponding NatSQL clause. Based on this modification, a seq2seq text-to-SQL model can be adapted to the Spider-SS.

We modify the RATSQL+GAP+NatSQL (Wang

et al., 2020; Shi et al., 2021; Gan et al., 2021b) model to adapt the Spider-SS. We choose this model because its performance is high enough, and it has the best performance among the text-to-SQL models that support NatSQL. We keep the same hyperparameters. Although re-search hyperparameters may improve the performance, it requires considerable computing resources (Wang et al., 2020).

5 Experiment

5.1 Experimental Setup

We evaluate the previous state-of-the-art models on the Spider-CG and Spider (Yu et al., 2018b) datasets. Since the Spider test set is not publicly accessible, Spider-CG does not contain a test set. As discussed in Section 3.1, we divide the Spider-CG into two subsets: CG-SUB and CG-APP. Therefore, there are five evaluation sets:

- **Spider$_{\mathrm{D}}$**: the original Spider development set with 1,034 examples for *cross-domain in-distribution* text-to-SQL evaluation.
- **CG-SUB$_{\mathrm{T}}$**: the CG-SUB training set, containing 21,168 examples generated from Spider-SS training set by substituting sub-sentence method. CG-SUB$_{\mathrm{T}}$ is used for *in-domain in-distribution* text-to-SQL evaluation.
- **CG-SUB$_{\mathrm{D}}$**: the CG-SUB development set containing 2,966 examples for *cross-domain in-distribution* text-to-SQL evaluation.
- **CG-APP$_{\mathrm{T}}$**: the CG-APP training set, containing 19,241 examples generated from Spider-SS training set by appending sub-sentence method. CG-APP$_{\mathrm{T}}$ is used for *in-domain out-of-distribution* text-to-SQL evaluation.
- **CG-APP$_{\mathrm{D}}$**: the CG-APP development set containing 3,290 examples for *cross-domain out-of-distribution* text-to-SQL evaluation.

Our evaluation is based on the exact match and execution match metric defined in the original Spider benchmark. The exact match metric measures whether the syntax tree of the predicted query without condition values is the same as that of the gold query. The execution match metric measures whether the query results from the predicted query are the same as the gold query results. All models are only trained on 7000 Spider or Spider-SS training examples. We evaluate the following open-source models that reach competitive performance on Spider:

- **RATSQL**: The RATSQL+BERT model trained on Spider (Wang et al., 2020; Devlin et al., 2019).
- **RATSQL$_{\mathrm{G}}$**: The RATSQL+GAP model trained on Spider (Shi et al., 2021).
- **RATSQL$_{\mathrm{GN}}$**: The RATSQL+GAP+NatSQL model trained on NatSQL (Gan et al., 2021b).
- **RATSQL$_{\mathrm{GNS}}$**: The modified RATSQL+GAP+NatSQL model trained on Spider-SS, as discussed in Section 4.

Dataset	Exact Match	Execution Match
Training Set	89.4%	94.0%
Development Set	90.0%	94.3%

Table 2: Use exact match and execution match metrics to evaluate the difference between the SQL in Spider and the SQL generated by NatSQL in Spider-SS.

Dataset	easy	medium	hard	extra
Spider$_{\mathrm{D}}$	24.1%	43.1%	16.8%	16.1%
CG-SUB$_{\mathrm{T}}$	28.3%	38.4%	20.8%	12.5%
CG-SUB$_{\mathrm{D}}$	33.8%	37.4%	13.6%	12.6%
CG-APP$_{\mathrm{T}}$	3.2%	30.3%	27.3%	39.1%
CG-APP$_{\mathrm{D}}$	2.3%	41.9%	22.9%	32.8%

Table 3: The difficulty distribution of five different evaluation sets.

5.2 Dataset Analysis

Spider-SS. Table 2 presents the difference between the SQL in Spider and the SQL generated by NatSQL in Spider-SS. Our evaluation results are lower than the original NatSQL dataset (Gan et al., 2021b) because the Spider-SS uses equivalent SQL and corrects some errors, as discussed in Section 2.3. Most equivalent and corrected SQL cannot get positive results in exact match metric, while a small part can not either in execution match. Therefore, the model trained on Spider-SS may not be ideal for chasing the Spider benchmark, especially based on the exact match metric. Similarly, the RATSQL$_{\mathrm{G}}$ extending NatSQL had achieved a previous SOTA result in the execution match of the Spider test set but get a worse result than the original in the exact match (Gan et al., 2021b). Thus, we recommend using NatSQL-based datasets to evaluate models trained on NatSQL.

Spider-CG. Table 3 presents the difficulty distribution of five different evaluation sets. The difficulty criteria are defined by Spider benchmark, including *easy, medium, hard* and *extra hard*. Experiments show that the more difficult the SQL is, the more difficult it is to predict correctly (Wang et al., 2020; Shi et al., 2021; Gan et al., 2021b).

It can be found from Table 3 that the difficulty distribution of CG-SUB$_\text{T}$ and CG-SUB$_\text{D}$ is similar to that of Spider$_\text{D}$. The similar distributions among CG-SUB$_\text{T}$, CG-SUB$_\text{D}$, and Spider$_\text{D}$ support the view discussed in Section 1 that the examples generated by the substitution method are in-distribution.

On the other hand, the difficulty distributions of CG-APP$_\text{T}$ and CG-APP$_\text{D}$ are obviously different from that of Spider$_\text{D}$. Due to appending the sub-sentence, the NL and SQL in CG-APP become more complex, where the proportion of SQL in *extra hard* increased significantly, while *easy* was the opposite.

Dataset	Deviation <= 1	Deviation <= 2
CG-SUB$_\text{T}$	93.2%	94.4%
CG-SUB$_\text{D}$	92.9%	94.1%
CG-APP$_\text{T}$	86.0%	90.4%
CG-APP$_\text{D}$	88.9%	92.6%

Table 4: The similarity between sub-sentences in Spider-SS and Spider-CG generated by the same split algorithm under the deviation of one or two words.

Approach	Spider$_\text{D}$	CG-SUB$_\text{T}$	CG-SUB$_\text{D}$	CG-APP$_\text{T}$	CG-APP$_\text{D}$
RATSQL	72.0%	79.5%	72.0%	45.1%	47.2%
RATSQL$_\text{G}$	72.7%	80.9%	70.3%	45.2%	44.2%
RATSQL$_\text{GN}$	73.9%	90.2%	75.0%	67.8%	60.5%
RATSQL$_\text{GNS}$	**74.5%**	**91.4%**	**76.7%**	**82.5%**	**68.3%**

Table 5: Exact match accuracy on evaluation sets.

5.3 Sentence Split Algorithm Evaluation

We generate the Spider-CG based on the combination of Spider-SS sub-sentences split by the algorithm introduced in Section 2.2. We can reuse this algorithm to split the sentence in Spider-CG and then compare the splitting results with the Spider-SS sub-sentences to evaluate the stability of the splitting algorithm. We consider that a deviation of one or two words in the splitting result is acceptable. For example, in Figure 1, we consider that putting the comma of the third sub-sentence into the second sub-sentence does not change the meaning of sub-sentences, same for moving both the comma and the word 'and'.

Table 4 presents the similarity between sub-sentences in Spider-SS and Spider-CG, which are generated by the same split algorithm under the deviation of one or two words. The similarity exceeds 90% in all evaluation set when two deviation words are allowed. Considering that the model trained on the Spider-SS does not require consistent split results, as discussed in Section 2.2, the similarity results of the splitting algorithm are good enough. The similarity of CG-SUB is higher than that of CG-APP, which means the more complex the sentence, the greater the challenge to the algorithm. Although the algorithm has been refined on the training set, the similarity between training and development in CG-SUB and CG-APP is close, showing that the algorithm performs consistently for unseen datasets. In summary, we consider that as long as the sentences are not more complex than CG-APP, the algorithm can be used stably in other text-to-SQL datasets.

5.4 Model Results

Table 5 presents the exact match accuracy on the five different evaluation sets. Specifically, the RATSQL$_\text{GNS}$ consistently outperforms other models. We use the sentence split algorithm to split every sentence before inputting the RATSQL$_\text{GNS}$. Although there are some un-similar splitting results, it did not prevent the RATSQL$_\text{GNS}$ from getting good performance. RATSQL$_\text{GNS}$ and RATSQL$_\text{GN}$ have close results in the in-distribution dataset, i.e., Spider$_\text{D}$, CG-SUB$_\text{T}$, and CG-SUB$_\text{D}$, but the RATSQL$_\text{GNS}$ significantly improves over RATSQL$_\text{GN}$ on the OOD dataset, i.e., CG-APP$_\text{T}$ and CG-APP$_\text{D}$. The results demonstrate that the sub-sentence-based method can improve the generalization performance.

The evaluation results of all models in Spider$_\text{D}$ and CG-SUB$_\text{D}$ are close, which further confirm that the sub-sentence substitution method generates in-distribution data in a cross-domain text-to-SQL setting. In the two OOD datasets, CG-APP$_\text{T}$ and CG-APP$_\text{D}$, the performance of all models has dropped by about 10% to 30%.

Although the performance of all models on Spider$_\text{D}$ is close, the performance of RATSQL$_\text{GN}$ and RATSQL$_\text{GNS}$ is significantly better in the rest four datasets. One of the reasons is that RATSQL and RATSQL$_\text{G}$ are trained on SQL while RATSQL$_\text{GN}$ and RATSQL$_\text{GNS}$ use NatSQL for training. As discussed in Section 5.2, since the training data of Spider and Spider-SS are about 10% different, this leads to the performance degradation in the model trained on Spider when evaluated on the SQL generated by the NatSQL of Spider-SS, and vice versa. On the other hand, ex-

Approach	Spider_{D}	CG-SUB_{T}	CG-SUB_{D}	CG-APP_{T}	CG-APP_{D}
$\text{RATSQL}_{\text{GN}}$	75.8%	86.7%	78.0%	70.4 %	68.9%
$\text{RATSQL}_{\text{GNS}}$	**76.7%**	**88.3%**	**80.4%**	**78.8%**	**75.1%**

Table 6: Execution match accuracy on evaluation sets.

periments in (Gan et al., 2021b) show that NatSQL improve the model performance in *extra hard* SQL. Therefore, $\text{RATSQL}_{\text{GN}}$ and $\text{RATSQL}_{\text{GNS}}$ suffer less performance degradation in CG-APP_{T} and CG-APP_{D} than RATSQL_{G} and RATSQL.

Since RATSQL and RATSQL_{G} do not support generating executable SQL, we compare the execution results between $\text{RATSQL}_{\text{GN}}$ and $\text{RATSQL}_{\text{GNS}}$ in Table 6. $\text{RATSQL}_{\text{GNS}}$ again consistently outperform the $\text{RATSQL}_{\text{GN}}$.

6 Related Work

Text-to-SQL translation. To achieve text-to-SQL translation, researchers have built various benchmarks (Iyer et al., 2017; Ana-Maria Popescu et al., 2003; Tang and Mooney, 2000; Giordani and Moschitti, 2012; Li and Jagadish, 2014; Yaghmazadeh et al., 2017; Zhong et al., 2017; Yu et al., 2018b). In particular, most recent works focus on improving the performance on Spider benchmark (Yu et al., 2018b), where models are required to generate complex SQL in cross-domain setting. Among various model architectures (Guo et al., 2019; Zhang et al., 2019; Wang et al., 2020), most of them have implemented the pre-training method, including the latest state-of-the-art model (Scholak et al., 2021; Cao et al., 2021), where Yu et al. (2021) and Rubin and Berant (2021) augment the text-to-SQL data for pre-training.

Data augmentation for text-to-SQL models. Data augmentation has been commonly used for improving performance (Xiong and Sun, 2019; Li et al., 2019). In the context of text-to-SQL generation, Yu et al. (2018a) generate synthetic training samples from some pre-defined SQL and NL question templates. Parikh et al. (2020) introduces an table-to-text dataset with over 120,000 examples that proposes a controlled generation task: given a Wikipedia table and a set of highlighted table cells, produce a one-sentence description. Yu et al. (2021) sample from the given examples and then give a large number of tables to generate new synthetic examples. Shi et al. (2021) present a model pre-training framework that jointly learns representations of NL utterances and table schemas by leveraging generation models to generate pre-train data.

Compositional generalization for semantic parsing. Compositional generalization for semantic parsing has captured wide attention recently (Finegan-Dollak et al., 2018; Oren et al., 2020; Furrer et al., 2020; Conklin et al., 2021). Most prior works on text-to-SQL tasks focus on the cross-domain generalization, which mainly assess how the models generalize the domain knowledge to new database schemas (Suhr et al., 2020; Gan et al., 2021a). On the other hand, Shaw et al. (2021) introduces TMCD splits for studying compositional generalization in semantic parsing, where they aim to maximize the divergence of SQL compounds between the training and test sets. However, this method does not support the benchmark construction for cross-domain out-of-distribution composition generalization evaluation.

Our model is inspired by prior works on neural parsers constructed to capture granular information from a whole. Yin et al. (2021) describe a span-level supervised attention loss that improves compositional generalization in semantic parsers. Herzig and Berant (2021) propose SpanBasedSP, a parser that predicts a span tree over an input utterance, and dramatically improves performance on splits that require compositional generalization. Chen et al. (2020) propose the Neural-Symbolic Stack machine (NeSS), which integrates a symbolic stack machine into a seq2seq generation framework, and learns a neural network as the controller to operate the machine.

7 Conclusion

We introduce Spider-SS and Spider-CG for measuring compositional generalization of text-to-SQL models. Specifically, Spider-SS is a human-curated sub-sentence-based text-to-SQL dataset built upon the Spider benchmark. Spider-CG is a synthetic text-to-SQL dataset constructed by substituting and appending sub-sentences of different samples, so that the training and test sets consist of different compositions of sub-sentences. We found that the performance of previous text-to-SQL models drop dramatically on the Spider-CG OOD examples, while modifying the RAT-SQL+GAP+NatSQL model to fit the segmented data of Spider-SS improves compositional generalization performance.

References

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a Theory of Natural Language Interfaces to Databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, pages 149–157.

Jacob Andreas. 2020. Good-enough compositional data augmentation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7556–7566, Online. Association for Computational Linguistics.

Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. LGESQL: Line graph enhanced text-to-SQL model with mixed local and non-local relations. 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 2541–2555, Online. 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.

Henry Conklin, Bailin Wang, Kenny Smith, and Ivan Titov. 2021. Meta-learning to compositionally generalize. 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 3322–3335, Online. Association for Computational Linguistics.

Marie-Catherine de Marnee and Christopher D. Manning. 2016. Stanford typed dependencies manual.

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.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. pages 351–360.

Daniel Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärli. 2020. Compositional generalization in semantic parsing: Pre-training vs. specialized architectures. *CoRR*, abs/2007.08970.

Yujian Gan, Xinyun Chen, and Matthew Purver. 2021a. Exploring underexplored limitations of cross-domain text-to-sql generalization. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R. Woodward, John Drake, and Qiaofu Zhang. 2021b. Natural sql: Making sql easier to infer from natural language specifications.

Alessandra Giordani and Alessandro Moschitti. 2012. Automatic Generation and Reranking of SQL-derived Answers to NL Questions. In *Proceedings of the Second International Conference on Trustworthy Eternal Systems via Evolving Software, Data and Knowledge*, pages 59–76.

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.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a Neural Semantic Parser from User Feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973.

Divyansh Kaushik, Eduard Hovy, and Zachary Lipton. 2020. Learning the difference that makes a difference with counterfactually-augmented data. In *International Conference on Learning Representations*.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2873–2882. PMLR.

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

Jingjing Li, Wenlu Wang, Wei Shinn Ku, Yingtao Tian, and Haixun Wang. 2019. SpatialNLI: A spatial domain natural language interface to databases using spatial comprehension. In *GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, pages 339–348, New York, NY, USA. Association for Computing Machinery.

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.

Ankur Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020. ToTTo: A controlled table-to-text generation dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1173–1186, Online. Association for Computational Linguistics.

Ohad Rubin and Jonathan Berant. 2021. SmBoP: Semi-autoregressive bottom-up semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 311–324, Online. Association for Computational Linguistics.

Shiori Sagawa, Pang Wei Koh, Tatsunori B. Hashimoto, and Percy Liang. 2020. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. Picard: Parsing incrementally for constrained auto-regressive decoding from language models.

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.

Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. 2021. Learning contextual representations for semantic parsing with generation-augmented pre-training. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(15):13806–13814.

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

Lappoon R Tang and Raymond J Mooney. 2000. Automated Construction of Database Interfaces: Intergrating Statistical and Relational Learning for Semantic Parsing. In *2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 133–141.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Hongvu Xiong and Ruixiao Sun. 2019. Transferable Natural Language Interface to Structured Queries Aided by Adversarial Generation. In *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*, pages 255–262. IEEE.

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. SQLizer: Query Synthesis from Natural Language. In *International Conference on Object-Oriented Programming, Systems, Languages, and Applications, ACM*, pages 63:1—-63:26.

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.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. 2021. Grappa: Grammar-augmented pre-training for table semantic parsing.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018a. SyntaxSQLNet: Syntax tree networks for complex and cross-domain text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1653–1663, Brussels, Belgium. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018b. 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.

Rui Zhang, Tao Yu, Heyang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-based SQL query generation for cross-domain context-dependent questions. pages 5338–5349.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *CoRR*, abs/1709.0.