Exploring Dimensions of Generalizability and Few-shot Transfer for Text-to-SQL Semantic Parsing

Rajaswa Patil
TCS Research
Pune, India
patil.rajaswa@tcs.com

Manasi Patwardhan
TCS Research
Pune, India
manasi.patwardhan@tcs.com

Shirish Karande
TCS Research
Pune, India
shirish.karande@tcs.com

Lovekesh Vig
TCS Research
New Delhi, India
lovekesh.vig@tcs.com

Gautam Shroff
TCS Research
New Delhi, India
gautam.shroff@tcs.com

Abstract

Existing work on generalization in Text-to-SQL semantic parsing has been restricted to a zero-shot cross-domain setting. In this paper, we introduce **Spider-Gen**: a Text-to-SQL benchmark to develop a paradigm of transfer learning across distinct dimensions of generalization in Text-to-SQL semantic parsing. The Spider-Gen benchmark focuses on few-shot adaption for Cross-domain, Lexical, and Structural generalization of Text-to-SQL models. Through our experiments with the Spider-Gen dataset, we show that Seq2Seq language models struggle to generalize against change in data distribution, lexical changes in database schema, and changes in SQL query complexity. Our experiments also reveal that performing few-shot fine-tuning helps Text-to-SQL models to generalize across these changes. However, such few-shot adaptation comes with a negative effect on the knowledge learnt during training. Hence, we also explore Parameter-efficient Fine-tuning methods to overcome the limitations of Seq2Seq Text-to-SQL models. We release the Spider-Gen dataset publicly to facilitate further research in generalization and transfer learning across various dimensions in Text-to-SQL semantic parsing.

1 Introduction

The Text-to-SQL semantic parsing task involves translating a Natural Language query (NL) into its corresponding SQL query logical form for a given database. Recent development in Text-to-SQL has brought in significant advances in methods and their downstream performance on various versions of the task. Within a span of five years, the performance on Text-to-SQL leaderboards like Spider [Yu et al., 2018] has moved from an accuracy of mere 8% (2018) to 75% (2022).[1]. Under the current evaluation paradigm for Text-to-SQL, benchmarks like Spider and WikiSQL [Zhong et al., 2017] require: **(i)** *In-domain* pre-training of a model on a large number of databases; **(ii)** Evaluating the trained model against *Cross-domain* novel unseen databases from the same dataset distribution. While this evaluation paradigm shows an ever increasing performance on leaderboards, it poses several limitations.

Firstly, datasets following such an evaluation paradigms only focus on a single axis of generalization: cross-domain generalization. This involves a low-granular change in terms of introduction of entirely new databases. However, there are other important high-granular axes of generalization like changes

[1] https://yale-lily.github.io/spider

Workshop on Transfer Learning for Natural Language Processing, 2022, New Orleans.

in the database schema and changes in the nature of querying for existing databases. Under a realistic setting, generalizing to such changes is relatively more frequent and equally important as performing cross-domain generalization. The current datasets do not offer evaluation settings for generalization over such high-granular changes. Further, the *In-domain* training and *Cross-domain* test databases from such datasets belong to the same data distribution with respect to the data annotation and normalization protocols (for the databases and the SQL queries) followed during the creation of the datasets [Yu et al., 2018]. Hence, even with changes in the domain, the cross-domain test databases share a certain degree of similarity with the training databases - leading to a relatively higher *cross-domain* performance of benchmarked models. However as soon as the data distribution of the evaluation databases diverges from that of the training databases, the model performance decreases significantly [Lee et al., 2021].

Annotating samples of NL-SQL pairs is a resource heavy task, requiring supervision from database administrators and experts. Hence, most of the existing work in Text-to-SQL focuses on a zero-shot inference setting. While a zero-shot evaluation setting does show great improvements on Text-to-SQL leaderboards, recent work has shown that zero-shot inference fails for certain generalization dimensions like the ones discussed above [Suhr et al., 2020, Lee et al., 2021]. In order to overcome such limitations, one needs to follow a transfer learning approach with adaptation data from the target generalization. While annotating large amounts of new adaptation data for such a transfer learning approach might not always be feasible, recent work by Lee et al. [2021] has shown that even few-shot adaptation can show significant improvement in performance over zero-shot inference. However, it has not yet been explored how adapting large Text-to-SQL models under a few-shot settings affects their existing knowledge captured during training.

In this work, we aim to introduce a new paradigm of modeling and evaluation for Text-to-SQL, with multiple axes of generalization and a few-shot adaptation-based transfer learning setting to overcome these generalization issues. We do so by:

1. Introducing a new benchmark based on Spider and related datasets: **Spider-Gen** for few-shot transfer learning and evaluation of Text-to-SQL models with respect to multiple axes of generalization.[2]
2. Exploring the limitations of standard Seq2Seq language model architectures under zero-shot, as well as a few-shot evaluation settings for Text-to-SQL semantic parsing.
3. Exploring Parameter-efficient fine-tuning (PEFT) methods to overcome the limitations faced by the vanilla Seq2Seq architectures.

2 Spider-Gen Benchmark

In order to facilitate research towards the development of transfer learning based Text-to-SQL methods, we design a new evaluation paradigm for Text-to-SQL: **Spider-Gen**, which covers three dimensions of generalization as discussed further in Section 2.1. The Spider-Gen dataset differs from existing Text-to-SQL benchmark datasets by: **(i)** Providing distinct evaluation splits for different types of generalizations; **(ii)** Providing a set of few-shot adaptation samples for transfer learning over target generalizations.; **(iii)** Providing a set of held-out samples to inspect the negative effect of such transfer learning. Unlike the existing zero-shot cross-domain evaluation settings, under this evaluation paradigm, a given model is first fine-tuned on a set of few-shot samples for the given generalization setting (cross-domain or otherwise), and then tested under a few-shot inference setting as shown in Figure 1.

2.1 Generalization

Most of the state-of-the-art (SOTA) Text-to-SQL semantic parsing systems are built upon large sequence-to-sequence (Seq2Seq) language models. Seq2Seq architectures usually struggle with compositional generalization Kim and Linzen [2020] that is required to generalize in *Cross-domain* and *In-domain* settings. The most coarse form of generalization faced by Text-to-SQL system in a realistic setting is that of cross-domain generalization with novel databases. This setting is captured extensively by existing work in Text-to-SQL semantic parsing. While such *Cross-domain*

[2]The Spider-Gen dataset is publicly available here: `https://github.com/ManasiPat/Spider-Gen`

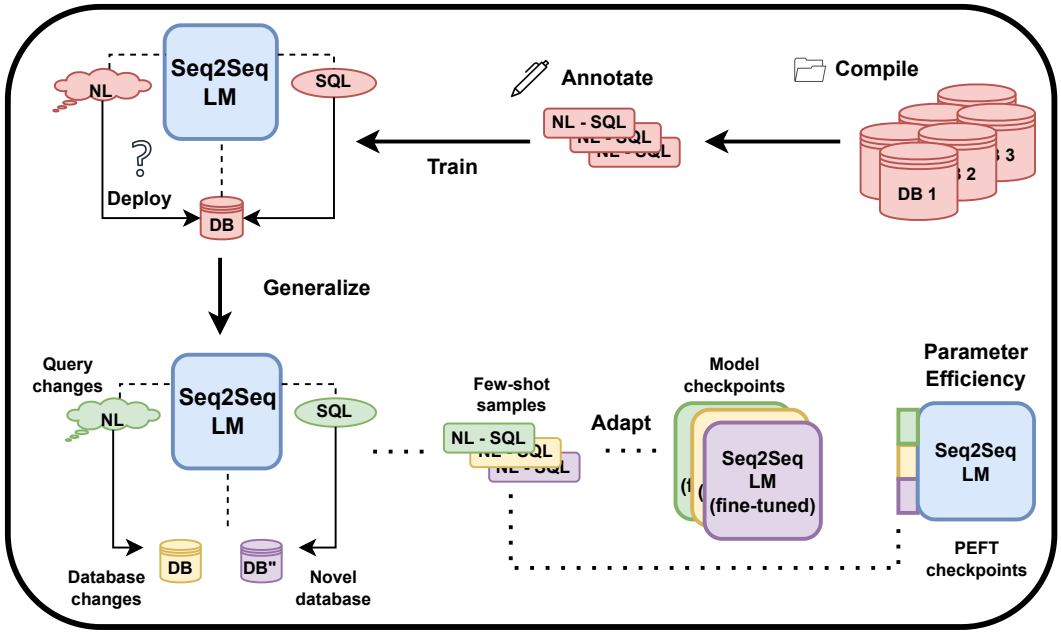

Figure 1: The classic Text-to-SQL modeling and evaluation paradigm with zero-shot inference (top). Various types of generalizations required in Text-to-SQL (bottom-left). Few-shot adaptation for transfer learning over unseen generalizations during training (bottom-middle). Parameter-efficient fine-tuning (PEFT) for generalization in Text-to-SQL (bottom-right).

evaluation covers compositional generalization on a broader scale, subtle sub-types of compositional generalizations are required with respect to the more frequent and specific changes in the *In-domain* training databases and querying. Hence, we additionally introduce two other forms of generalizations which cover finer high-granular changes for the *In-domain* databases.

Structural Generalization: Any change in the intended querying usage results in a change in the compositions and the complexity of the required SQL queries. Adapting to such change requires models to generalize to structurally more difficult queries. We use the query complexity measure defined by [Yu et al., 2018] to annotate SQL query complexity as - *easy*, *medium*, *hard*, or *extra-hard*. Under this setting, for a given particular database - the model is only exposed to SQL queries of lower complexity during training, and then evaluated against SQL queries of higher complexity during testing. We cover two levels of SQL query complexity in Spider-Gen: ***hard*** and ***extra-hard***.

Lexical Generalization: Any modification to the existing databases like addition or deletion of tables, columns and rows, or change in naming conventions and database constraints can result in significant lexical changes in the database schema - which is an important part of the input to Text-to-SQL models. Adapting to such frequent database modifications requires models to generalize lexically to new unseen lexical items from the database schema and content (tables, columns, values, etc.). Under this setting, for a given particular database - the model is only exposed to SQL queries of all - but one held-out database table during training, and then evaluated against SQL queries for the held-out unseen table during testing.

2.2 Dataset

We construct a new dataset for the Spider-Gen benchmark by modifying and combining several existing Text-to-SQL datasets as shown in Table 1. The Spider Train and Development databases are sourced from Yu et al. [2018]. Whereas, the KaggleDBQA databases are sourced from Lee et al. [2021]. We also include several old Text-to-SQL benchmark databases which are compiled by Finegan-Dollak et al. [2018]. Using databases from different datasets in such a manner ensures that the models are evaluated for robustness against a change in data distribution. Based on their intended use-case, the databases in the Spider-Gen dataset are classified in five categories as shown in

Database Category	Description	Source Dataset	Train split	Held-out split	Adapt split	Test split
Training	Databases used to train the model, whose schema and query complexity remain unchanged post-training	**Spider Train**	Y	Y	N	N
Validation	Databases used to validate the model, whose schema and query complexity remain unchanged post-training	**Spider Dev**	Y	N	N	N
Cross-domain Generalization	Databases not seen during the training	**Spider Dev KaggleDBQA**	N	N	Y	Y
Lexical Generalization	Databases seen during training whose schema is modified post-training	**Spider Train Old datasets**	Y	Y	Y	Y
Structural Generalization	Databases seen during training whose queries become more complex post-training	**Spider Train Old datasets**	Y	Y	Y	Y

Table 1: The details of various categories of databases and data sample splits that are used in the Spider-Gen dataset. {Y: Yes, N: No}

Table 1. While the first two categories (Training and Validation) are used to train the base Text-to-SQL model, the final three categories are used to evaluate the trained model against the three types of generalizations discussed in Section2.1 under a zero-shot as well as few-shot setting. For each type of generalization, we source databases from both Spider and non-Spider datasets to ensure diversity in data distributions.

The *Cross-domain* generalization setting is covered by using the databases from the KaggleDBQA dataset and half the databases from the Spider Development dataset.[3] This gives us a total of 10 cross-domain generalization databases for Spider, and 8 databases from a different data distribution of the KaggleDBQA dataset. Within the *In-domain* databases (seen during training) that undergo certain changes, we focus on *Lexical* and *Structural* generalization. The databases for Lexical generalization are carefully chosen by sampling the top-2 databases each from the Spider and the Old datasets according to the maximum number of samples per table. Similarly, the databases for Structural generalization are chosen by sampling the top-2 databases according to the maximum number of SQL queries for the target complexity. Such filtering of databases ensures an availability of a high number of samples for few-shot adaptation as well as test evaluation. This results in two Spider databases, and two databases from a different data distribution of old datasets for Lexical, Structural - Hard, and Structural - Extra Hard generalization settings. This leaves us with a total of 12 generalization databases from the *In-domain* training data. Example SQL queries for the newly introduced Lexical and Structural Generalization settings can be found in Appendix B.

2.3 Evaluation

We cover three important types of evaluation settings in the Spider-Gen benchmark:

Zero-shot (Test data): Under this setting, a given Text-to-SQL model is directly tested without any adaptation or transfer learning for the target generalization. This setting is equivalent to the existing standard Cross-domain evaluation settings in Text-to-SQL, additionally extended to Lexical and Structural generalizations.

Few-shot (Test data) : Under this setting, a given Text-to-SQL model is fine-tuned on a set of few-shot samples for the target generalization. Here, we use a setting equivalent to that of the recently released KaggleDBQA dataset [Lee et al., 2021], where 30% of samples are used for adaptation. The few-shot samples are used to adapt the model for unseen databases, unseen tables, and unseen query complexity for Cross-domain, Lexical, and Structural generalization respectively.

[3]The Spider Test dataset is not publicly available. Hence, we split the Development set into Test and Validation, where the other Spider Development databases are used as Validation databases (Table 1).

Few-shot (Held-out data) : The pre-training on the data from the *In-domain* databases provides us a base Text-to-SQL model for transfer on the target generalization databases. While few-shot fine-tuning can improve the performance of this base model on the target test generalization samples, this might show a negative effect on the knowledge captured during the *In-domain* training - affecting performance on the *In-domain* training databases which don't need models to generalize. Ideally post fine-tuning, a model should expand it's knowledge to target generalization while retaining the existing knowledge of the task and databases. In order to inspect this possible negative effect during transfer learning, we introduce an additional **Held-out** split. Under this setting, the model is evaluated on samples within the generalizations seen during training, both before and after few-shot fine-tuning on samples from target unseen generalizations.

In order to conduct experiments with the proposed few-shot adaptation based transfer learning with the above evaluation settings, the samples from each database are divided into two or four splits according to the database category. The **Train split** includes samples which are used to train and validate the Text-to-SQL model. Whereas, the samples from the **Held-out split** are used to record the possible negative effects of few-shot transfer. The samples from the **Adapt split** and **Test split** are used to fine-tune and evaluate the model on target generalizations respectively. Here, the samples from the **Train split** and **Held-out split** belong to the same data distributions, covering samples for the database tables and SQL query complexities **seen** during training. Similarly, the samples from the **Adapt split** and **Test split** belong to the same data distributions, covering samples for the database tables and SQL query complexities that are **NOT seen** during training.

For the generalization databases, 30% of the generalization samples (samples from unseen tables for lexical generalization, and samples from higher query complexity for structural generalization) are used as Adaptation samples, and the rest 70% samples are used for Testing purposes. These samples are split uniformly with respect to SQL query lengths to avoid any unnecessary latent biases with respect to length generalization. All the non-generalization samples are used as training samples. The training samples from every database are further divided into actual training and held-out training set in the same split of 70%-30% respectively.

Data Source	Samples	Seq2Seq		Prefix Tuning		Prompt Tuning	
		Zero-shot	Few-shot	Zero-shot	Few-shot	Zero-shot	Few-shot
Spider Dev	**HO-Tr**	88.32	**84.89**	85.98	**84.82**	88.37	**88.23**
	TE-Gen	61.11	**76.93**	57.52	**57.58**	60.65	**60.92**
KaggleDBQA	**HO-Tr**	88.32	**83.79**	85.98	**85.82**	88.37	**88.21**
	TE-Gen	21.61	**22.94**	19.43	**19.57**	21.33	**22.35**

Table 2: Average Execution accuracy scores for Cross-domain generalization evaluation under a zero-shot and few-shot transfer setting. {HO-Tr: Combined Held-out Training database samples, TE-Gen: Combined Cross-domain Generalization Test database samples}

3 Experiments

For all our experiments, we follow the standard practice of training Text-to-SQL model by feeding NL queries and serialized schema to the model, and optimizing over the token-level cross-entropy losses over the expected SQL queries. We follow Lin et al. [2020]'s schema serialization technique to encode the schema with its table, column, and value tokens - giving the model access to the database structure and content. We use the CodeT5 model Wang et al. [2021] (a code pre-trained version of the State-of-the-Art T5 model) for all our experiments, referred to as "**Seq2Seq**" model here onward in the text. Further, for all our experiments, we discuss two evaluation settings: **Zero-shot** inference vs. **Few-shot** adaptation.[4]

We use the test-suite query execution accuracy introduced by Zhong et al. [2020] as our evaluation metric. We report results on the target generalization Test splits (**TE-Gen**), as well as the Held-out splits. The Held-out training (**HO-Tr**) split scores represent the model's performance on all the combined Training databases before and after few-shot fine-tuning, whereas the Held-out generalization (**HO-Gen**) split scores specifically represent the model's performance on the held-out

[4]The hyperparameter details for all our experiments can be found in Appendix A.

Generalization	Data Source	Database	Samples	Seq2Seq		Prefix Tuning		Prompt Tuning	
				Zero-shot	Few-shot	Zero-shot	Few-shot	Zero-shot	Few-shot
Lexical	Spider	hr_1	HO-Tr	88.29	82.68	85.94	85.80	88.34	88.34
			HO-Gen	83.33	66.67	83.33	83.33	83.33	83.33
			TE-Gen	35.62	49.32	35.62	35.62	34.25	32.88
		network_2	HO-Tr	88.26	86.00	85.90	85.76	88.21	88.26
			HO-Gen	100.00	100.00	100.00	100.00	100.00	100.00
			TE-Gen	6.06	54.55	12.12	12.12	9.09	12.12
	Others	IMDB	HO-Tr	88.23	84.38	85.68	85.58	88.27	88.37
			HO-Gen	100.00	55.56	88.89	88.89	100.00	100.00
			TE-Gen	20.37	70.37	25.93	27.78	22.22	20.37
		Yelp	HO-Tr	88.34	80.81	86.08	85.60	88.29	88.72
			HO-Gen	83.33	16.67	83.33	83.33	83.33	83.33
			TE-Gen	29.69	65.62	26.56	26.56	34.38	34.38
Structural (Hard)	Spider	music_1	HO-Tr	88.35	84.99	86.00	85.80	88.31	88.50
			HO-Gen	83.33	66.67	83.33	83.33	83.33	83.33
			TE-Gen	38.24	73.53	35.29	41.18	38.24	35.29
		college_2	HO-Tr	88.30	84.48	85.88	85.78	88.30	88.49
			HO-Gen	90.91	90.91	95.45	95.45	90.91	90.91
			TE-Gen	38.30	59.57	38.30	38.30	40.43	38.30
	Others	GeoQuery	HO-Tr	88.03	72.99	85.75	85.45	88.13	88.83
			HO-Gen	94.74	76.32	92.11	92.11	93.42	93.42
			TE-Gen	87.82	95.51	83.97	83.97	87.18	87.18
		Scholar	HO-Tr	88.22	82.99	85.80	85.70	88.22	88.12
			HO-Gen	96.30	96.30	96.30	96.30	96.30	96.30
			TE-Gen	72.22	93.33	70.00	68.89	75.56	73.33
Structural (Extra Hard)	Spider	dorm_1	HO-Tr	88.17	85.32	85.90	85.85	88.22	88.27
			HO-Gen	100.00	89.47	94.74	94.74	100.00	100.00
			TE-Gen	38.46	57.69	23.08	30.77	34.62	34.62
		college_1	HO-Tr	88.60	86.94	86.50	86.50	88.65	88.84
			HO-Gen	71.05	52.63	65.79	68.42	73.68	71.05
			TE-Gen	30.77	50.00	30.77	30.77	23.08	23.08
	Others	Academic	HO-Tr	88.25	83.95	85.98	86.03	88.25	88.54
			HO-Gen	90.91	59.09	90.91	90.91	90.91	90.91
			TE-Gen	29.33	73.33	33.33	32.00	34.67	34.67
		Restaurants	HO-Tr	88.47	84.62	86.11	85.49	88.32	75.54
			HO-Gen	66.67	44.44	77.78	77.78	88.89	100.00
			TE-Gen	19.70	100.00	19.70	24.24	21.21	42.42
AVERAGE			HO-Tr	88.29	**83.34**	85.96	**85.78**	88.29	**87.40**
			HO-Gen	88.38	**67.89**	87.66	**87.88**	90.34	**91.04**
			TE-Gen	37.22	**70.23**	36.22	**37.68**	37.91	**39.05**

Table 3: Execution accuracy scores for Lexical and Structural generalization evaluation under a zero-shot and few-shot setting. {HO-Tr: Combined Held-out Training database samples, HO-Gen: Held-out Generalization samples for database under consideration, TE-Gen: Test Generalization samples for database under consideration}

training samples of the particular generalization database (i.e. performance on seen tables for lexical generalization, and samples from seen lower complexity query for structural generalization). Since, the model is trained only on databases from the Spider dataset, in order to inspect the effect of shift in data distribution, we report results on both Spider and non-Spider databases for all our experiments. We report the results for our Cross-domain generalization experiments in Table 2, and the results for our Lexical and Structural generalization experiments are shown in Table 3.

3.1 Few-shot Transfer with Seq2Seq Model

Under the Cross-domain generalization setting, the Seq2Seq model performs significantly well under a zero-shot setting over the Spider databases (Table 2). However, the model fails to generalize on the non-Spider KaggleDBQA datasets with a significant drop in accuracy. Performing few-shot fine-tuning for transfer over target cross-domain databases increases the performance significantly for Spider databases, and slightly for KaggleDBQA databases. These observations highlight the

importance of using a few-shot transfer learning approach for Text-to-SQL generalization. Further, these also show the necessity of using datasets from varying distributions in the Spider-Gen benchmark for a robust evaluation of generalization in Text-to-SQL. Further, we observe a significant drop in accuracy over the Held-out Training samples post few-shot fine-tuning. This indicates a negative effect of few-shot adaptation on the knowledge captured during training - an important observation which current Text-to-SQL benchmarks do not reveal.

Under the Lexical generalization setting, the Seq2Seq model performs quite well with near-perfect generalization accuracies over seen tables (zero-shot HO-Gen scores in Table 3). However, the model fails to generalize to unseen tables (zero-shot TE-Gen scores) with a significant drop in accuracy. Again, performing few-shot fine-tuning based transfer learning for the unseen table increases the accuracy significantly across all the Spider and non-Spider databases (few-shot TE-Gen scores). However, the model again faces a negative effect of few-shot transfer with significant drop in few-shot accuracies over seen generalizations (HO-Gen) as well on the broader training set (HO-Tr). Similarly, for Structural generalization (both Hard and Extra hard) we see the Seq2Seq model struggle with SQL queries of higher complexity (TE-Gen) under a zero-shot setting, and an improvement in performance with few-shot transfer. The performance drop in generalizing to queries of higher complexity is relatively bigger for the Extra Hard queries as compared to the Hard queries - suggesting certain degree of correlation in query difficulty and structural generalization capabilities of the Seq2Seq model. The negative effect of few-shot adaptation can again be seen with accuracy on Held-out samples. However, the drop in Held-out accuracies is found to be relatively less in transfer for Structural generalization as compared to that in Lexical generalization.

Overall, we observe that performing transfer learning with a few-shot adaptation setting greatly benefits a Seq2Seq model with significant improvements in execution accuracies over both coarse and fine dimensions of generalizations in Text-to-SQL semantic parsing (Table 3). However, these performance gains also come along with the negative effects of transfer on existing knowledge learnt during training of the Seq2Seq model.

3.2 Few-shot Transfer with Parameter-efficient Fine-tuning (PEFT)

The experiments done with the Spider-Gen dataset reveal several limitations of existing Seq2Seq Text-to-SQL models. While performing few-shot fine-tuning with the adaptation samples from the Spider-Gen dataset helps overcome certain generalization issues, it still poses issues with Seq2Seq architectures: **(i)** The models fail to retain the knowledge learnt during training after few-shot fine-tuning (Table 2, Table 3); **(ii)** With an increasing size of SOTA Seq2Seq Text-to-SQL model, performing few-shot fine-tuning for multiple changes (in databases or queries) and generalization becomes infeasible in terms of compute-efficiency and memory required to deploy fine-tuned checkpoints.

Recent progress in Parameter-efficient Fine-tuning (PEFT) methods like Prefix Tuning Li and Liang [2021] and Prompt Tuning Lester et al. [2021] show promise to address both these issues. Since PEFT methods keep the underlying language models frozen during fine-tuning, all the knowledge captured during training is preserved in the model parameters during fine-tuning. Further, the PEFT methods train a relatively very small number of parameters, and the underlying trained and frozen Seq2Seq language model is same across different fine-tuned prompts or prefixes. Hence, this provides an excellent level of compute-efficiency in terms of computational costs for fine-tuning, as well as memory efficiency in terms of deploying a single checkpoint of the large trained language model and multiple relatively smaller fine-tuned prefix or prompt checkpoints.

Prefix Tuning has previously shown competitive performance with full model tuning for various text-generation datasets under a low data setting Li and Liang [2021]. On the other hand, Prompt Tuning has also proven to be competitive with full model tuning for various text-classification tasks Lester et al. [2021]. Hence, we experiment with both Prefix Tuning and Prompt Tuning for few-shot fine-tuning as shown in Figure 1.

Under this setting, we use the Seq2Seq model trained with Training and Validation databases (Table 1) as the underlying language model. Similar to the Seq2Seq model, the Prefixes and Prompts are initialized with their respective versions trained with Training and Validation databases (Table 1), and then fine-tuned with few-shot data for Cross-domain, Lexical, or Structural generalization databases. For Cross-domain generalization, we observe that both Prefix Tuning and Prompt Tuning show slightly lower, yet comparable results with the Seq2Seq model under both zero-shot and few-shot

setting (TE-Gen scores in Table 2). Further, they also show relatively less negative effects of few-shot fine-tuning on the Held-out training samples as compared to the Seq2Seq model (few-shot HO-Gen scores in Table 2). For Lexical generalization, both Prefix Tuning and Prompt Tuning fail to compete with the full tuning of the Seq2Seq model under a few-shot setting (few-shot TE-Gen scores in Table 3). A similar trend is observed for all the databases for the Structural Hard as well Structural Extra Hard generalization settings as well.

Overall, both PEFT methods show competitive results with the Seq2Seq model under a few-shot fine-tuning setting for Cross-domain generalization, while using significantly less compute and memory. However, they show significantly lower performance with the more granular Lexical and Structural generalization evaluation setting. This suggests that unlike the other NLP tasks where Prefix Tuning and Prompt Tuning perform quite well, they fail to address the task of generalization in Text-to-SQL.

4 Conclusion and Future Work

In this paper, we propose introducing few-shot fine-tuning based transfer learning for generalization in the Text-to-SQL semantic parsing. Since, all the existing Text-to-SQL benchmark focus on zero-shot cross-domain inference, we introduce a new benchmark: **Spider-Gen** for evaluating three specific types of generalizations required for Text-to-SQL semantic parsing: **(i)** Cross-domain generalization; **(ii)** Lexical generalization; **(iii)** Structural generalization.

Through our experiments with the Spider-Gen dataset, we show that a Seq2Seq model struggles to perform cross-domain generalization on novel unseen databases from different data distributions. We also show that even for databases seen during training, a Seq2Seq model fails to generalize Lexically and Structurally with respect to fine-grained changes in the database schema and SQL query complexity respectively. These findings highlight the need of a few-shot adaptation based transfer learning paradigm in Text-to-SQL semantic parsing. Hence, we also include few-shot adaptation samples in the Spider-Gen dataset. Our results show that while few-shot fine-tuning of Seq2Seq model shows performance gains across different types of generalization, it comes at the cost of losing out the knowledge learnt during training for the training databases. Hence, reporting results on the Held-out evaluation samples from the Spider-Gen dataset helps in monitoring such negative effects of few-shot adaptation based transfer learning in Seq2Seq Text-to-SQL models.

Through our experiments with the Seq2Seq model, we show several limitations of generalizing with few-shot full model fine-tuning of Seq2Seq models. In order to overcome these limitations, we also explore Parameter-efficient Fine-tuning techniques (PEFT) like Prefix Tuning and Prompt Tuning. These PEFT techniques show at-par performance with full model tuning of a Seq2Seq model with relatively less computation-costs and lesser magnitude of negative effects of transfer. However, they fail to significantly improve over a fully-tuned Seq2Seq model. They also fail to address Lexical and Structural generalization under both zero-shot and few-shot settings. These findings suggest that better PEFT techniques need to be designed for more complex tasks like Text-to-SQL.

While we perform a preliminary set of experiments with the Spider-Gen dataset in this work, the dataset can be used for various benchmarking and interpretability studies to build better generalizing Text-to-SQL semantic parsing systems. We release the dataset publicly under the same licenses as the its underlying component datasets. The dataset is also designed to match the standard format of the original Spider dataset in order to facilitate quick adoption of the benchmark and reproducibility of studies over transfer learning for generalization in Text-to-SQL models. Some possible future research directions with the Spider-Gen dataset can include exploring the sample-efficiency during the few-shot fine-tuning - where one can inspect the nature and frequency of few-shot samples needed to obtain optimal performance under different types of generalizations. Given the diversity in the database domains, data distribution, and generalizations covered in the Spider-Gen dataset - one can also use it to predict an estimate of performance under different settings where no NL-SQL annotations are available to test a particular model. The Spider-Gen dataset can also be used to benchmark models which are to be deployed in real-world practical settings - where performing Lexical and Structural generalization is quite important. Lastly, the dataset can come in handy to predict and optimize transferability across different learning settings, databases, and domains.

References

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1033. URL https://aclanthology.org/P18-1033.

Najoung Kim and Tal Linzen. 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, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.731. URL https://aclanthology.org/2020.emnlp-main.731.

Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. KaggleDBQA: Realistic evaluation of text-to-SQL parsers. 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 2261–2273, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.176. URL https://aclanthology.org/2021.acl-long.176.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.243. URL https://aclanthology.org/2021.emnlp-main.243.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. 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 4582–4597, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4870–4888, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.438. URL https://aclanthology.org/2020.findings-emnlp.438.

Alane Suhr, Ming-Wei Chang, Peter Shaw, and Kenton Lee. 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, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.742. URL https://aclanthology.org/2020.acl-main.742.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.685. URL https://aclanthology.org/2021.emnlp-main.685.

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. 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, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1425. URL https://aclanthology.org/D18-1425.

Ruiqi Zhong, Tao Yu, and Dan Klein. Semantic evaluation for text-to-SQL with distilled test suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 396–411, Online, November 2020. Association for Computational

Linguistics. doi: 10.18653/v1/2020.emnlp-main.29. URL https://aclanthology.org/2020.emnlp-main.29.

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017.

A Hyperparameter Settings

Hyperparameter	Seq2Seq		Prefix Tuning		Prompt Tuning	
	Training	Fine-tuning	Training	Fine-tuning	Training	Fine-tuning
Epochs	50	15	30	50	25	30
Learning Rate	1.00E-04	5.00E-05	5.00E-04	5.00E-05	1.00E-02	1.00E-03
Weight Decay	0	1.00E-06	0	1.00E-06	0	0
Batch Size	128	2	128	2	128	2
LR Scheduler	constant	constant	linear	constant	linear	constant

Table 4: The hyperparameter settings used for training and fine-tuning the Seq2Seq, Prefix Tuning, and Prompt Tuning based models in our experiments.

B Generalization Examples

```
/* TRAIN QUERY: What are the department names, cities, and state
    provinces for each department? */

SELECT T1.department_name, T2.city, T2.state_province FROM departments
     AS T1 JOIN locations AS T2 ON T2.location_id = T1.location_id


/* TEST QUERY: display the employee number, name( first name and last
    name ), and salary for all employees who earn more than the
    average salary and who work in a department with any employee with
     a 'J' in their first name. */

SELECT employee_id, first_name, last_name, salary FROM employees WHERE
      salary > (SELECT Avg (salary) FROM employees) AND department_id
    IN (SELECT department_id FROM employees WHERE first_name LIKE '%J%
    ')
```

Figure 2: Example for Lexical Generalization from the "**hr_1**" database. The database table: "**employees**" and its content is unseen in training queries, and is used in test queries.

```
/* TRAIN QUERY (easy): What are the names of all songs that are
    ordered by their resolution numbers? */

SELECT song_name FROM song ORDER BY resolution


/* TRAIN QUERY (medium): What is the average song rating for each
    language? */

SELECT Avg(rating), languages FROM song GROUP BY languages


/* TEST QUERY (hard): Find the file format that is used by the most
    files. */

SELECT formats FROM files GROUP BY formats ORDER BY Count (*) DESC
    LIMIT 1
```

Figure 3: Example for Structural Generalization (Hard) from the "**music_1**" database. Here, only the SQL queries of complexity "**easy**" and "**medium**" are seen during training. Whereas, the queries of complexity "**hard**" are only seen during testing.

```
/* TRAIN QUERY (easy): How many dorms are in the database? */

SELECT Count(*) FROM dorm


/* TRAIN QUERY (medium): Find the average age of all students living
    in the each city. */

SELECT Avg(age), city_code FROM student GROUP BY city_code


/* TRAIN QUERY (hard): What are the names of all the dorms that don't
    have any amenities? */

SELECT dorm_name FROM dorm WHERE dormid NOT IN (SELECT dormid FROM
    has_amenity)


/* TEST QUERY (extra hard): Find the name of dorms which have TV
    Lounge but no Study Room as amenity. */

SELECT T1.dorm_name FROM dorm AS T1 JOIN has_amenity AS T2 ON T1.
    dormid = T2.dormid JOIN dorm_amenity AS T3 ON T2.amenid = T3.
    amenid WHERE T3.amenity_name = 'TV Lounge' EXCEPT SELECT T1.
    dorm_name FROM dorm AS T1 JOIN has_amenity AS T2 ON T1.dormid = T2
    .dormid JOIN dorm_amenity AS T3 ON T2.amenid = T3.amenid WHERE T3.
    amenity_name = 'Study Room'
```

Figure 4: Example for Structural Generalization (Extra Hard) from the "**dorm_1**" database. Here, only the SQL queries of complexity "**easy**", "**medium**" and "**hard**" are seen during training. Whereas, the queries of complexity "**extra hard**" are only seen during testing.