Diverse Parallel Data Synthesis for Cross-Database Adaptation of Text-to-SQL Parsers

Abhijeet Awasthi **Ashutosh Sathe** **Sunita Sarawagi**
Indian Institute of Technology Bombay, India
{awasthi,absathe,sunita}@cse.iitb.ac.in

Abstract

Text-to-SQL parsers typically struggle with databases unseen during the train time. Adapting parsers to new databases is a challenging problem due to the lack of natural language queries in the new schemas. We present REFILL, a framework for synthesizing high-quality and textually diverse parallel datasets for adapting a Text-to-SQL parser to a target schema. REFILL learns to retrieve-and-edit text queries from the existing schemas and transfers them to the target schema. We show that retrieving diverse existing text, masking their schema-specific tokens, and refilling with tokens relevant to the target schema, leads to significantly more diverse text queries than achievable by standard SQL-to-Text generation methods. Through experiments spanning multiple databases, we demonstrate that fine-tuning parsers on datasets synthesized using REFILL consistently outperforms the prior data-augmentation methods.

1 Introduction

Natural Language interface to Databases (NLIDB) that translate text queries to executable SQLs is a challenging task in the field of Semantic Parsing (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Berant et al., 2013). In addition to understanding the natural language and generating an executable output, Text-to-SQL also requires the ability to reason over the schema structure of relational databases. Recently, datasets such as Spider (Yu et al., 2018) comprising of parallel (Text,SQL) pairs over hundreds of schemas have been released, and these have been used to train state-of-the-art neural Text-to-SQL models (Wang et al., 2020; Scholak et al., 2021a; Rubin and Berant, 2021; Scholak et al., 2021b; Xu et al., 2021). However, several studies have independently shown that such Text-to-SQL models fail catastrophically when evaluated on unseen schemas from the real-world databases (Suhr et al., 2020; Lee et al., 2021; Hazoom et al., 2021). Adapting existing parsers to new schemas is challenging due to the lack of parallel data for fine-tuning the parser.

Synthesizing parallel data, that is representative of natural human generated queries (Wang et al., 2015; Herzig and Berant, 2019), is a long-standing problem in semantic parsing. Several methods have been proposed for supplementing with synthetic data, ranging from grammar-based canonical queries to full-fledged conditional text generation models (Wang et al., 2015; Herzig and Berant, 2019; Zhong et al., 2020; Yang et al., 2021; Zhang et al., 2021; Wang et al., 2021). For Text-to-SQL, data-augmentation methods are primarily based on training an SQL-to-Text model using labeled data from pre-existing schemas, and generating data in the new schemas. We show that the text generated by these methods, while more natural than canonical queries, lacks the rich diversity of natural multi-user queries. Fine-tuning with such data often deteriorates the model performance since the lack of diversity leads to a biased model.

We propose a framework called REFILL (§ 2) for generating diverse text queries for a given SQL workload that is often readily available (Baik et al., 2019). REFILL leverages parallel datasets from several existing schemas, such as Spider (Yu et al., 2018), to first retrieve a diverse set of text paired with SQLs that are structurally similar to a given SQL q (§ 2.1). Then, it trains a novel *schema translator* model for converting the text of the training schema to the target schema of q. The schema translator is decomposed into a `mask` and `fill` step to facilitate training without direct parallel examples of schema translation. Our design of the `mask` module and our method of creating labeled data for the `fill` module entails non-trivial details that we explain in this paper (§ 2.2). REFILL also incorporates a method of filtering-out inconsistent (Text,SQL) pairs using an independent binary classifier (§ 2.3), that provides more useful

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 11548–11562
December 7-11, 2022

quality scores, than the cycle-consistency based filtering (Zhong et al., 2020). Our approach is related to retrieve-and-edit models that have been used for semantic parsing (Hashimoto et al., 2018), dialogue generation (Chi et al., 2021), translation (Cai et al., 2021), and question answering (Karpukhin et al., 2020). However, our method of casting the "edit" as a two-step mask-and-fill schema translation model is different from the prior work.

We summarize our contributions as follows: (i) We propose the idea of retrieving and editing natural text from several existing schemas for transferring it to a target schema, obtaining higher text diversity compared to the standard SQL-to-Text generators. (ii) We design strategies for masking schema-specific words in the retrieved text and training the REFILL model to fill in the masked positions with words relevant to the target schema. (iii) We filter high-quality parallel data using a binary classifier and show that it is more efficient than existing methods based on cycle-consistency filtering. (iv) We compare REFILL with prior data-augmentation methods across multiple schemas and consistently observe that fine-tuning Text-to-SQL parsers on data generated by REFILL leads to more accurate adaptation.

2 Diverse data synthesis with REFILL

Our goal is to generate synthetic parallel data to adapt an existing Text-to-SQL model to a target schema unseen during training. A Text-to-SQL model $\mathcal{M} : \mathcal{X}, \mathcal{S} \mapsto \mathcal{Q}$ maps a natural language question $x \in \mathcal{X}$ for a database schema $s \in \mathcal{S}$, to an SQL query $\hat{q} \in \mathcal{Q}$. We assume a Text-to-SQL model $\mathcal{M}$ trained on a dataset $\mathcal{D}_{\text{train}} = \{(x_i, s_i, q_i)\}_{i=1}^{N}$ consisting of text queries x_i for a database schema s_i, and the corresponding gold SQLs q_i. The train set $\mathcal{D}_{\text{train}}$ typically consists of examples from a wide range of schemas $s_i \in \mathcal{S}_{\text{train}}$. For example, the Spider dataset (Yu et al., 2018) contains roughly 140 schemas in the train set. We focus on adapting the model $\mathcal{M}$ to perform well on a target schema s different from the training schemas in $\mathcal{S}_{\text{train}}$. To achieve this, we present a method of generating synthetic data $\mathcal{D}_{\text{syn}}$ of Text-SQL pairs containing diverse text queries for the target schema s. We fine-tune the model $\mathcal{M}$ on $\mathcal{D}_{\text{syn}}$ to adapt it to the schema s. Our method is agnostic to the exact model used for Text-to-SQL parsing. We assume that on the new schema s we have a workload $\mathcal{QW}_s$ of SQL queries. Often in existing databases a substantial SQL workload is already available in the query logs at the point a DB manager decides to incorporate the NL querying capabilities (Baik et al., 2019). The workload is assumed to be representative but not exhaustive. In the absence of a real workload, a grammar-based SQL generator may be used (Zhong et al., 2020; Wang et al., 2021).

Algorithm 1: Data Synthesis with REFILL

input: $\mathcal{QW}_s$, $\mathcal{M}$, $\mathcal{D}_{\text{train}}$
$\mathcal{D}_{\text{syn}} \leftarrow \phi$
for $q \leftarrow$ SampleSQLQueries $(\mathcal{QW}_s)$ **do**
  $\{q_r, x_r\} \leftarrow$ **RetrieveRelatedPairs**$(q, \mathcal{D}_{\text{train}})$
  $\{x_r^{\text{masked}}\} \leftarrow$ **MaskSchemaTokens**$(\{q_r, x_r\})$
  $\{x_r^{q}\} \leftarrow$ **EditAndFill**$(\{q, x_r^{\text{masked}}\})$
  $\mathcal{D}_{\text{syn}} \leftarrow \mathcal{D}_{\text{syn}} \cup$ **Filter**$(q, \{x_r^{q}\})$
$\mathcal{M}_{\text{new}} \leftarrow$ fine-tune$(\mathcal{M}, \mathcal{D}_{\text{syn}})$

Figure 1 and Algorithm 1 summarizes our method for converting a workload $\mathcal{QW}_s$ of SQL queries into a synthetic dataset $\mathcal{D}_{\text{syn}}$ of Text-SQL pairs containing diverse text queries. Given an SQL query $q \in \mathcal{QW}_s$ for the target schema s, our method first retrieves related SQL-Text pairs $\{q_r, x_r\}_{r=1}^{R}$ from $\mathcal{D}_{\text{train}}$ on the basis of a tree-edit-distance measure such that the SQLs $\{q_r\}_{r=1}^{R}$ in the retrieved pairs are structurally similar to the SQL q (§ 2.1). We then translate each retrieved text query x_r so that its target SQL changes from q_r to q on schema s (§ 2.2). We decompose this task into two steps: masking out schema specific tokens in x_r, and filling the masked text to make it consistent with q using a conditional text generation model $\mathcal{B}$ like BART (Lewis et al., 2020). The translated text may be noisy since we do not have direct supervision to train such models. Thus, to improve the overall quality of the synthesized data we filter out the inconsistent SQL-Text pairs using an independent binary classifier (§ 2.3). Finally, we adapt the Text-to-SQL model $\mathcal{M}$ for the target schema s by fine-tuning it on the diverse, high-quality filtered data $\mathcal{D}_{\text{syn}}$ synthesized by REFILL.

2.1 Retrieving related queries

Given an SQL $q \in \mathcal{QW}_s$ sampled from SQL workload, we extract SQL-Text pairs $\{q_r, x_r\} \in \mathcal{D}_{\text{train}}$, from the train set such that the retrieved SQLs $\{q_r\}$ are structurally similar to the SQL q. We utilize tree-edit-distance (Pawlik and Augsten, 2015, 2016) between the relational algebra trees of SQLs q and q_r — smaller distance implies higher structural similarity. Since the retrieved SQLs come

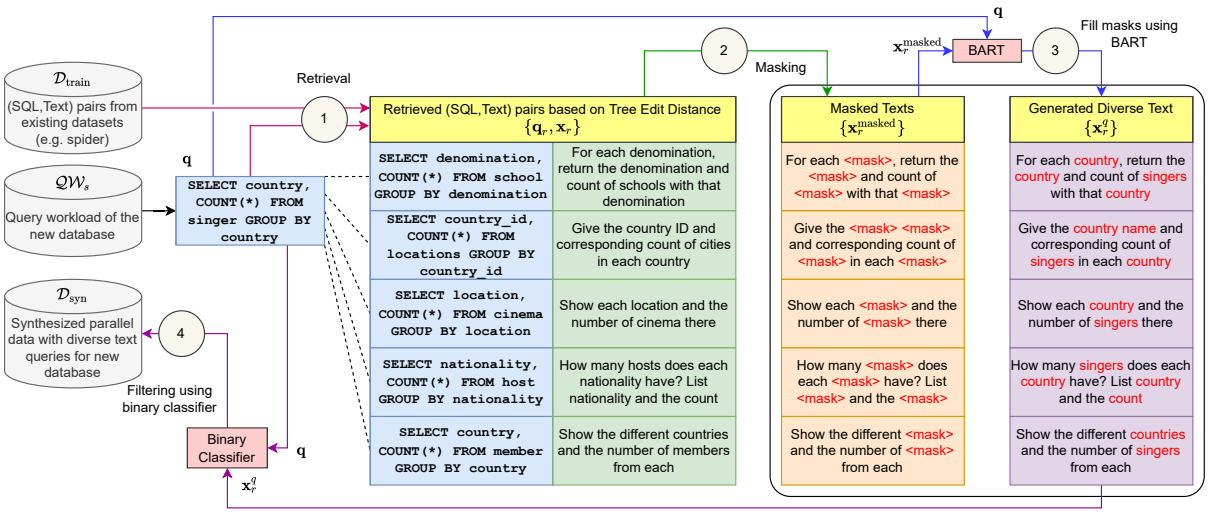


Figure 1: Diverse parallel data synthesis by retrieving-and-editing existing examples using REFILL. Given an SQL q from a new schema, REFILL **(1)** Retrieves SQL-Text pairs from an existing dataset (§ 2.1) where the SQLs are structurally similar to q (indicated by dashed lines). **(2)** Since the retrieved text come from a different schema, we mask-out the schema-specific words (§ 2.2). **(3)** The masked text and the SQL q are then translated into the target schema via an Edit and Fill step that uses a conditional text generation model like BART (§ 2.2). In this way, we transfer the text from multiple existing schemas to generate diverse text for the new schemas. **(4)** Finally, we use a binary classifier as a filtering model to retain only the consistent Text-SQL pairs in the output dataset (§ 2.3).

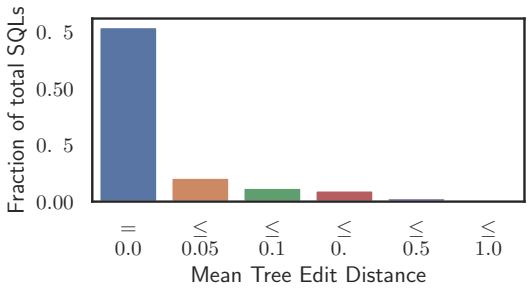


Figure 2: Frequency distribution of average tree-edit-distance between SQLs and their three nearest neighbours from other schemas within Spider's train set.

from different schemas, we modify the tree-edit-distance algorithm to ignore the schema names and the database values. The tree-edit-distance is further normalized by the size of the larger tree. We only consider the $\{q_r, x_r\}$ pairs where the SQLs $\{q_r\}$ have a distance of less than 0.1 w.r.t. the SQL q. Within datasets like Spider that span hundreds of schemas, it is often possible to find several SQLs structurally similar to a given SQL q. For example, in Spider we found that 76% of the train SQLs contain at least three zero-distance (structurally identical) neighbours in other schemas. In Figure 2, we present more detailed statistics.

2.2 Translating text of related queries

Our next goal is to translate the retrieved x_r from being a text for SQL q_r to a text $\hat{x}$ for SQL q , where $q \approx q_r$ structurally. However, we do not have a readily labeled dataset to learn a model that translates x_r to $\hat{x}$ while being consistent with q. We therefore decompose this task into two steps: 1) A simpler task of masking schema-specific tokens in x_r to get a template x_r^{masked} and 2) A conditional text generation model that maps (x_r^{masked}, q) to the text $\hat{x}$ consistent with q, by filling the masked positions in x_r^{masked} as per q. We re-purpose $\mathcal{D}_{\text{train}}$ to get indirect supervision for training the text generation model. We now present each step in detail.

Masking the retrieved text Converting the retrieved text queries $\{x_r\}$ to masked templates $\{x_r^{\text{masked}}\}$ is a critical component of REFILL's pipeline since irrelevant tokens like references to schema elements of the original database can potentially misguide the text generation module. Our initial approach was to mask tokens based on a match of text tokens with schema names and manually refined schema-to-text linked annotations as in Lei et al. (2020). However, this approach failed to mask all schema-related terms since their occurrences in natural text often differed significantly from schema names in the database. Table A7 shows some anecdotes. Consequently, we designed

a simple frequency-based method of masking that is significantly more effective for our goal of using the masked text to just guide the diversity. For each word that appears in the text queries of the train set, we count the number of distinct databases where that word gets mentioned at least once. For example, common words like {'show', 'what', 'list', 'order'} get mentioned in more than 90% of the schemas, and domain specific words like {'countries', 'government'} occur only in text queries of a few schemas. We mask out all the words that appear in less than 50% of the schemas. The words to be masked are replaced by a special token MASK, and consecutive occurrences of MASK are collapsed into a single MASK token. Thus we obtain masked templates $\{x_r^{\text{masked}}\}$ retaining minimal information about their original schema.

Editing and Filling the masked text Given a masked template x_r^{masked}, and an SQL query q, we wish to edit and fill the masked portions in x_r^{masked} to make it consistent with the SQL q. We utilize a conditional text generation model $\mathcal{B}$ like BART (Lewis et al., 2020) for this purpose. We first convert q into a pseudo-English representation q^{Eng} similar to Shu et al. (2021), to make it easier for $\mathcal{B}$ to encode q. In addition, we wrap the table, column, or value tokens in q^{Eng} with special tokens to provide explicit signals to the text generation model $\mathcal{B}$ that such tokens are likely to appear in the output text $\hat{x}$. Next, we concatenate the tokens in x_r^{masked} and q^{Eng} for jointly encoding them as an input to $\mathcal{B}$. The output of $\mathcal{B}$'s decoder is text $\hat{x}$, which is expected to be consistent with the SQL q.

Since we do not have direct supervision to fine-tune $\mathcal{B}$ for this task, we present a method of repurposing $\mathcal{D}_{\text{train}}$ for fine-tuning $\mathcal{B}$. $\mathcal{D}_{\text{train}}$ contains SQL-Text pairs (q_i, x_i) from various schemas s_i. A *Naïve* way to train $\mathcal{B}$ is to provide $[x_i^{\text{masked}}|q_i^{\text{Eng}}]$, the concatenation of x_i^{masked} and q_i^{Eng} as an input to the encoder and maximize the likelihood of x_i in the decoder's output. This way the decoder of $\mathcal{B}$ learns to refill the masked tokens in x_i^{masked} by attending to q_i^{Eng} to recover x_i in the output. While useful for learning to refill the masked positions, this *Naïve* method of training $\mathcal{B}$ is mismatched from its use during inference in two ways: (i) For a given SQL q, REFILL might fail to retrieve a similar structure neighbour of q_i from $\mathcal{D}_{\text{train}}$. In such cases, $\mathcal{B}$ should be capable of falling back to pure SQL-to-Text generation mode to directly translate q into $\hat{x}$. (ii) During inference, x_r^{masked} and q come from different schemas. However, during *Naïve* training, the masked text x_i^{masked} and the SQL q_i are derived from the same example (q_i, x_i). To address these two limitations, we train $\mathcal{B}$ in a more *Robust* manner as follows: (a) For a random one-third of the train steps we train $\mathcal{B}$ in the Naïve way, allowing $\mathcal{B}$ to learn the filling of the masked tokens using q_i^{Eng}. (b) For another one-third, we pass only q_i^{Eng} as an input and maximize the likelihood of x_i. This ensures that model is capable of generating the text from the q_i^{Eng} alone, if the templates x_i^{masked} are unavailable or noisy. (c) For the remaining one-third, we first retrieve an SQL-Text pair (q_j, x_j), from a different schema such that the SQL q_j is structurally similar to q_i (§ 2.1), and the word edit distance between the masked templates x_i^{masked} and x_j^{masked} is also small. We can then replace x_i^{masked} with x_j^{masked} and encode $[x_j^{\text{masked}}|q_i^{\text{Eng}}]$ as an input to $\mathcal{B}$ and maximize the likelihood of x_i in the decoder's output. This step makes the training more consistent with the inference, as x_j^{masked} and q_i^{Eng} now come from different schemas. In § 5.4, we justify training *Robustly* compared to *Naïve* training.

2.3 Filtering the Generated Text

Since the data synthesized using REFILL is used to fine-tune a downstream Text-to-SQL parser, we learn a Filtering model $\mathcal{F} : (\mathcal{X}, \mathcal{Q}) \mapsto \mathbb{R}$ to discard inconsistent examples from the generated dataset. $\mathcal{F}$ assigns lower scores to inconsistent Text-SQL pairs. For each SQL $q \in \mathcal{QW}_s$, we select the top-5 sentences generated by REFILL and discard all the sentences that are scored below a fixed threshold as per the filtering model. Existing work depended on a trained Text-to-SQL parser $\mathcal{M}$ to assign cycle-consistency scores (Zhong et al., 2020). However, we show that cycle-consistency filtering favors text on which $\mathcal{M}$ already performs well, and hence does not result in a useful dataset for fine-tuning $\mathcal{M}$.

We instead train a filtering model $\mathcal{F}$ as a binary classifier, independent of $\mathcal{M}$. The Text-SQL pairs $\{(x_i, q_i)\}$ in the training set $\mathcal{D}_{\text{train}}$, serve as positive (consistent) examples and we synthetically generate the negative (inconsistent) examples as follows: (i) Replace DB values in the SQL q_i with arbitrary values sampled from the same column of the database. (ii) Replace SQL-specific tokens in q_i with their corresponding alternates e.g. replace ASC with DESC, or '>' with '<'. (iii) Cascade previous two perturbations. (iv) Replace the entire SQL q_i with a randomly chosen SQL q_j from the same

schema. (v) Randomly drop tokens in the text query x_i with a fixed probability of 0.3. (vi) Shuffle a span of tokens in the text query x_i, with span length set to 30% of the length of x_i. Thus, for a given Text-SQL pair (x_i, q_i) we obtain six corresponding negative pairs $\{(x_j^n, q_j^n)\}_{j=1}^6$. Let s_i be the score provided by the filtering model for the original pair (x_i, q_i) and $\{s_j\}_{j=1}^6$ be the scores assigned to the corresponding negative pairs $\{(x_j^n, q_j^n)\}_{j=1}^6$. We supervise the scores from the filtering model using a binary-cross-entropy loss over the Sigmoid activations of scores as in Equation 1.

$$\mathcal{L}_{\text{bce}} = -\log \sigma(s_i) - \sum_{j=1}^{6} \log \sigma(1 - s_j) \quad (1)$$

To explicitly contrast an original pair with its corresponding negative pairs we further add another Softmax-Cross-Entropy loss term.

$$\mathcal{L}_{\text{xent}} = -\log \frac{\exp(s_i)}{\exp(s_i) + \sum_{j=1}^{6} \exp(s_j)} \quad (2)$$

3 Related Work

SQL-to-Text generation Many prior works perform training data augmentation via pre-trained text generation models that translate SQLs into natural text (Guo et al., 2018; Zhong et al., 2020; Shi et al., 2020; Zhang et al., 2021; Wang et al., 2021; Yang et al., 2021; Shu et al., 2021). For example, Wang et al. (2021) fine-tune BART (Lewis et al., 2020) on parallel SQL-Text pairs to learn an SQL-to-Text translation model. Shu et al. (2021) propose a similar model that is trained in an iterative-adversarial way along with an evaluator model. The evaluator learns to identify inconsistent SQL-Text pairs, similar to our filtering model. To retain high quality synthesized data Zhong et al. (2020) additionally filter out the synthesized pairs using a pre-trained Text-to-SQL model based on cycle consistency, that we show to be sub-optimal (§ 5.5). The SQL workload in most of the prior work was typically sampled from hand-crafted templates or a grammar like PCFG induced from existing SQLs, or crawling SQLs from open-source repositories Shi et al. (2020). However, database practitioners have recently drawn attention to the fact that SQL workloads are often pre-existing and should be utilized (Baik et al., 2019).

Retrieve and Edit Methods Our method is related to the retrieve-and-edit framework, which has been previously applied in various NLP tasks. In Semantic Parsing, question and logical-form pairs from the training data relevant to the test-input question are retrieved and edited to generate the output logical forms in different ways (Shaw et al., 2018; Das et al., 2021; Pasupat et al., 2021; Gupta et al., 2021). In machine translation, memory augmentation methods retrieve-and-edit examples from translation memory to guide the decoder's output (Hossain et al., 2020; Cai et al., 2021). Our editing step — masking followed by refilling is similar to style transfer methods that minimally modify the input sentence with help of retrieved examples corresponding to a target attribute (Li et al., 2018). In contrast to learning a retriever, we find simple tree-edit distance to be an effective metric for retrieving the relevant examples for our task.

4 Experimental Set-up[1]

We adapt pretrained Text-to-SQL parsers on multiple database schemas unseen during the train time. Here, we describe the datasets, models, and evaluation metrics used in our experiments.

Datasets: We primarily experiment with the Spider dataset (Yu et al., 2018). Spider's train split contains 7000 Text-to-SQL examples spanning 140 database schemas, and the dev split contains roughly 1000 examples spanning 20 schemas[2]. Since individual schemas in the dev split typically contain less than 50 examples, to evaluate on a larger set of examples we adapt and evaluate the Text-to-SQL parser on groups of similar schemas instead of individual schemas. We create 4 groups, with each group having database schemas from a similar topic. For example, Group-1 consists of databases `{Singer, Orchestra, Concerts}`. We utilize all the available Text-SQL pairs in each group for evaluation. In appendix Table A.1, we provide detailed statistics about each group. On average, each group contains 69 unique SQLs and 131 evaluation examples. To simulate a query workload $\mathcal{QW}_s$ for each group, we randomly select 70% of the available SQLs and replace the constant-values in the SQLs with values sampled from their corresponding column in the database. We also evaluate on query workloads of size 30% and 50% of the available SQL queries. The SQL queries in the workload are translated using REFILL or an SQL-to-Text model, and the resulting Text-SQL pairs are

[1] Code: github.com/awasthiabhijeet/refill

[2] Spider's test-split is inaccessible as of 10/26/2022

	Group 1		Group 2		Group 3		Group 4		Average	
Method	EM	EX	EM	EX	EM	EX	EM	EX	EM	EX
BASE-M	80.9	84.3	64.8	67.2	64.0	65.9	45.8	35.8	63.8	63.3
L2S (Wang et al., 2021)	**88.7**	**87.8**	61.3	62.1	62.8	61.0	42.5	35.0	63.8	61.4
GAZP (Zhong et al., 2020)	85.2	85.2	58.9	66.9	70.1	60.5	52.5	40.8	66.6	63.3
SNOWBALL (Shu et al., 2021)	85.2	**87.8**	59.7	60.5	64.0	65.9	44.2	38.3	63.2	63.1
REFILL (Ours)	**88.7**	87.0	**69.7**	**73.8**	**73.2**	**70.1**	**55.8**	**45.0**	**71.8**	**68.9**

Table 1: Results for finetuning a base semantic parser (SMBOP) on Text-SQL pairs generated using various SQL-to-Text baselines and REFILL (§ 5.1). REFILL provides consistent gains over the base model across all the database groups, while gains from other methods are often negative or small.

then used to fine-tune a base Text-to-SQL parser.

We further experiment with four datasets outside Spider in Section 5.6. We work with GeoQuery (Zelle and Mooney, 1996), Academic (Li and Jagadish, 2014), IMDB and Yelp (Navid Yaghmazadeh and Dillig, 2017). We utilize the preprocessed version of these datasets open-sourced by Yu et al. (2018). In appendix Table A2, we present statistics about each of the four datasets.

Text-to-SQL parser: We experiment with SMBOP (Rubin and Berant, 2021) as our base Text-to-SQL parser, and utilize author's implementation. The SMBOP model is initialized with a ROBERTA-BASE model, followed by four RAT layers, and trained on the train split of Spider dataset. The dev set used used for selecting the best model excludes data from the four held-out evaluation groups.

Edit and Fill model: We utilize a pre-trained BART-BASE as our conditional text generation model for editing and filling the masked text. The model is fine-tuned using the train split of Spider dataset as described in Section 2.2

Filtering Model: We train a binary classifier based on a ROBERTA-BASE checkpoint on Spider's train split to filter out inconsistent SQL-Text pairs as described in Section 2.3.

Baselines: For baseline SQL-to-Text generation models, we consider recently proposed models like L2S (Wang et al., 2021), GAZP (Zhong et al., 2020), and SNOWBALL (Shu et al., 2021). All the baselines utilize pre-trained language models like BART (Lewis et al., 2020) or BERT (Devlin et al., 2018) for translating SQL tokens to natural text in a standard seq-to-seq set-up. The baselines mostly differ in the way of encoding SQL tokens as an input to the language model. In Section 3, we reviewed the recent SQL-to-Text methods.

Evaluation Metrics We evaluate the Text-to-SQL parsers using the Exact Set Match (EM), and the Exection Accuracy (EX) Yu et al. (2018). The EM metric measures set match for all the SQL clauses and returns 1 if there is a match across all the clauses. It ignores the DB-values (constants) in the SQL query. The EX metric directly compares the results obtained by executing the predicted query $\hat{q}$ and the gold query q on the database.

We provide more implementation details including the hyperparameter settings in appendix A.5.

5 Results and Analysis

We first demonstrate the effectiveness of the synthetic data generated using REFILL for fine-tuning Text-to-SQL parsers to new schemas. We compare with the recent methods that utilize SQL-to-Text generation for training-data augmentation (§ 5.1). We then evaluate the intrinsic quality of the synthetic data generated by different methods in terms of the text diversity and the agreement of the generated text with the ground truth (§ 5.2). We demonstrate that higher text diversity results in better performance of the adapted parsers (§ 5.3). We then justify the key design choices related to masking of the retrieved text and training of the schema translator module that improves the quality of REFILL generated text (§ 5.4). Finally, we demonstrate the importance of using an independent binary classifier over cycle-consistency filtering (§ 5.5).

5.1 Evaluating adapted parsers

In Table 1, we compare the performance of parsers fine-tuned on Text-SQL pairs generated using REFILL and other SQL-to-Text generation baselines. We observe that fine-tuning on high-quality and diverse text generated by REFILL provides consistent performance gains over the base model across all the database groups. On average, REFILL improves the base model by 8.0 EM in comparison to a gain of 2.8 EM by the best baseline (GAZP). We observe that the gains from baseline methods are

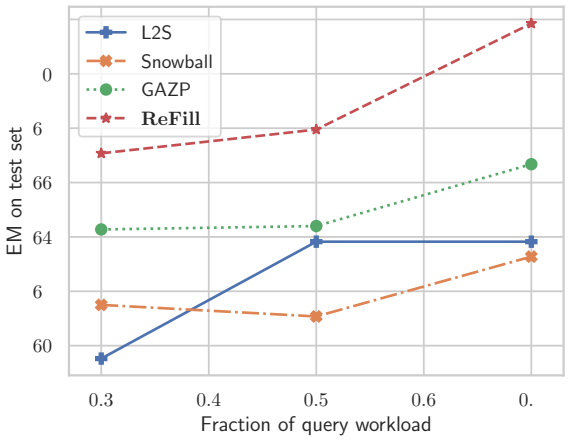


Figure 3: Average EM performance of Text-to-SQL models on the four groups vs. the size of SQL workload (§ 5.1). Data generated by REFILL using 30% SQL workload yields better performance than the data from the existing best baseline utilizing 70% workload.

Method	BLEU ↑ (Quality)	100-SelfBLEU ↑ (Diversity)
Gold-Ref	100	68.8
L2S	38.0	2.2
GAZP	38.8	2.0
SnowBall	40.2	2.8
REFILL	**48.6**	**33.8**

Table 2: Comparison of quality (BLEU) and diversity (100-SelfBLEU) scores across various SQL-to-Text models including REFILL (§ 5.2). Gold-Ref represents the scores corresponding to the reference text.

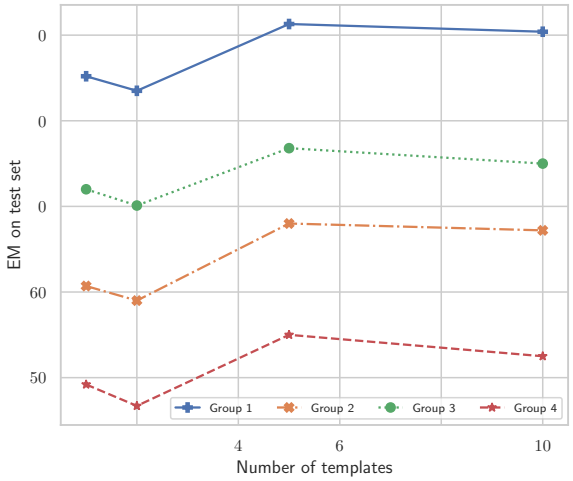


Figure 4: Accuracy of fine-tuned parsers Vs. the number of templates per SQL used by REFILL (§ 5.3).

often small or even negative. REFILL continues to yield positive gains even for smaller workload sizes — In Figure 3, we plot the fraction of the total SQL workload used on the x-axis and the EM of the fine-tuned parsers averaged across all the four groups, on the y-axis. When using the data synthesized by REFILL, the performance of the parser improves steadily with an increasing size of the SQL workload. In contrast, the baseline SQL-to-Text generation methods fail to provide significant improvements. Interestingly, the data synthesized by REFILL using the 30% SQL workload leads to better downstream performance of the adapted parsers than any of the baselines utilizing 70% SQL workload for SQL-to-Text generation.

5.2 Quality and Diversity of generated text

We explain our gains over existing methods due to the increased quality and diversity of the generated text. We measure quality using the BLEU score of the set $S(q)$ of generated text for an SQL q, with the gold text of q as reference. To measure diversity we utilize SelfBLEU (Zhu et al., 2018) that measures the average BLEU score among text in $S(q)$. Lower SelfBLEU implies higher diversity. We evaluate on all the gold SQL-Text pairs available in the Spider's dev set. In Table 2, we compare the quality and diversity of the text generated using REFILL with prior SQL-to-Text generation methods. For each method we generate 10 hypotheses per SQL query, and pick the hypothesis with the highest BLEU to report the overall BLEU scores. To allow baselines to generate more diverse text than the standard beam search, we utilize beam-sampling (Fan et al., 2018; Holtzman et al., 2019). For REFILL, the 10 hypothesis come from using upto 10 retrieved-and-masked templates. We observe that our method of masking and refilling the natural text retrieved from existing datasets allows REFILL to generate higher quality text (+8.4 BLEU) with naturally high text diversity.

5.3 Importance of Text Diversity

Retrieving and editing text from multiple existing examples enables REFILL to generate diverse text. In Figure 4, we show that increased diversity of the generated text leads to improved performance of the fine-tuned parser. We vary the number of retrieved-and-masked templates on the x-axis and plot the performance of the fine-tuned parsers on the y-axis for each group. To maintain the number of synthesized examples the same, the product of beam-samples and the number of retrieved templates is held constant. We observe that fine-tuning

	Naïve Train	Robust Train
Schema-Match	37.2	41.8
Frequency	40.2	43.8

Table 3: Analyzing the impact of design choices related to Schema Translation, by observing BLEU-4 scores of the text generated by REFILL (§ 5.4). Frequency based masking and Robust training leads to a higher quality of the generated text.

	EM	EX
BASE-M	45.8	35.8
No Filtering	40.8	31.7
Cycle Consistent	29.2	22.5
Filtering Model	**48.3**	**36.7**

Table 4: Using an independent filtering model allows us to retain more useful training examples than cycle consistent filtering, leading to better performance of the fine-tuned Text-to-SQL models (§ 5.5).

the parser on more diverse data generated using 5 retrieved templates per SQL provides consistently superior EM performance across all the four groups than using less diverse data obtained by retrieving just one or two templates per SQL. The consistent drops in EM while increasing the retrieved templates from 5 to 10 is explained by the reduction in text diversity. Using 5 retrieved templates yields a $100 - \text{SelfBLEU}$ score of 46.7, while with 10 retrieved templates $100 - \text{SelfBLEU}$ reduces to 33.8. This reduction is due to the inclusion of more similar templates as we increase their number from 5 to 10. Finally, the drop in REFILL's performance with reduced text diversity reconfirms the worse performance of SQL-to-Text baselines reported in Section 5.1 that do not offer enough text diversity.

5.4 Design choices of Schema Translator

In Section 2.2, we described two important design choices: (1) Method of masking schema-relevant tokens and (2) Method of training the Edit-and-Fill model for editing and refilling the masked text. We justify these design choices by comparing the quality of the generated text with each combination of these choices in Table 3. Comparing across rows (Schema-Match Vs Frequency), we observe that Frequency based masking results in 2 to 3 point improvements in BLEU compared to masking by matching schema names. Table A7 shows specific examples where the schema-match method fails to mask sufficiently. In contrast, even though the frequency-based method might over-mask it still suffices for our goal of guiding the text generation model. Comparing across columns (Naïve Train Vs. Robust Train) we observe that specifically training the template filling model for being robust to the input templates also improves quality of the generated text by 3.6 to 4.6 points.

5.5 Importance of Filtering model

Cycle-consistency based filtering (Zhong et al., 2020) rejects a synthesized SQL-Text pair (q, x) if the output SQL $\hat{q}$ generated by the base Text-to-SQL parser for the input text x does not match with the SQL q. We argue that cycle-consistency based filtering is sub-optimal for two reasons: (i) *Data Redundancy*: Since the Text-to-SQL parser is already capable of generating the correct output for the retained examples, fine-tuning on these examples does not offer much improvements. (ii) *Data Loss*: If the base Text-to-SQL model is weak in parsing text-queries for the target database, a large portion of potentially useful training examples get filtered out due to cycle-inconsistency. In response, we train a Filtering model as described in Section 2.3. Since our filtering model is independent of the base parser, it retains many useful generated examples that might otherwise be filtered out by cycle-consistency filtering using a weak Text-to-SQL parser. In appendix Table A6, we provide instances of useful training examples that get filtered because of cycle-inconsistency, but are retained by our filtering model. In Table 4, we compare the base Text-to-SQL parser, with models fine-tuned without any filtering, with cycle-consistent filtering, and with using our filtering model. We focus on Group-4 where the base Text-to-SQL parser is significantly weaker compared to other groups, and use REFILL to synthesize data for the 30% query workload. Not using any filtering, or using cycle-consistent filtering results in worse performance, while applying our filtering model offers significant improvements over the base model.

5.6 Experiments on datasets outside Spider

We further validate our method on four single-database datasets outside Spider, namely GeoQuery (Zelle and Mooney, 1996), Academic (Li and Jagadish, 2014), IMDB and Yelp (Navid Yaghmazadeh and Dillig, 2017). In Table 5, we first report the performance of our base Text-to-SQL parser and observe poor cross-database generalization with an average EM of just 9.7. We adapt

Method	Geo	Acad	IMDB	Yelp	Average
BASE-M	9.8	2.8	19.3	7.2	9.7
L2S	27.9	14.4	24.8	19.8	21.7
GAZP	20.8	16.0	19.3	10.8	16.7
SNOWBALL	25.6	8.8	21.1	18.9	18.6
REFILL(Ours)	**30.1**	**27.6**	**26.6**	**29.7**	**28.5**

Table 5: Evaluation on datasets outside Spider. We continue to observe that fine-tuning on data synthesized by REFILL offers superior EM performance (§ 5.6).

the base parser by fine-tuning it on the synthetic datasets generated by REFILL and other baselines utilizing 30% of the available SQL workload. Table 5 compares the EM accuracy of the adapted parsers. We continue to observe that fine-tuning on datasets synthesized by REFILL consistently outperforms prior SQL-to-Text based data-augmentation baselines. In appendix Table A4, we provide results for 50% and 70% workload settings.

6 Conclusion

We presented REFILL, a framework for generating diverse and high quality parallel data for adapting existing Text-to-SQL parsers to a target database. REFILL translates a given SQL workload into a dataset of Text-SQL pairs containing diverse text queries. To achieve higher diversity, REFILL retrieves and edits examples from existing datasets and transfers them to a target schema. Through extensive experiments across multiple databases, we demonstrate that fine-tuning Text-to-SQL parsers on datasets generated using REFILL result in more accurate adaptation to target schemas. Even with smaller SQL workloads REFILL often outperforms the SQL-to-Text generation baselines utilizing larger SQL workloads.

7 Limitations

This work focuses on synthesizing parallel data containing diverse text queries for adapting pre-trained Text-to-SQL models to new databases. Thus, our current effort toward diverse text query generation using REFILL is limited to the Text-to-SQL semantic parsing task. Extending REFILL for data-augmentation in other semantic parsing or question-answering tasks is an exciting direction we hope to explore as part of future work.

Our experimental set-up assumes a small workload of real SQL queries. As per Baik et al. (2019), a small workload of real SQL queries is a reasonable assumption since SQL query logs are often available for existing in-production databases that are to be supported by a Text-to-SQL service. Synthesizing realistic SQL-query workloads for newly instantiated databases is a challenging and promising direction but different from the diverse text-query generation aspect of our work.

8 Ethical Considerations

Our goal with REFILL is to synthesize parallel data for adapting Text-to-SQL parsers to new schemas. We believe that the real-world deployment of Text-to-SQL or any semantic parser trained on text generated by language models must go through a careful review of any harmful biases. Also, the intended users of any Text-to-SQL service must be made aware that the answers generated by these systems are likely to be incorrect. We do not immediately foresee any serious negative implications of the contributions that we make through this work.

9 Acknowledgements

We thank the reviewers from ACL rolling review for their insightful and quality feedback. We gratefully acknowledge support from IBM Research, specifically the IBM AI Horizon Networks-IIT Bombay initiative. Abhijeet thanks Google for supporting his research with Google PhD Fellowship.

References

Christopher Baik, H. V. Jagadish, and Yunyao Li. 2019. Bridging the semantic gap with SQL query logs in natural language interfaces to databases. In *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*, pages 374–385. IEEE.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1533–1544.

Deng Cai, Yan Wang, Huayang Li, Wai Lam, and Lemao Liu. 2021. Neural machine translation with monolingual translation memory. 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 7307–7318. Association for Computational Linguistics.

Ethan A Chi, Caleb Chiam, Trenton Chang, Swee Kiat Lim, Chetanya Rastogi, Alexander Iyabor, Yutong He, Hari Sowrirajan, Avanika Narayan, Jillian Tang, et al. 2021. Neural, neural everywhere: Controlled

generation meets scaffolded, structured dialogue. *Alexa Prize Proceedings*.

Rajarshi Das, Manzil Zaheer, Dung Thai, Ameya Godbole, Ethan Perez, Jay-Yoon Lee, Lizhen Tan, Lazaros Polymenakos, and Andrew McCallum. 2021. Case-based reasoning for natural language queries over knowledge bases. *arXiv preprint arXiv:2104.08762*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, Melbourne, Australia. Association for Computational Linguistics.

Daya Guo, Yibo Sun, Duyu Tang, Nan Duan, Jian Yin, Hong Chi, James Cao, Peng Chen, and Ming Zhou. 2018. Question generation from sql queries improves neural semantic parsing. *arXiv preprint arXiv:1808.06304*.

Vivek Gupta, Akshat Shrivastava, Adithya Sagar, Armen Aghajanyan, and Denis Savenkov. 2021. Retronlu: Retrieval augmented task-oriented semantic parsing. *arXiv preprint arXiv:2109.10410*.

Tatsunori B Hashimoto, Kelvin Guu, Yonatan Oren, and Percy S Liang. 2018. A retrieve-and-edit framework for predicting structured outputs. *Advances in Neural Information Processing Systems*, 31.

Moshe Hazoom, Vibhor Malik, and Ben Bogin. 2021. Text-to-SQL in the wild: A naturally-occurring dataset based on stack exchange data. In *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)*, pages 77–87. Association for Computational Linguistics.

Jonathan Herzig and Jonathan Berant. 2019. Don't paraphrase, detect! rapid and effective data collection for semantic parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3810–3820.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Nabil Hossain, Marjan Ghazvininejad, and Luke Zettlemoyer. 2020. Simple and effective retrieve-edit-rerank text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2532–2538.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. 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. Association for Computational Linguistics.

Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. 2020. Re-examining the role of schema linking in text-to-SQL. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6943–6954. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

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.

Juncen Li, Robin Jia, He He, and Percy Liang. 2018. Delete, retrieve, generate: a simple approach to sentiment and style transfer. In *NAACL-HLT*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Ro{bert}a: A robustly optimized {bert} pretraining approach.

Isil Dillig Navid Yaghmazadeh, Yuepeng Wang 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.

Panupong Pasupat, Yuan Zhang, and Kelvin Guu. 2021. Controllable semantic parsing via retrieval augmentation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.

Mateusz Pawlik and Nikolaus Augsten. 2015. Efficient computation of the tree edit distance. *ACM Transactions on Database Systems (TODS)*, 40(1):1–40.

Mateusz Pawlik and Nikolaus Augsten. 2016. Tree edit distance: Robust and memory-efficient. *Information Systems*, 56:157–173.

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*.

Torsten Scholak, Raymond Li, Dzmitry Bahdanau, Harm de Vries, and Christopher Pal. 2021a. Duorat: Towards simpler text-to-sql models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1313–1321.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021b. Picard - parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. 2020. Learning contextual representations for semantic parsing with generation-augmented pre-training. *arXiv preprint arXiv:2012.10309*.

Chang Shu, Yusen Zhang, Xiangyu Dong, Peng Shi, Tao Yu, and Rui Zhang. 2021. Logic-consistency text generation from semantic parses. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4414–4426.

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*.

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.

Bailin Wang, Wenpeng Yin, Xi Victoria Lin, and Caiming Xiong. 2021. Learning to synthesize data for semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2760–2766.

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.

Tomer Wolfson, Jonathan Berant, and Daniel Deutch. 2021. Weakly supervised mapping of natural language to sql through question decomposition. *ArXiv*, abs/2112.06311.

Peng Xu, Dhruv Kumar, Wei Yang, Wenjie Zi, Keyi Tang, Chenyang Huang, Jackie Chi Kit Cheung, Simon J.D. Prince, and Yanshuai Cao. 2021. Optimizing deeper transformers on small datasets. 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 2089–2102. Association for Computational Linguistics.

Wei Yang, Peng Xu, and Yanshuai Cao. 2021. Hierarchical neural data synthesis for semantic parsing. *arXiv preprint arXiv:2112.02212*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 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*.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, pages 1050–1055.

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

Ao Zhang, Kun Wu, Lijie Wang, Zhenghua Li, Xinyan Xiao, Hua Wu, Min Zhang, and Haifeng Wang. 2021. Data augmentation with hierarchical sql-to-question generation for cross-domain text-to-sql parsing. *arXiv preprint arXiv:2103.02227*.

Victor Zhong, Mike Lewis, Sida I. Wang, and Luke Zettlemoyer. 2020. Grounded adaptation for zero-shot executable semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6869–6882. Association for Computational Linguistics.

Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. 2018. Texygen: A benchmarking platform for text generation models. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1097–1100.

A Appendix

A.1 Dataset Details

Group	Number of queries by hardness					
	easy	medium	hard	extra	total examples	unique SQLs
Group 1	24	60	25	6	115	62
• concert_singer	4	24	13	4	45	26
• singer	6	18	6	0	30	16
• orchestra	14	18	6	2	40	20
Group 2	14	58	16	36	124	65
• dog_kennels	10	36	10	26	82	42
• pets_1	4	22	6	10	42	23
Group 3	46	60	32	26	164	84
• students_transcripts_tracking	26	24	8	20	78	41
• course_teach	8	14	8	0	30	15
• network_1	12	22	16	6	56	28
Group 4	24	46	20	60	120	65
• world_1	24	46	20	60	120	65

Table A1: Number of schemas and statistics of query workload for each group. Related schemas were grouped together in order to obtain larger evaluation sets per group.

Dataset	Number of queries by hardness					
	easy	medium	hard	extra	total examples	unique SQLs
Geoquery (Geo)	224	32	220	77	553	210
Academic (Acad)	20	29	25	107	181	176
IMDB	23	12	48	26	109	75
Yelp	13	29	25	24	111	98

Table A2: Statistics of queries in additional (non-spider) datasets. We utilize the pre-processed versions of these datasets provided by Yu et al. (2018).

A.2 Results in low or medium SQL workload setting

	Group 1		Group 2		Group 3		Group 4		Average	
Method	EM	EX	EM	EX	EM	EX	EM	EX	EM	EX
Results with 30% SQL workload										
BASE-M	80.9	84.3	64.8	**67.2**	64.0	65.9	45.8	35.8	63.8	63.3
L2S	82.6	84.3	60.5	65.3	61.6	63.4	26.7	26.7	57.8	59.9
GAZP	83.5	84.3	61.3	64.5	66.5	67.1	45.8	37.5	64.3	63.3
SNOWBALL	80.0	83.5	59.7	63.7	67.7	**68.3**	39.2	32.5	61.6	62.0
REFILL	**86.1**	**86.1**	**65.6**	65.6	**68.3**	67.1	**48.3**	**36.7**	**67.1**	**63.8**
Results with 50% SQL workload										
BASE-M	80.9	84.3	64.8	67.2	64.0	65.9	45.8	35.8	63.8	63.3
L2S	**89.6**	88.7	66.1	68.5	57.9	58.5	41.7	35.8	63.8	62.8
GAZP	87.8	87.0	58.9	63.7	65.9	**68.9**	45.0	35.0	64.4	63.6
SNOWBALL	83.5	85.2	55.6	66.1	65.2	66.5	40.0	32.5	61.1	62.6
REFILL	88.7	**91.3**	**67.2**	**69.7**	**70.7**	67.1	**45.8**	**38.3**	**68.1**	**66.6**
Results with 70% SQL workload										
BASE-M	80.9	84.3	64.8	67.2	64.0	65.9	45.8	35.8	63.8	63.3
L2S	**88.7**	**87.8**	61.3	62.1	62.8	61.0	42.5	35.0	63.8	61.4
GAZP	85.2	85.2	58.9	66.9	70.1	60.5	52.5	40.8	66.6	63.3
SNOWBALL	85.2	**87.8**	59.7	60.5	64.0	65.9	44.2	38.3	63.2	63.1
REFILL	**88.7**	87.0	**69.7**	**73.8**	**73.2**	**70.1**	**55.8**	**45.0**	**71.8**	**68.9**

Table A3: Evaluation on four groups of schemas held out from Spider's dev set, for varying sizes of query workload {30%, 50%, 70%} used for SQL-to-Text translation.

Method	Geo	Acad	IMDB	Yelp	Average
Results with 30% SQL workload					
BASE-M	9.8	2.8	19.3	7.2	9.7
L2S	27.9	14.4	24.8	19.8	21.7
GAZP	20.8	16.0	19.3	10.8	16.7
SNOWBALL	25.6	8.8	21.1	18.9	18.6
REFILL	**30.1**	**27.6**	**26.6**	**29.7**	**28.5**
Results with 50% SQL workload					
BASE-M	9.8	2.8	19.3	7.2	9.7
L2S	**33.6**	20.4	25.7	18.0	24.4
GAZP	21.5	11.1	23.9	14.4	17.7
SNOWBALL	26.0	24.3	18.3	27.9	24.1
REFILL	27.9	**37.6**	**28.4**	**35.1**	**32.2**
Results with 70% SQL workload					
BASE-M	9.8	2.8	19.3	7.2	9.7
L2S	**33.9**	19.3	29.4	23.4	26.5
GAZP	25.4	13.8	22.0	15.3	19.1
SNOWBALL	30.9	20.9	20.1	**35.1**	26.7
REFILL	32.8	**37.0**	**33.0**	**35.1**	**34.4**

Table A4: EM evaluation on four additional datasets outside Spider, for varying sizes of query workload {30%, 50%, 70%} used for SQL-to-Text translation. Since the contents of Acad, IMDB, and Yelp databases were not publicly accessible to us, we are unable to report EX results on these databases. EX results for GeoQuery appear in Table A5.

	Fraction of SQL workload		
Method	30%	50%	70%
BASE-M	27.2	27.2	27.2
L2S	30.4	**35.1**	37.5
GAZP	20.8	22.9	29.2
SNOWBALL	28.7	28.5	33.8
REFILL	**33.0**	34.2	**38.9**

Table A5: EX accuracy evaluation on GeoQuery dataset, for varying sizes of query workload {30%, 50%, 70%}.

A.3 Examples rejected by cycle-consistency but retained by our filtering model

Generated text	How many countries are governed by Islamic Emirate?
Gold SQL	`SELECT count(*) FROM country WHERE GovernmentForm = 'Islamic Emirate'`
Predicted SQL	`SELECT COUNT(*) FROM country WHERE country.code NOT IN (SELECT countrylanguage.countrycode FROM countrylanguage)`
Generated text	What is the number of languages that are official in Australia?
Gold SQL	`SELECT COUNT(*) FROM country AS T1 JOIN countrylanguage AS T2 ON T1.Code = T2.CountryCode WHERE T1.Name = 'Australia' AND IsOfficial = 'T'`
Predicted SQL	`SELECT COUNT(*) FROM countrylanguage JOIN country ON countrylanguage.countrycode = country.code WHERE country.name = 'Australia'`
Generated text	How many countries have both “Karen" and “Mandarin Chinese" languages?
Gold SQL	`SELECT COUNT(*) FROM (SELECT T1.Name FROM country AS T1 JOIN countrylanguage AS T2 ON T1.Code = T2.CountryCode WHERE T2.Language = 'Karen' INTERSECT SELECT T1.Name FROM country AS T1 JOIN countrylanguage AS T2 ON T1.Code = T2.CountryCode WHERE T2.Language = 'Mandarin Chinese')`
Predicted SQL	`SELECT COUNT(*) FROM countrylanguage JOIN country ON countrylanguage.countrycode = country.code WHERE countrylanguage.language = 'Karen'`
Generated text	Find the language of the country that has the head of state Salahuddin Abdul Aziz Shah Alhaj and is official.
Gold SQL	`SELECT T2.Language FROM country AS T1 JOIN countrylanguage AS T2 ON T1.Code = T2.CountryCode WHERE T1.HeadOfState = 'Salahuddin Abdul Aziz Shah Alhaj' AND T2.IsOfficial = 'T'`
Predicted SQL	`SELECT countrylanguage.language FROM countrylanguage JOIN country ON countrylanguage.countrycode = country.code WHERE country.headofstate = 'Salahuddin Abdul Aziz Shah Alhaj'`
Generated text	What are the names of countries with surface area greater than the smallest area of any country in Antarctica?
Gold SQL	`SELECT Name FROM country WHERE SurfaceArea > (SELECT min(SurfaceArea) FROM country WHERE Continent = 'Antarctica')`
Predicted SQL	`SELECT country.name FROM country WHERE country.surfacearea > (SELECT MAX(country.surfacearea) FROM country WHERE country.continent = 'Antarctica')`

Table A6: Consistent SQL-Text pairs rejected by cycle-consistency but retained by our filtering model. Predicted SQL is the output of the Text-to-SQL model used for checking cycle consistency, and does not match the gold SQL often due to minor errors.

A.4 Examples of masking

SQL	SELECT T1.template_type_code , count(*) FROM Templates AS T1 JOIN Documents AS T2 ON T1.template_id = T2.template_id GROUP BY T1.template_type_code [**Schema Name**: Document Template Management]
Reference	Show all template type codes and the number of documents using each type.
Retrieved SQL	T1.FacID , count(*) FROM Faculty AS T1 JOIN Student AS T2 ON T1.FacID = T2.advisor GROUP BY T1.FacID [**Schema Name**: Faculty Student Activity]
Retrieved Text	Show the faculty id of each faculty member, along with the number of students he or she advises.
Sch-match Mask	Show the MASK of each MASK member , along with the number of MASK he or she advises .
Filled Text	Show the type code of each template member, along with the number of documents he or she advises.
Freq Mask	Show the MASK of each MASK , MASK with the number of MASK he or she MASK .
Filled Text	Show the code of each template type, together with the number of documents corresponding to it.
SQL	SELECT T2.name , T2.capacity FROM concert AS T1 JOIN stadium AS T2 ON T1.stadium_id = T2.stadium_id WHERE T1.year >= 2014 GROUP BY T2.stadium_id ORDER BY count(*) DESC LIMIT 1 [**Schema Name**: Concert Singer]
Reference	Show the stadium name and capacity with most number of concerts in year 2014 or after.
Retrieved SQL	SELECT T2.name , T1.team_id_winner FROM postseason AS T1 JOIN team AS T2 ON T1.team_id_winner = T2.team_id_br WHERE T1.year = 2008 GROUP BY T1.team_id_winner ORDER BY count(*) DESC LIMIT 1 [**Schema Name**: Baseball 1]
Retrieved Text	What are the name and id of the team with the most victories in 2008 postseason?
Sch-match Mask	What are the MASK and MASK of the MASK with the most victories in MASK
Filled Text	What are the name and capacity of the stadium with the most victories in year 2014?
Freq Mask	What are the MASK and MASK of the MASK with the most MASK in MASK
Filled Text	What are the name and capacity of the stadium with the most concerts in 2014?

Table A7: Masking the text based on string matches Vs. our method of frequency based masking. Schema-relevant words like 'victories', 'members', 'advises' that do not have a sufficient string match with any of the table or column names of their schema, get left out when using string-match based matches. Thus failing to mask the words in the original schema might lead to copying of the word in the target schema, thus making the generated text semantically inconsistent. Words in blue are schema relevant words for the target database and should appear in the generated output.

A.5 Hyperparameters

Our Edit and Fill model (139.2M parameters) is based on a pretrained BART-BASE (Lewis et al., 2020) model. We fine-tune this model for 100 epochs with learning rate of 3×10^{-5}, weight decay of 0.01 and batch size of 64. The pretrained model is obtained from HuggingFace[3].

The proposed binary classifier (124.6M params) is pretrained ROBERTA-BASE (Liu et al., 2020) (obtained from HuggingFace[4]) fine-tuned for 100 epochs on our data with learning rate 10^{-5}, weight decay 0.01 and batch size 16 for 100 epochs.

For SMBOP experiments, we use a smaller SMBOP model with 4 RAT layers and ROBERTA-BASE (Liu et al., 2020) encoder as a baseline. The number of parameters in this model is 132.9M. All the adaptation experiments use learning rate of 5×10^{-6}, learning rate of language model of 3×10^{-6} and batch size of 8. All the models were trained for 100 epochs.

All the experiments were performed on a single NVIDIA RTX 3060 GPU. Training times for the Edit-and-Fill model and binary classifiers were $\approx$ 4.5 hrs and $\approx$ 6.5 hrs respectively. Each of the fine-tuning experiment took $3-4$ hrs to complete.

A.6 Cost function for Tree Edit Distance

Group	Value	Cost
Equal	Equal	0
Equal	Unequal	0.5
Unequal	Equal	0
Unequal	Unequal	1

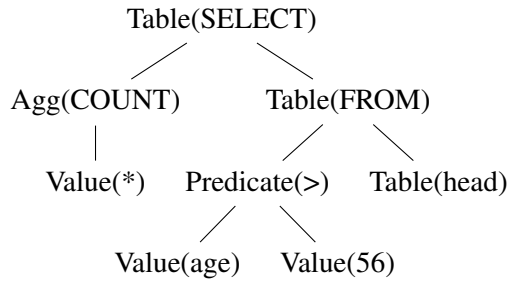
Table A8: Cost function of nodes n_1 and n_2 based on their groups and value.

Group	SQL elements
Aggregation	MAX, MIN, AVG, COUNT, SUM
Order	ORDERBY_ASC, ORDERBY_DESC
Boolean	OR, AND
Set	UNION, INTERSECT, EXCEPT
Leaf	VAL_LIST, VALUE, LITERAL, TABLE
Similarity	LIKE, IN, NOT_IN
Comparison	$>, \geq, <, \leq, =, \neq$

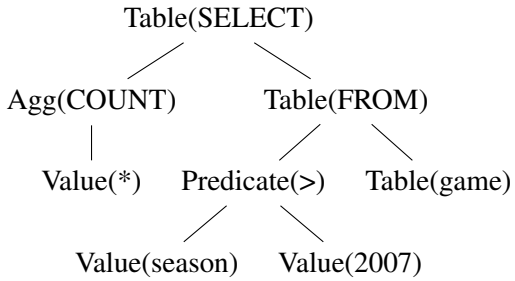
Table A9: Group definitions for TED calculation.

We use APTED library (Pawlik and Augsten, 2015, 2016) to compute TED between 2 parsed SQL trees. For every node in the tree, a group is assigned according to table A9. Then the cost for various combinations of node groups and node values is described in table A8. If either of the nodes does not belong to any of the groups in table A9, their groups are considered to be "unequal" and cost will be assigned based on their values.

A.7 Examples of TED neighbours

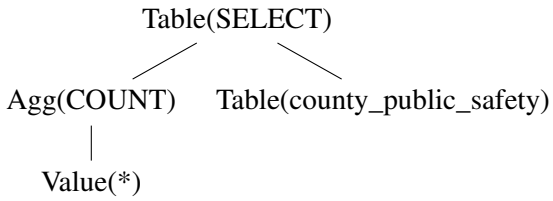


(a) `SELECT count(*) FROM head WHERE age > 56`

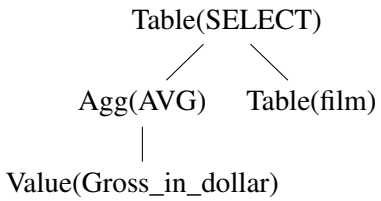


(b) `SELECT count(*) FROM game WHERE season > 2007`

Figure 5: Example of tree pair with TED=0



(a) `SELECT count(*) FROM county_public_safety`



(b) `SELECT avg(Gross_in_dollar) FROM film`

Figure 6: Example of tree pair with non-zero(=0.125) TED

[3] https://huggingface.co/facebook/bart-base
[4] https://huggingface.co/roberta-base