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

Anonymous ACL submission

Abstract

Serving novel schemas for semantic parsing of natural language queries over relational databases is a challenging problem owing to a huge diversity of schemas and zero availability of text queries in the target schema until the initial deployment of the parser in the real world. We present REFILL, a framework for synthesising diverse and high quality parallel data of Text-SQL pairs for adapting semantic parsing models on a new schema. Unlike prior approaches that synthesize text using an SQL-to-Text model trained on existing datasets, our approach uses a novel method of retrieving diverse existing text, masking their schema-specific tokens, and refilling to translate to the target schema. We show that this process leads to significantly more diverse text than achievable by sampling the beam of a plain SQL-to-Text model. Experiments across four groups of relational databases establish that finetuning a semantic parser on the datasets synthesized by REFILL offers consistent performance gains over prior data-augmentation methods.

1 Introduction

Natural Language interface to Databases (NLIDB) that translate textual queries to SQLs executable on a relational database is an ambitious goal in the field of Semantic Parsing. Unlike other semantic parsing tasks, Text-to-SQL also demands the ability to reason over the schema structure of a relational database, in addition to understanding the natural text and generating a syntactically correct structured output. 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-art neural Text-to-SQL models (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). Since database schemas are typically proprietary or private, generalizing over the unseen schema structure becomes even harder due to the lack of labeled training data. In general, adapting an existing semantic parser to a new schema requires significant amounts of labeled data for finetuning.

Lack of 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, state of the art data-generation methods are based on training an SQL-to-Text model using labeled data from pre-existing schemas, and generating in data in 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 for generating synthetic, diverse text for a given SQL workload that is often readily available (Baik et al., 2019). REFILL leverages the availability of parallel datasets such as Spider (Yu et al., 2018) from several existing schemas to first retrieve a diverse set of text paired with SQLs that are similar structurally to a given SQL $\mathbf{q}$. Then, it trains a novel *schema translator* model for converting the text of the training schema to the target schema of $\mathbf{q}$. 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 la-

beled data for the `fill` module entails non-trivial details that we explain in this paper. REFILL also incorporates a method of filtering high-quality text using an independent binary classifier, that provides more useful independent quality scores, than the cycle consistency scores used in (Zhong et al., 2020). Our approach is related to retrieve-and-edit models that have been used in other NLP tasks including dialogue generation (Chi et al., 2021), translation (Cai et al., 2021), Question Answering (Karpukhin et al., 2020), and Semantic Parsing (Hashimoto et al., 2018; Pasupat et al., 2021; Das et al., 2021). However, our method of casting the "edit" as a two-step mask-and-fill schema translation model is different from existing methods.

Our key contributions are as follows (i) We propose the idea of translating natural text from several existing schemas for synthesizing text for a target schema to get greater diversity than achievable by beam-sampling a SQL-to-Text model. (ii) We design strategies for masking schema specific words in the retrieved text, and training the REFILL model to translate to the target schema using existing single schema pairs. (iii) We present a method for filtering high quality parallel data using a binary classifier and show that it is more efficient than existing methods based on cycle consistency. (iv) We compare REFILL with existing conditional generation methods and show that our more diverse synthetic data yields significantly more accurate adaption of Text-to-SQL models to new database schemas.

2 Diverse parallel data synthesis with REFILL

Our goal is to generate synthetic parallel data to adapt a trained Text-to-SQL model to a new schema unseen during training. A Text-to-SQL model $\mathbf{M}$: $\mathcal{X}, \mathcal{S} \mapsto \mathcal{Q}$ maps a natural language question $\mathbf{x} \in \mathcal{X}$ addressed on a database schema $\mathbf{s} \in \mathcal{S}$, to an SQL query $\hat{\mathbf{q}} \in \mathcal{Q}$. We assume a Text-to-SQL model $\mathbf{M}$ trained on a dataset $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, \mathbf{s}_i, \mathbf{q}_i)\}_{i=1}^{N}$ consisting of text queries $\mathbf{x}_i$ addressed on a database schema $\mathbf{s}_i$, and the corresponding gold SQL queries $\mathbf{q}_i$. Our approach is agnostic to the exact model used for Text-to-SQL. The train set $\mathcal{D}_{\text{train}}$ consists of examples from a wide range of schemas $\mathbf{s}_i \in \mathcal{S}_{\text{train}}$, e.g. the Spider dataset (Yu et al., 2018) which contains roughly 140 schemas in the train set i.e. $|\mathcal{S}_{\text{train}}| = 140$. We focus on adapting a model

Algorithm 1: Data Synthesis with REFILL

input: $\mathcal{QW}_s$, $\mathbf{M}$, $\mathcal{D}_{\text{train}}$
$\mathcal{D}_{\text{syn}} \leftarrow \phi$
for $\mathbf{q} \leftarrow$ SampleSQLQueries $(\mathcal{QW}_s)$ **do**
  $\{\mathbf{q}_r, \mathbf{x}_r\} \leftarrow$ **RetrieveRelatedPairs**$(\mathbf{q}, \mathcal{D}_{\text{train}})$
  $\{\mathbf{x}_r^{\text{masked}}\} \leftarrow$ **MaskSchemaTokens**$(\{\mathbf{q}_r, \mathbf{x}_r\})$
  $\{\mathbf{x}_r^{q}\} \leftarrow$ **EditAndFill**$(\{\mathbf{q}, \mathbf{x}_r^{\text{masked}}\})$
  $\mathcal{D}_{\text{syn}} \leftarrow \mathcal{D}_{\text{syn}} \cup$ **Filter**$(\mathbf{q}, \{\mathbf{x}_r^{q}\})$
$\mathbf{M}_{\text{new}} \leftarrow$ Finetune$(\mathbf{M}, \mathcal{D}_{\text{syn}})$

$\mathbf{M}$ trained on $\mathcal{D}_{\text{train}}$ to perform well on queries from a new schema $\mathbf{s}$ different from the schemas in $\mathcal{S}_{\text{train}}$. We propose to generate diverse parallel data $\mathcal{D}_{\text{syn}}$ using which we fine-tune the pre-trained model $\mathbf{M}$ to the new schema $\mathbf{s}$. We assume that on the new schema $\mathbf{s}$ 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).

Figure 1 and Algorithm 1 summarizes our method for synthesizing diverse SQL-Text pairs for adapting an existing Text-to-SQL semantic parsing model $\mathbf{M}$ to a target database $\mathbf{s} \in \mathcal{S}_{\text{target}}$ not seen during training $\mathbf{s} \notin \mathcal{S}_{\text{train}}$. Given a SQL query $\mathbf{q}$ on the target schema $\mathbf{s}$, our method first retrieves related SQL-Text pairs $\{\mathbf{q}_r, \mathbf{x}_r\}_{r=1}^{R}$ from the $\mathcal{D}_{\text{train}}$ on the basis of a tree-edit-distance measure such that the SQLs $\{\mathbf{q}_r\}_{r=1}^{R}$ in the retrieved pairs have similar structure as the given SQL query $\mathbf{q}$. We then translate each text $\mathbf{x}_r$ so its target query changes from $\mathbf{q}_r$ to $\mathbf{q}$ on schema $\mathbf{s}$. We decompose this task into two steps: mask out schema specific tokens in $\mathbf{x}_r$, and fill the masked text to represent $\mathbf{q}$ with the help of a conditional text generation model $\mathbf{B}$ like BART (Lewis et al., 2020). The translated text may be noisy since we do not have direct supervision to train such models. To improve the overall quality of the synthesized data we filter out the unlikely SQL-Text pairs with the help of an independent binary classifier. Finally, we adapt the given Text-to-SQL model $\mathbf{M}$ for the target database by fine-tuning it on the diverse, high-quality filtered data $\mathcal{D}_{\text{syn}}$ synthesized by our method. We now describe each step in further detail.

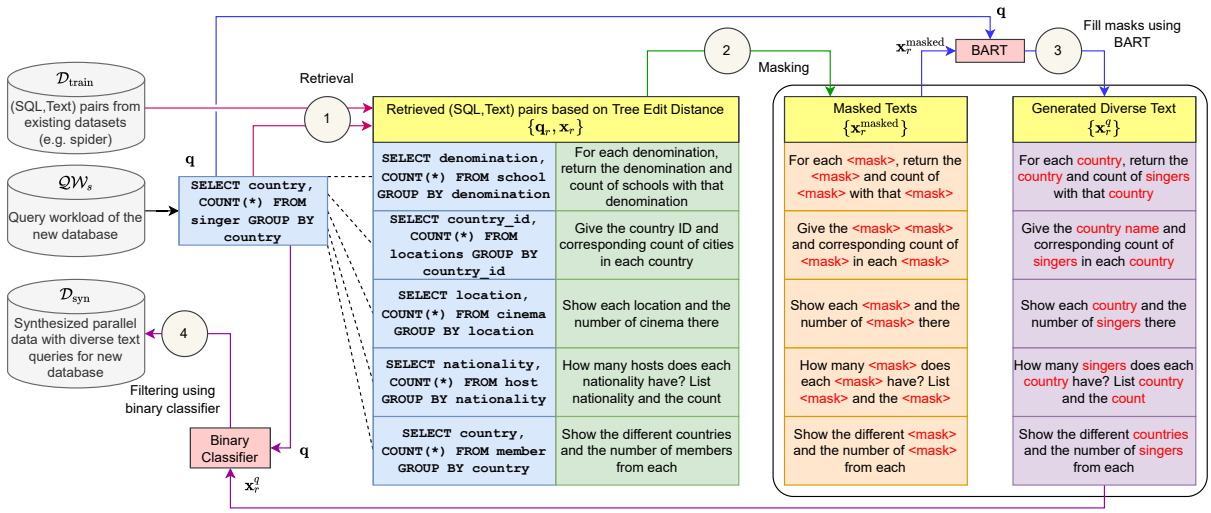


Figure 1: Diverse parallel data synthesis by editing related examples using REFILL. Given a query $\mathbf{q}$ from a new database, REFILL **(1)** Retrieves SQL-Text pairs from an existing dataset where the SQLs have a small edit distance w.r.t. the query $\mathbf{q}$ (indicated by dashed lines in the diagram). **(2)** Since the retrieved text come from a different database, the schema specific words are masked out. **(3)** The masked text and the query $\mathbf{q}$ are then translated into the target schema via an Edit and Fill step that uses a conditional text generation model like BART. **(4)** Finally, the synthesized SQL-Text pairs are filtered using a binary classifier model that is trained to retain only the consistent SQL-Text pairs. Translating the text from multiple related examples allows REFILL to generate diverse and high quality text for the new schemas.

2.1 Retrieving related queries

Given a query $\mathbf{q} \in \mathcal{QW}_s$ sampled from the workload, we extract the query-text pairs $\{\mathbf{q}_r, \mathbf{x}_r\} \in \mathcal{D}_{\text{train}}$ from the train set such that the retrieved queries $\{\mathbf{q}_r\}$ are similar in structure of the query $\mathbf{q}$. We utilize tree-edit-distance (Pawlik and Augsten, 2015, 2016) between the relational algebra trees corresponding to the queries $\mathbf{q}$ and $\mathbf{q}_r$. Since the retrieved queries come from a different schema, we modify the tree edit distance algorithm to ignore the schema names and the database values. The tree-edit-distance is further normalized by size of the larger tree. We only consider the pairs with queries having a distance of less than 0.1 w.r.t. the query $\mathbf{q}$. On existing datasets like Spider, it is often possible to find several SQLs structurally similar to a $\mathbf{q}$. For example, in Spider we found that 76% of test SQLs contain at least three SQLs in $\mathcal{D}_{\text{train}}$ that are structurally identical, that is, have a tree-edit-distance of 0. Figure 2 shows more detailed statistics.

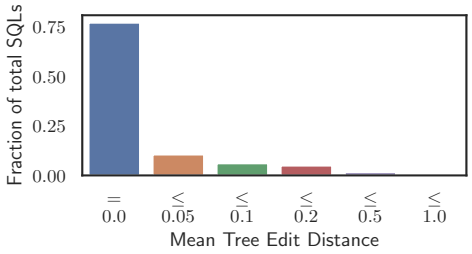


Figure 2: Frequency distribution of average tree-edit-distance of test-queries in Spider to its top-3 structurally similar queries in the training set.

2.2 Translating text of related queries

Our next goal is to translate $\mathbf{x}_r$ from being a query on $\mathbf{q}_r$ on DB-schema $\mathbf{s}_r$ to a text for query $\mathbf{q}$ on schema $\mathbf{s}$ where $\mathbf{q} \approx \mathbf{q}_r$ structurally. We cannot train a direct translation model with $(\mathbf{x}_r, \mathbf{q})$ as input since we do not have any parallel labeled data for this new type of translation task. We therefore decompose this into two steps: 1) a simpler task of masking schema-specific tokens in $\mathbf{x}_r$ to get a template $\mathbf{x}_r^{\text{masked}}$, and 2) a conditional text generation model that maps $(\mathbf{x}_r^{\text{masked}}, \mathbf{q})$ to the target text for which we modify $\mathcal{D}_{\text{train}}$ to get indirect supervision. We describe these steps next:

Masking retrieved text Converting retrieved text queries to masked templates is a critical component of REFILL's pipeline since irrelevant tokens e.g. references to schema elements of the original database, can potentially misguide the text generation. Our initial approach was to mask to-

kens based on 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 occurrences in natural text often differed significantly from schema names in the database. Table 9 shows some anecdotes. Consequently, we designed a simple frequency-based method of masking that was significantly more effective for our goal of using the masked text to just guide the diversity. For each word that appears in the questions of the train set, we count the number of distinct databases for which that word appears at least once in one of the text questions for that database. E.g. words like {'show', 'what', 'list', 'order'} appear in more than 90% of schemas, and schema specific words like {'countries', 'government'} occur only in queries of a few schemas. We mask out all the words that appear in less than 50% of schema. The words to be masked are replaced by a special token MASK. Consecutive occurrences of MASK are collapsed into a single MASK token. Thus we obtain masked templates $\{\mathbf{x}_r^{\text{masked}}\}$ retaining minimal information about their original schema.

Editing and Filling the masked text Given a masked template $\mathbf{x}_r^{\text{masked}}$, and an SQL query $\mathbf{q}$ that needs to be translated into a text query $\hat{\mathbf{x}}$, we first convert $\mathbf{q}$ into a pseudo-English representation $\mathbf{q}^{\text{Eng}}$ similar to the one described in (Shu et al., 2021). In addition, we wrap the table, column, or value tokens in $\mathbf{q}$ with special tokens to provide explicit signals to the text generation module that such tokens are likely to appear in the generated text. Next, we concatenate the tokens in the masked text $\mathbf{x}_r^{\text{masked}}$ and the query $\mathbf{q}^{\text{Eng}}$ for jointly encoding them as an input to a conditional text generation model like BART. The output of the decoder is expected to be natural language text $\hat{\mathbf{x}}$ consistent with the query $\mathbf{q}$. Since we do not have direct supervision for such training, we transform $\mathcal{D}_{\text{train}}$ to generate parallel data for this training as follows:

Given a training dataset $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, \mathbf{s}_i, \mathbf{q}_i)\}_{i=1}^N$ of Text-SQL pairs $(\mathbf{x}_i, \mathbf{q}_i)$ for different schemas $\mathbf{s}_i \in \mathcal{S}_{\text{train}}$, the conditional text generation model is now finetuned for translating $\{\mathbf{x}_i^{\text{masked}}, \mathbf{q}_i^{\text{Eng}}\}$ to $\mathbf{x}_i$ as follows. **(a)** For one-third of random train steps we provide $[\mathbf{x}_i^{\text{masked}}|\mathbf{q}_i^{\text{Eng}}]$, the concatenation of the masked text and the $\mathbf{q}_i^{\text{Eng}}$ as an input to the encoder and maximize the likelihood of $\mathbf{x}_i$ in the decoder's output. **(b)** For another one-third we pass only $\mathbf{q}_i^{\text{Eng}}$ as an input maximize the likelihood of $\mathbf{x}_i$. This ensures that model is capable of generating the text from the query alone, if the templates are unavailable or noisy. **(c)** For the last one-third, we use masked templates $\mathbf{x}_j^{\text{masked}}$, across two different schemas $\mathbf{s}_i$ and $\mathbf{s}_j$, such that the tree-edit distance between the queries $\mathbf{q}_i$ and $\mathbf{q}_j$ is small, and the word edit distance between the masked templates $\mathbf{x}_i^{\text{masked}}$ and $\mathbf{x}_j^{\text{masked}}$ is also small. This makes the training more consistent with the inference, where the schemas are different. In Section 4.4, we establish the importance of steps **(b)** and **(c)** for generating text that is more consistent with the SQL queries (See Table 3).

2.3 Filtering Generated Text

Since the data synthesized using REFILL is used to finetune the semantic parsing models in the downstream, we learn a Filtering model $\mathbf{F} : (\mathcal{X}, \mathcal{Q}) \mapsto \mathbb{R}$ that assigns lower scores to inconsistent SQL-Text pairs and higher scores to the consistent ones. We select the top-5 sentences for each query generated by REFILL and reject all the sentences that are scored below a fixed threshold as per the Filtering model. Existing work depended on the trained Text-to-SQL $\mathbf{M}$ to assign quality scores, however we found that such filtering did not result in a useful dataset for fine-tuning $\mathbf{M}$ since it favored text on which $\mathbf{M}$ was already good.

We instead train an independent binary classification model for filtering as follows: The SQL-Text pairs in the training set $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, \mathbf{s}_i, \mathbf{q}_i)\}_{i=1}^N$ serve as the positive (consistent) examples and we synthetically generate the negative (inconsistent pairs) as follows: (i) Replace DB values in the SQL with arbitrary values sampled from the same column of the database. (ii) Replace SQL-specific tokens with their corresponding alternates e.g. replace ASC with DESC, or '>' with '<'. (iii) Cascade previous two perturbations. (iv) Replace the entire SQL with a randomly chosen SQL from the same schema. (v) Randomly drop tokens in the text query with a fixed probability of 0.3. (vi) Shuffle a span of tokens in the text query, with span length set to 30% of the length of the text query. Thus for a given Text-SQL pair $(\mathbf{x}, \mathbf{q})$ we obtain six corresponding negative pairs $\{(\mathbf{x}_i^n, \mathbf{q}_i^n)\}_{i=1}^6$. Let s be the score provided by the filtering model for the original pair $(\mathbf{x}, \mathbf{q})$ and $\{s_i\}_{i=1}^6$ be the scores assigned to the corresponding negative pairs

$\{(\mathbf{x}_i^n, \mathbf{q}_i^n)\}_{i=1}^6$. We supervise the scores from the filtering model using a binary-cross-entropy loss over the Sigmoid activations of scores as in Equation 1.

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

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

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

3 Related Work

SQL-to-Text generation A large body of prior work performs training data augmentation via pre-trained conditional text generation modelsthat 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) finetune 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 in Section 4.5. The SQL workload in these 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 the context of various NLP tasks. In Semantic Parsing, question and logical-form pairs from the training data relevant to the 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, Translation-memory augmented methods like (Hossain et al., 2020; Cai et al., 2021) retrieves and edit examples from translation memory to guide the decoder's outputs. Our editing step masking followed by refilling is somewhat similar to style transfer methods like (Li et al., 2018) that minimally modify the input sentence with help of retrieved examples corresponding to the target attribute. In contrast to a learned retrieval, we find simple tree-edit distance based retrieval to be highly effective for retrieving the relevant examples for our task.

4 Experiments

We demonstrate the effectiveness of the data synthesized using REFILL for adapting base semantic parsing models to new groups of databases in Section 4.1. We compare with the recent and competitive baselines that utilize SQL-to-Text generation methods for improving the performance of semantic parsers via training-data augmentation (Wang et al., 2021; Zhong et al., 2020). We also evaluate the intrinsic quality of the generated synthetic data in-terms of diversity and agreement with gold text queries in the test data. In Section 4.2 we compare the quality and the diversity of the text generated using REFILL with the relevant SQL-to-Text baselines. Section 4.4 justifies the key design choices related to masking and the training of schema translator module, that helps REFILL synthesize high quality text. Section 4.5 demonstrates the importance of using an independent binary classifier over cycle-consistency filtering.

4.1 Experimental Setup

Datasets: We create 4 Groups of databases chosen from Spider's dev-set. The databases within each group have a similar topic. E.g. Group-1 consists of databases `{Singer, Orchestra, Concerts}`. We utilize all the available Text-SQL pairs in each group for evaluation. 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 constants-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 an SQL-to-Text model, and the resulting Text-SQL pairs are then

used to finetune a base semantic parsing model.

Base Semantic Parsers: We experiment with SmBoP (Rubin and Berant, 2021) as our base Text-to-SQL semantic parser, and utilize author's implementation for our experiments. 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 while training excludes the data from the four groups used for evaluation.

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 finetuned using the train split of Spider dataset as described in Section 2.2

Filtering Model: We utilize a pre-trained ROBERTA-base for learning a binary classifier to filter out inconsistent SQL-Text pairs as described in Section 2.3. The model is trained using the train split of Spider dataset.

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 feeding SQL tokens as an input to the models. Section 3 provides more details about the baselines.

Evaluation Metrics Following the prior work, we evaluate the Text-to-SQL parsers using the Exact Set Match (EM), and the Exection Accuracy (EX), as proposed in Yu et al. (2018). The EM metric measures set match for all the SQL clauses and returns one 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 $\hat{\mathbf{q}}$ and the gold query $\mathbf{q}$ on the database.

Additional implementation details including the hyperparameters are reported in the Appendix A.5

4.2 Main Results

Evaluating finetuned parsers Table 1 presents results for finetuning the base Text-to-SQL model on the SQL-Text pairs obtained by translating the SQL workload using various SQL-to-Text generation models. Compared to prior methods for SQL-to-Text generation that lack both in the diversity and the quality of the generated text, finetuning over the 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% by the best baseline (GAZP). The gains from the baseline methods are often small or even negative. Our gains over baselines continue even for other settings of workload sizes. Figure 3 plots size of the workload on the x-axis vs. the EM of the finetuned 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 query workload. On the other hand, the baseline SQL-to-Text generation methods fail to provide significant improvements. Interestingly, the data synthesized by ReFill for the 30% query workload is more effective on average than any of the baselines utilizing the 70% query workload for SQL-to-Text generation.

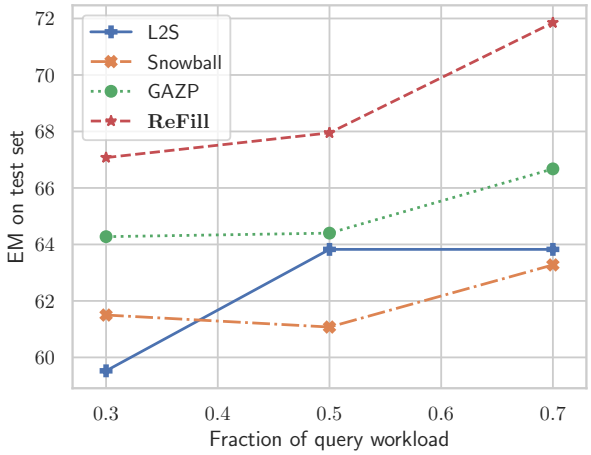


Figure 3: Average EM performance of Text-to-SQL models on the four groups vs. the size of query workload. The data generated by ReFill using 30% query workload yields better performance than the data from the best baseline on 70% workload.

Intrinsic quality and diversity of generated text We explain our gains over existing methods to the increased quality and diversity of the generated text. We measure quality by reporting the BLEU score of the set $S(\mathbf{q})$ of generated text for a SQL $\mathbf{q}$ with the gold text of the query in the test data. To measure diversity we report 1-SelfBLEU (Zhu et al., 2018) that measures the average BLEU score among text in $S(\mathbf{q})$. We evaluate on all the gold

SmBoP	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 SQL-Text pairs generated various SQL-to-Text baselines and ReFill, as described in Section 4.2. ReFill provides consistent gains over the base model across all the database groups, while gains from other methods are often negative or small.

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. Gold-Ref represents the scores corresponding to gold-references as outputs

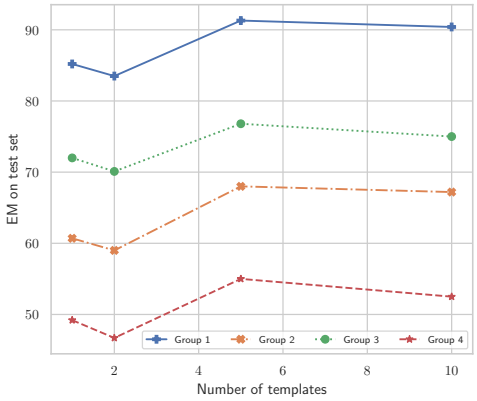


Figure 4: EM performance of finetuned SQL-to-Text models Vs. the number of templates per SQL used by ReFill

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

Table 3: Analyzing impact of design choices related to Schema Translation, by observing BLEU-4 scores of the text generated by ReFill

SQL-Text pairs available in the Spider's dev set. Table 2 reports the results. For each model we 10 hypotheses per SQL query, and pick the hypothesis with the highest BLEU to report the overall BLEU scores. To allow baselines for generating more diverse outputs than the standard beam search, we utilize beam-sampling (Fan et al., 2018; Holtzman et al., 2019). For ReFill-10, the 10 hypothesis come from using upto 10 retrieved templates. ReFill generates both diverse and high-quality hypotheses. We observe that leveraging text from other schemas allows ReFill to generate higher quality text (+9.8 BLEU points), while simultaneously enabling higher diversity.

4.3 Importance of Text Diversity

Utilizing the retrieved text templates from multiple schemas allows ReFill to generate diverse text. Figure 4 justifies the importance of diversity in the generated text for improved performance, by varying the number of templates on the x-axis and performance of the finetuned models on y-axis for each group. To keep the number of synthesized examples same, the product of beam-samples and the number of templates is held constant. Utilizing the more diverse data generated via 5 templates is consistently superior than using less diverse data obtained by using lesser templates. As indicated by the Self-BLEU scores, the diversity of the data generated via 10 templates is also low, due to the repetition of similar templates. Drop in ReFill's performance with lesser templates reconfirms the worse performance observed with the reported SQL-to-Text baselines that do not offer textual diversity.

4.4 Design choices of Schema Translator

Section 2.2 described two important design choices: (1) Method of masking schema-relevant tokens and (2) Method of training the Edit-and-Fill model for generating text. Table 3 justifies these design choices. Comparing across rows (Schema-Match Vs Frequency), we observe that Frequency based

	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 finetuned Text-to-SQL models

masking results in 2 to 3 point improvements in the BLEU scores compared to matching schema names. Table 9 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 (Naive 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.

4.5 Importance of Filtering model

Methods like GAZP (Zhong et al., 2020) utilize consistency-based filtering to reject the synthesized SQL-Text pairs $(\mathbf{q}, \mathbf{x})$ that are inconsistent with the output $\hat{\mathbf{q}}$ produced by the base Text-to-SQL for the text query $\mathbf{x}$. We argue that cycle-consistency based filtering is sub-optimal for two reasons: (i) **Data Redundancy**: Since the Text-to-SQL model is already capable of generating the correct output for the retained examples, these samples do not offer much improvements while training. (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.
As a solution, we train a Filtering model described in Section 2.3. The filtering is now independent of the base semantic parser, thus capable of retaining the high quality generated examples which might otherwise be filtered out by cycle-consistency filtering using a weak Text-to-SQL model. Table 4 compares the base Text-to-SQL model, with models finetuned without any filtering, with cycle-consistent filtering, and with using the filtering model. We focus on Group-4 where the base Text-to-SQL model is significantly weaker in comparison to other groups, and use REFILL to synthesize data for the 30% query workload. Not using any filtering, or using cycle-consist filtering result in worse performance, while using our filtering model offers significant improvements over the base model. Table 8 provides anecdotes of potentially useful training examples that were filtered out by the cycle-consistency, but retained by our filtering model.

5 Conclusion and Future Work

We presented REFILL, a framework for generating diverse and high quality parallel data for adapting Text-to-SQL models to a target database. REFILL translates a given SQL query into diverse questions by retrieving and editing examples in the other schemas and using a masking and refill mechanism. Our experiments show that REFILL generates higher quality and more diverse text which are key to better performance of the downstream semantic parsers finetuned on the data generated by REFILL. Our experiments show that even in the lowest query workload, the proposed framework outperforms the best baseline with highest query workload. However, currently the experiments are only performed on academic datasets such as Spider. We hope to explore applications of our work on real-world databases in the future.

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.

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, Online. 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, Online. 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)*, pages 6769–6781.

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, Online. 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, Online. 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*, pages 7871–7880.

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.

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*, pages 7683–7698.

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*, pages 311–324.

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*, pages 8372–8388.

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, Online. 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*, pages 3911–3921.

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, Online. 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 Grouping Details

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

Table 5: Group statistics for every group. Group 4 contains only 1 schema since the world_1 schema is large enough. On other groups, we collect together 2-3 similar schema to get a larger workload as shown.

A.2 Results in low or medium overlap setting

SmBoP	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	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	61.7	62.0
ReFill	86.1	86.1	65.6	65.6	68.3	67.1	48.3	36.7	67.1	63.8

Table 6: Results on SmBoP under 0.3 overlap setting

SmBoP	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	89.6	88.7	66.1	68.5	57.9	58.5	41.7	35.8	63.8	62.9
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

Table 7: Results on SmBoP under 0.5 overlap setting

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 8: Useful training examples 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 9: 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[1].

The proposed binary classifier (124.6M params) is pretrained ROBERTA-BASE (Liu et al., 2020) (obtained from HuggingFace[2]) finetuned 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 NVIDIA RTX 3060 GPU. Training times for template filling model and binary classifiers were $\approx$ 4.5 hrs and $\approx$ 6.5 hrs respectively. Each of the finetuning 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 10: Cost function of nodes n_1 and n_2 based on their groups and value

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 11. Then the cost for various combinations of node groups and node values is described in table 10. If either of the nodes does not belong to any of the groups in table 11, their groups are considered to be "unequal" and cost will be assigned based on their values.

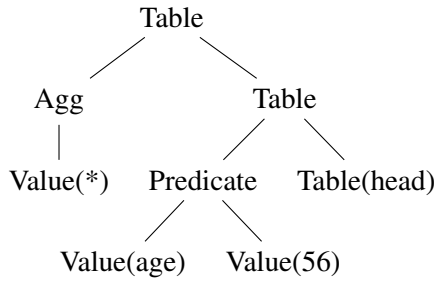
A.7 Examples of TED neighbours

[1] https://huggingface.co/facebook/bart-base

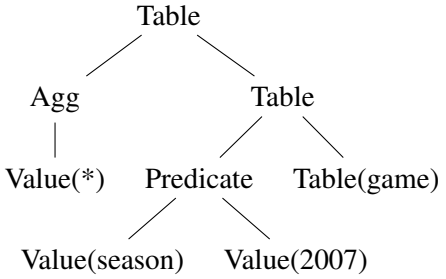
[2] https://huggingface.co/roberta-base

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 11: Group definitions for TED calculation

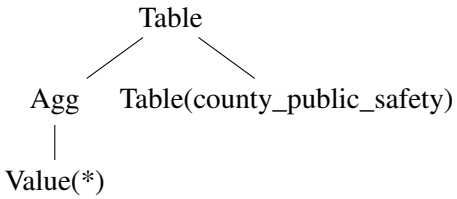


(a) `SELECT count(*) FROM head WHERE age > 56`

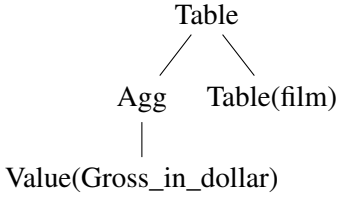


(b) `SELECT count(*) FROM game WHERE season > 2007`

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 TED