Bridging the Generalization Gap in Text-to-SQL Parsing with Schema Expansion

Chen Zhao*
New York University
cz1285@nyu.edu

Yu Su
Microsoft Semantic Machines
yusu2@microsoft.com

Adam Pauls
Microsoft Semantic Machines
adam.pauls@microsoft.com

Emmanouil Antonios Platanios
Microsoft Semantic Machines
anthony.platanios@microsoft.com

Abstract

Text-to-SQL parsers map natural language questions to programs that are executable over tables to generate answers, and are typically evaluated on large-scale datasets like SPIDER (Yu et al., 2018). We argue that existing benchmarks fail to capture a certain *out-of-domain generalization* problem that is of significant practical importance: matching domain specific phrases to composite operations over columns. To study this problem, we propose a synthetic dataset and a re-purposed train/test split of the SQUALL dataset (Shi et al., 2020) as new benchmarks to quantify domain generalization over column operations. Our results indicate that existing state-of-the-art parsers struggle in these benchmarks. We propose to address this problem by incorporating prior domain knowledge by preprocessing table schemas, and design a method that consists of two components: *schema expansion* and *schema pruning*. This method can be easily applied to multiple existing base parsers, and we show that it significantly outperforms baseline parsers on this domain generalization problem, boosting the underlying parsers' overall performance by up to 13.8% relative accuracy gain (5.1% absolute) on the new SQUALL data split.

1 Introduction

Text-to-SQL parsing is the task of translating natural language questions over provided tables to SQL queries which can be executed to produce answers. In recent years, with the availability of large-scale datasets (e.g., Zhong et al., 2017; Yu et al., 2018), neural semantic parsers have witnessed significant success on this task. However, recent work (Suhr et al., 2020; Lee et al., 2021) has suggested that these state-of-the-art parsers are far from successful in terms of *out-of-domain generalization* in real scenarios, where users may ask questions related to potentially very large tables with the goal of improving their productivity (e.g., while they are viewing or editing a large Excel spreadsheet).

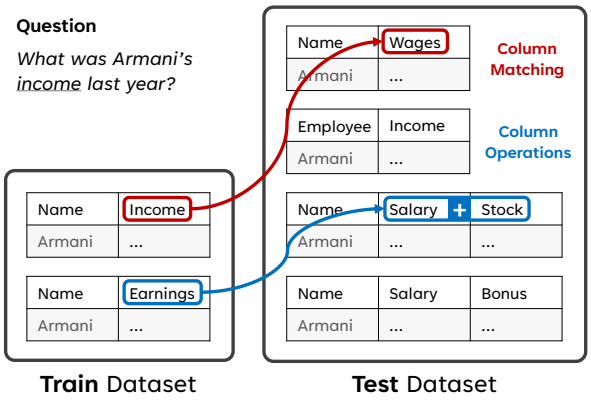

Figure 1: Illustration of two aspects of out-of-domain generalization that are challenging for text-to-SQL parsers. While existing methods partially address the "column matching" issue, they still suffer when it comes to "column operations". Note that there are more tables on the right to illustrate the fact that there are a variety of settings the parser may run into at test time.

In such scenarios, it is common to encounter tables specific to new domains that were not encountered while training a parser. Perhaps the most challenging aspect of domain generalization is that models need to understand domain-specific phrases that they have not seen before, and translate them into logical form segments that involve references to table elements (e.g., column names or aggregation operations over columns). We argue that two kinds of abstract operations, shown in Figure 1, are particularly challenging for new domains:

1. Column Matching: The task of mapping natural language phrases to the most relevant columns (e.g., mapping "Income" to the `"Wages"` column). This can be challenging because some mappings may be implicit or may require domain knowledge.
2. Column Operations: The task of mapping natural language phrases to composite expressions over table columns. For example, in Figure 1, we need to map *income* to just `"Wages"` for one table, and to `"Salary" + "Stock"` for another table. Similarly, consider the complex `"Term"` column in Figure 2, in which two subfields `_1` and `_2` represent the term start (e.g., 1926) and term end (e.g., 1927), respectively. Some questions may ask about the *term duration* while others may ask about the *term start*.

* This work was performed during a research internship at Microsoft Semantic Machines.

Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 5568 - 5578
May 22-27, 2022

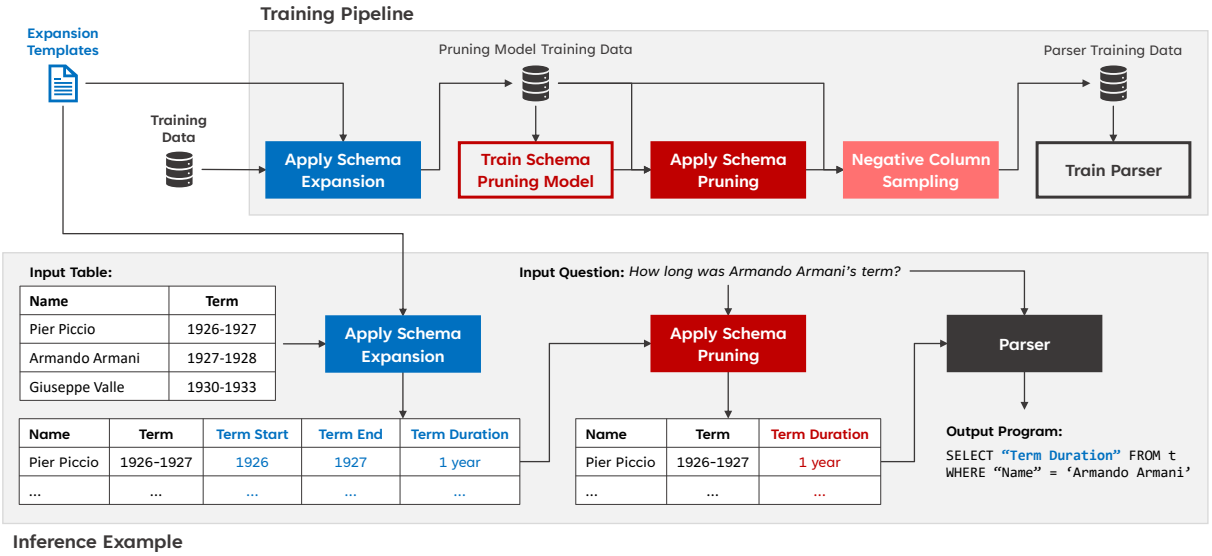

Figure 2: Illustration of the training pipeline for the proposed method and the inference process for an example. The proposed method is described in detail in §4. Note that the proposed components interact with the parser by modifying the table that is fed to it as input, as well as the target program during training in the case of *schema expansion*.

Each of these questions requires mapping the corresponding phrase to an expression that refers to this column (e.g., `"Term"._2 - "Term"._1` for the former and `"Term"._1` for the latter).

While recent approaches rely on pre-trained language models (e.g., Yin et al., 2020; Deng et al., 2021) for addressing the column matching challenge, column operations remain relatively unexplored due to the lack of evaluation benchmarks.

To this end, we first propose two new benchmarks: a synthetic dataset and a train/test repartitioning of the SQUALL dataset (Shi et al., 2020); both capable of quantifying out-of-domain generalization on column operations. We then show that existing neural parsers underperform on both benchmarks because they require an impractically large amount of in-domain training data—which is not available in our setting—to effectively "memorize" mappings from natural language phrases to program fragments. Finally, we propose a new method for making any existing text-to-SQL parsers aware of prior information that may be available about the domains of interest. Specifically, we propose two new components: *schema expansion* and *schema pruning*.

Schema expansion uses heuristics to expand columns into sets of derived columns based solely on their *types* (all schemas are assumed to be typed which tends to be true for both relational databases and Excel spreadsheets in practice; Excel uses a built-in type inference mechanism). Relying on generic types makes this method applicable to new domains, as long as they make use of similar underlying types. This process allows us to transform complex program fragments (e.g., `"Term"._2 - "Term"._1`) into simpler ones (e.g., `"Term Duration"`) that are better aligned with the natural language questions, thus making the underlying parser's job easier. While schema expansion may result in a large number of unnecessary expanded columns, schema pruning then examines both the input question and the available columns (original and expanded) and prunes the set of columns that the final parser is exposed to.

Our experiments show that schema expansion and schema pruning can boost the underlying parsers' performance by up to 13.8% relative accuracy (5.1% absolute) on the new SQUALL data split. Furthermore, they also boost performance over the original SQUALL data splits by up to 4.2% relative (1.9% absolute). One of our main goals in this paper is to put attention on the difficult problem of domain generalization by providing a new evaluation benchmark, as well as an initial direction for solving this problem. Our evaluation benchmarks along with code for reproducing our experiments are available at `https://aka.ms/text-to-sql-schema-expansion-generalization`.

2 Background

Task. Semantic parsing has been widely studied in the context of multiple other tasks like instruction following (Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013), code generation (Oda et al., 2015; Iyer et al., 2018), knowledge graph question answering (Berant et al., 2013; Yih et al., 2015), etc. We focus on using tables as the context in which semantic parsing is performed, where the goal is to translate pairs of natural language questions and tables to executable SQL queries, also known as text-to-SQL parsing (Androutsopoulos et al., 1995; Minock et al., 2008). Note that, while we focus on questions in the English language, there exists prior work on multilingual semantic parsing as well (Jie and Lu, 2014; Sherborne et al., 2020) and the contributions of our work also apply there. Formally, our goal is to map a pair (q, T), where q is a natural language question and T is a table, to an executable program π that,

when executed against table T, will produce the answer α to question q. We focus on the fully-supervised setting where the target executable program π^* is provided as supervision for training our parser.

Out-of-Domain Generalization. Generalization in machine learning is often defined as the ability to do well on a test set after learning from a training set, where all examples in both sets are drawn independently from the same distribution (*i.i.d. generalization*). However, as Gu et al. (2021) argue, in real-world applications such as semantic parsing, the test data may involve new compositional structures (*compositional generalization*), or new domains (*domain generalization*) that are not encountered during training. Existing work in compositional generalization for semantic parsing has focused on using synthetic datasets (e.g., Keysers et al., 2020; Lake and Baroni, 2018), or repartitioning existing text-to-SQL datasets into new train and test splits (e.g., Finegan-Dollak et al., 2018). Both approaches have generally shown that compositional generalization remains an important challenge (e.g., Shaw et al., 2021). We focus on the arguably even more challenging domain generalization problem, also known as *domain adaptation*, where entire domains may never be encountered during training or may only be encountered a small number of times (Motiian et al., 2017). Even though this problem has been studied extensively in the context of classification (Daumé III and Marcu, 2006), machine translation (Daumé III and Jagarlamudi, 2011), and question answering (Talmor and Berant, 2019), it remains underexplored for semantic parsing. To be applicable in real scenarios, semantic parsers must be able to generalize to new domains since collecting domain-specific labeled data is often prohibitively expensive. Recent approaches have focused on data synthesis (Yin et al., 2021), meta-learning (Wang et al., 2021), relation-aware schema encoding (Wang et al., 2020), and encoder pretraining (Yin et al., 2020; Herzig et al., 2020; Yu et al., 2020; Deng et al., 2021). In this paper, we hone in on one aspect of domain generalization that we shall broadly refer to as *column operations* and which was introduced in §1 and illustrated in Figure 1.

Evaluation Benchmarks. Text-to-SQL parsing became popular after the introduction of large-scale datasets and evaluation benchmarks. Zhong et al. (2017) first introduced WIKISQL, which contains Wikipedia tables paired with questions and annotated with SQL queries, albeit the queries are generated from a limited set of templates. SPIDER was introduced by Yu et al. (2018) the following year. It contains more complex questions and SQL queries and focuses on generalizing to previously unseen database schemas, but the dataset has the artifact from its annotation design that the references columns are often mentioned verbatim in the natural language questions. Deng et al. (2021) attempt to address this limitation by repartitioning SPIDER to produce a more realistic benchmark, and Lee et al. (2021) propose a challenging test set from Kaggle for evaluating parsers trained on SPIDER dataset. However, as Suhr et al. (2020) point out, SPIDER also uses a simplified setting which excludes examples that involve multiple columns (e.g., adding two columns together), as well as ones that require background knowledge. These benchmarks are thus limited in their usefulness for evaluating parsers in real-world settings where they may encounter complex questions that require mapping specific phrases to expressions over table columns, rather than to a single column. Furthermore, while both WIKISQL and SPIDER assume "simple" tables with only `String`- or `Number`-valued columns, in practice we may encounter tables where the columns themselves may have structured types (e.g., `TimeSpan`). For example, consider the table shown on the top left of Figure 2. In this case, the `"Term"` column is of type `TimeSpan` and consists of two `Numbers` that represent the beginning and the end of the timespan. In this case, users may ask questions that require constructing expressions to access nested elements from the `"Term"` column (e.g., "How long was Pier's term?"). Recently, Shi et al. (2020) introduced SQUALL, a dataset that annotates WIKITABLEQUESTIONS (Pasupat and Liang, 2015) with SQL queries and refined column types like `Date`, `Score`, `(T1, T2)`, and `List[T]`. However, SQUALL distributes tables evenly between the train and test splits, thus not allowing us to evaluate the kind of out-of-domain generalization we are interested in. Therefore as we will show in the following section, we aim to address this limitation by repartitioning SQUALL into new train and test splits.

Neural Text-to-SQL Parsers. Neural encoder-decoder models have recently gained popularity for text-to-SQL parsing (e.g., Xu et al., 2017). We focus on two models that represent the current state-of-the-art for SQUALL and SPIDER, respectively: SEQ2SEQ of Shi et al. (2020) [1] and SMBOP of Rubin and Berant (2021). Both models concatenate the question with a textual representation of the table schema, separated by a special `[SEP]` token, and feed the combined sequence to a pre-trained instance of the BERT (Devlin et al., 2019) language model. The activations of the last layer represent the encoded representations of the question and the table schema. SEQ2SEQ then uses a *autoregressive decoder*, which represents programs as token sequences and at each decoding step it: (1) predicts the next token type (i.e., whether the next token is a SQL keyword, a column name, or a literal value), and (2) predicts the token conditioned on its type. SMBOP, on the other hand, uses *bottom-up decoding*, which represents programs as abstract syntax trees and constructs these trees in a bottom-up fashion (i.e., it starts by predicting the leaf nodes and then recursively composes generated sub-trees into new trees and ranks them, in a way that resembles beam search), until it reaches the tree root. We refer the reader to the aforementioned papers for details.

[1] Shi et al. (2020) mentioned that the SEQ2SEQ model in their experiment is competitive with a state-of-the-art system (Suhr et al., 2020) on the SPIDER leaderboard.

3 Proposed Evaluation Benchmarks

Our goal is to design an evaluation benchmark that has the following out-of-domain generalization properties: (i) the training data involves a different set of domains from the test data, (ii) the questions and tables that appear in the train and test data are non-overlapping, not only in terms of the domains they belong to, but also in terms of the program fragments that they contain, and (iii) to simulate the more challenging setting that is often encountered in real applications, the test data is biased to contain more examples that involve both nested *column access operations*, like getting the start of a `"Term"` in Figure 2, as well as *composite column expressions*, like getting the duration of a `"Term"`. To this end, we propose a new synthetic dataset and a repartitioning of the SQUALL dataset into new train/test splits.

3.1 Synthetic Dataset

We consider three fictional domains inspired by common uses of tables: *finance*, *sports*, and *science*. We explain our synthetic dataset generation process through a running example as follows:

1. For each domain, we declare a set of formulas that relate different quantities (e.g., `"Income" = "Salary" + "Stock"`). The primitives used in these formulas define the set of available columns.
2. For each column we declare a set of noun phrases that can be used to refer to it (e.g., "wages" for `"Income"` and "base salary" for "salary"). We also define a SQL query template that shall be used for all programs: `SELECT <column> FROM t WHERE "Year" = <year>`, and a question template `What was <column> in <year>`? Note that the `"Year"` column is special and is included in all examples of this synthetic dataset.
3. We sample a formula and a variable from that formula (e.g., `"Income"` from formula `"Income" = "Salary" + "Stock"`). We then generate a question asking for this variable, randomly replacing the variable with a noun phrase in the corresponding set, and randomly generate a year value (e.g., use "wages" to replace "income" and generate a question "What was `[wages]` in `[2011]`?").
4. To generate the target program π^*, we randomly drop a variable from the sampled formula in step 3. If the asked value corresponds to this variable, we transform its reference in the SQL query so that it is expressed as a function of the columns that are kept (e.g., `"Salary" + "Stock"`), otherwise we use the column name (e.g., `"Income"`).
5. To generate a table schema we first add a `"Year"` column and two of the columns that were not sampled from the formula (e.g., `"Salary"` and `"Stock"`). We then sample k other columns and add them to schema ($k = 15$ in our experiments) as *distractor* columns. Note that we do not generate full tables for this synthetic dataset since we do not evaluate on table cell selections.

We construct benchmark datasets by first generating 1,000 examples per domain and then iterating over the domains and keeping the data generated for the current domain as our test data, while using the data of the remaining two domains for training. This results in three datasets, each with 2,000 train examples and 1,000 test examples. More details on the declarations for our domains can be found in Appendix A.1.

3.2 SQUALL Repartitioning

Aside from the synthetic dataset we also propose to repartition SQUALL into new train and test data splits, with a focus on the aforementioned out-of-domain generalization properties. The original splits for SQUALL were produced by uniformly sampling 20% of the tables to produce the test set and using the remaining 80% as the train set. This process was repeated 5 times and the evaluation metric results were averaged over the results obtained for each repetition. This resulted in similar tables being included in both the train and test sets (e.g., tables referring to two different basketball matches, but having identical schemas), and few examples in the test set required column operations. In order to avoid this issue, we propose the following algorithm for automatically constructing data splits focused on out-of-domain generalization on column operations:

1. Collect the table schemas used across all examples in the train and dev splits of the dataset (there are about 1,600 schemas; note that the test set is not annotated with SQL queries).
2. Construct a graph by treating each schema as a node, and adding an edge for each pair of schemas that share more than 33% of their columns.
3. Find all connected components of the graph. Each defines a cluster of table schemas.
4. Each table has a set of SQL queries associated with it: one for each example that uses this table. For each query we check if it is a `SELECT` of a single column or if it is a `SELECT` that involves column operations such as field accessors or arithmetic operations. We associate each cluster with the number of queries that involve such column operations.
5. Sort the clusters based on this number, in decreasing order, and then use the first 20% as the test set and the remaining as the train set. Note that adding a cluster to the train/test set is equivalent to

Data Category	# Examples	
	Train	Test
All	8,956	2,320
w/ `Score` Accessors	91	86
w/ `Score` Expressions	47	53
w/ `Date` Accessors	81	173
w/ `Date` Expressions	18	95

Table 1: Statistics for our repartitioned version of SQUALL, including the categories that we use in our empirical analysis and which are presented in §3.2.

adding all examples that use tables included in this cluster. This step will result in disproportionally more column operations being used in our test set than in our train set, which means that the model will need to learn to generalize well in this setting to do well in this dataset.

In the following sections we pay special attention to four data subcategories that are representative of the out-of-domain generalization setting for SQUALL:

- `Score` Expressions: Represents SQL queries that include expressions over columns of type `Score` (e.g., a query selecting the score difference for a basketball game).
- `Score` Accessors: Represents SQL queries that include field accessors for columns of type `Score` (e.g., a column with the results of a basketball game, like "89-72", and a query that requires accessing the first element of this score; i.e., "89").
- `Date` Expressions: Similar to `Score` expressions except using the `Date` and `TimeSpan` type (e.g., a query asking for the duration of a presidency term).
- `Date` Accessors: Similar to `Score` Accessors, except using the `Date` and `TimeSpan` type (e.g., a query asking for the start of a term).

We shall refer to these categories when reporting experimental results in §5. We provide statistics for the resulting dataset in Table 1.

4 Proposed Method

In this section we propose a simple approach for tackling this specific out-of-domain generalization problem that ought to serve as evidence that it is a real problem and that it is solvable, as well as a reference point for evaluating future approaches. Our approach consists of two new components that can be used in combination with any existing text-to-SQL parser: *schema expansion* and *schema pruning*. These components interact with the parser by preprocessing the table that is fed to it as input. This is illustrated in Figure 2.

As discussed in §1, there are two kinds of challenges related to out-of-domain generalization in text-to-SQL parsing, *column matching* and *column operations*, with the latter being more challenging. The goal of *schema expansion* is to reduce column operation challenges to column matching by adding synthetic columns to the table schema. These synthetic columns correspond to expressions or accessors over existing columns (e.g., a column that represents the sum of two columns). Rather than learning (or memorizing) the ways in which different types of columns can be composed together, we propose to inject prior knowledge as to what kind of symbolic operations are possible based solely on the column types in a schema. This reduces column operations to column matching by effectively bringing the target programs closer to their surface form in the natural language question. For example, `"Income"` can now map to a synthetic column that corresponds to the sum of `"Salary"` and `"Stock"` instead of having the parser produce the sum expression directly. Since our expansion is based on column types, we argue that it is reasonable to assume that all schemas are typed and our expansion could be applied to any new domain. It is also worth noting that even though our templates may not cover all cases,[2] when applying our method to new domains, developers can declare a few templates of their interest and apply schema expansion on these templates to create parser-friendly schemas. This would be more cost-effective compared to collecting large in-domain training data for training the parser.

Naturally, having a component that expands the table schema means that we may end up with large schemas that the parser has to deal with, which will often involve a lot of irrelevant columns (partially because the schema expansion component does not peek at the question). This can result in increased latency which is not desirable in real-world systems. To this end, we introduce a *schema pruning* component that looks at both the expanded table schema and the question and decides which columns to prune before invoking the parser. It can be argued that this pruning is as hard as parsing itself, but there is evidence from other areas that it can indeed be helpful (e.g., vocabulary selection; Chen et al., 2019; Xu et al., 2021). As we shall show schema pruning can actually provide an additional boost in accuracy, depending on architecture of the underlying parser.

4.1 Schema Expansion

A domain developer first declares a set of *templates* that specify the ways in which different column types can interact (e.g., it specifies that given a typed `TimeSpan` column that contains two subfields, `_1` and `_2`, the expression `TimeSpan._2 - TimeSpan._1` can be constructed that represents a duration), and the names for each such interaction (e.g., `"Duration"`).[3] The schema expansion component receives as input this set of templates along with the table schema and returns an expanded schema that includes additional columns generated by using all applicable templates. For our SQUALL experiments, we declared the templates shown in Table 2. Although these templates are somewhat tailored to this dataset, our main goal is to show that there is considerable room for improvement in this challenging generalization scenario, and that even a simple approach with minimal manual effort can result in significant gains.

4.2 Schema Pruning

We propose a simple schema pruning approach that is inspired by vocabulary selection methods in machine translation. Let us denote the input question

[2] An interesting future project idea would be to automatically expand schemas using pre-trained language models.

[3] Having a name that accurately reflects the meaning of the column operation results is desired, as semantic parsers are sensitive to column names for column matching.

Column : Type : Fields	Synthetic Column		Example	
	Column Name	Expression	Original Column Name(s)	Synthetic Column Name(s)
Expressions				
`x:TimeSpan:_1,_2`	`[x]` Duration	`x._2 - x._1`	Term	Term Duration
`x:Date,y:Date`	`[x & y]` Duration	`y - x`	Term Start; Term End	Term Duration
`x:Score:_1,_2`	`[x]` Difference	`x._2 - x._1`	Result	Result Difference
`x:Score:_1,_2`	`[x]` Sum	`x._2 + x._1`	Result	Result Sum
Accessors				
`x:TimeSpan:_1,_2`	`[x]` Start; `[x]` End	`x._1`; `x._2`	Term	Term Start; Term End
`x:Score:_1,_2`	Home `[x]`; Away `[x]`	`x._1`; `x._2`	Result	Home Result; Away Result
`x:Score:_1,_2,_3`	Win Record; Loss Record; Tie Record	`x._1`; `x._2`; `x._3`	Result	Win Record; Loss Record; Tie Record
`x:Score:_1,_2,_3`	First Round `[x]`; Second Round `[x]`; Total `[x]`	`x._1`; `x._2`; `x._3`	Score	First Round Score; Second Round Score; Total Score

Table 2: The templates we used for schema expansion over `TimeSpan`-, `Date`-, `Score`-valued columns, including column expressions and accessor operations. `x` and `y` denote columns in the original schema, `_1`, `_2`, and `_3` refer to tuple field accessors, `&` denotes overlapping column name tokens, and `;` is used as a column separator.

by q and the input column names after expansion by $c_1, \ldots, c_M$. We concatenate the question and the column names as `[CLS]` q `[SEP]` c_1 `[SEP]` ... `[SEP]` c_M `[SEP]` and feed the resulting sequence to a BERT encoder (Devlin et al., 2019). We then define the embedding of each column, $\boldsymbol{c}_i$, as the final-layer representation of the last token of that column's name. Finally, we define the probability that a column should be kept as $p_i = \text{Softmax}\left(\text{MLP}(\boldsymbol{c}_i)\right)$. We train this model based on whether each column is used in the corresponding SQL program. At inference time, we need to choose a threshold on the predicted probabilities for deciding whether to prune a column or not. We assume a transductive setting and choose this threshold such that the ratio of pruned columns over the test set equals to the ratio of pruned columns over the train set plus a constant hyperparameter to account for fact that accuracy will likely be lower for the test set than the train set. Note that assuming a transductive setting is fine because in a real-world system we could be tuning this threshold based on the last t requests made to the model. While this is not equivalent, assuming a large enough t, we should be able to adapt this threshold using the same approach.

Negative Column Sampling. As is evident from Figure 2, we also introduce a negative column sampling component. This is because we train our pruning model on the same data that we use to train the underlying parser (aside from the modified table schemas) and thus the pruning model can become good at pruning all irrelevant columns over this dataset. This will result in the underlying parser being unable to handle situations where irrelevant columns are mistakenly left unpruned by the pruning model. To this end, during training we introduce some irrelevant columns to improve the robustness of the underlying parser. We found that making sure to always include at least 3 columns in the resulting schemas was sufficient and equivalent to randomly sampling 1 or 2 additional columns for each training example, and so that is what we did in our experiments.

5 Experiments

We performed experiments on the two proposed benchmarks (as well as the existing version of the SQUALL benchmark), using the two current state-of-the-art parser architectures presented in §2 in combination with our proposed schema expansion and pruning components.

5.1 Experimental Setup

As described in §3, our synthetic benchmark consists of three domains, *finance*, *sports* and *science*. We repeat our experiments once for each domain. For each repetition we test on one of the domains, while training on the other two. For SQUALL, we present results on our repartitioned split from §3.2. For both datasets, we also include results for three i.i.d. splits. In each experiment, we compare four different configurations for the parsers:

1. Base: The underlying parser which can be either SEQ2SEQ or SMBOP.
2. Base + P: Base while also using schema pruning.
3. Base + E: Base while also using schema expansion.
4. Base + P + E: Base while also using both schema expansion and schema pruning.

We repeat each experiment three times using different random seeds and report mean *exact match accuracy* (i.e., fraction of examples where the predicted SQL queries exactly match the gold queries), and standard error for this mean.

Note that for SQUALL, researchers often also report *execution accuracy*, which measures the fraction of examples for which executing the predicted SQL queries results in the correct answer to the input question. However, we found that for 7% of the examples that are representative of out-of-domain generalization, executing the gold SQL queries does not yield the correct answer (e.g., in cases where the correct answer is a sub-string of a cell value). Therefore we chose to only report exact match accuracy in our experiments.

Dataset Split	Seq2Seq				SmBop			
	Base	Base + P	Base + E	Base + E + P	Base	Base + P	Base + E	Base + E + P
Synthetic Dataset								
I.I.D.	52.5 ± 2.2	86.5 ± 0.7	93.4 ± 0.6	**96.2** ± 0.3	85.3 ± 0.5	90.9 ± 0.2	**97.3** ± 0.1	**97.3** ± 0.1
Finance	17.4 ± 0.6	18.3 ± 0.8	58.7 ± 0.3	**65.9** ± 0.7	23.6 ± 0.2	24.4 ± 0.3	**69.7** ± 0.3	68.7 ± 0.1
Sports	16.8 ± 0.6	26.7 ± 0.4	69.0 ± 0.4	**71.8** ± 0.5	28.8 ± 0.9	28.8 ± 0.2	77.3 ± 0.5	**79.5** ± 0.2
Science	12.8 ± 0.1	17.5 ± 0.8	64.8 ± 0.2	**69.9** ± 0.6	20.1 ± 2.0	26.3 ± 0.2	69.7 ± 0.6	**71.2** ± 0.2
Squall Dataset								
I.I.D.	45.1 ± 0.6	46.3 ± 1.0	47.2 ± 0.9	**47.5** ± 1.0	46.4 ± 0.8	46.7 ± 0.6	48.1 ± 0.8	**48.2** ± 0.5
Repartitioning	35.0 ± 0.3	36.8 ± 0.2	38.0 ± 0.3	**39.3** ± 0.2	37.0 ± 0.1	39.8 ± 0.3	**42.1** ± 0.3	**42.1** ± 0.2
`Date` Expressions	1.4 ± 0.7	4.5 ± 0.3	28.0 ± 2.3	**43.2** ± 1.0	3.2 ± 0.9	10.6 ± 1.9	**50.2** ± 2.5	46.3 ± 0.6
`Score` Expressions	9.4 ± 1.1	5.0 ± 2.7	30.9 ± 3.5	**33.9** ± 1.7	26.4 ± 3.1	30.0 ± 3.0	47.2 ± 1.3	**51.6** ± 1.3
`Date` Accessors	19.1 ± 0.7	**26.3** ± 0.4	24.9 ± 1.0	26.0 ± 0.5	21.4 ± 0.3	**24.8** ± 1.4	23.5 ± 0.8	**24.8** ± 0.3
`Score` Accessors	18.1 ± 2.3	**22.6** ± 1.1	21.8 ± 1.0	18.1 ± 0.9	21.8 ± 1.3	**31.7** ± 0.8	26.4 ± 1.8	25.3 ± 2.0

Table 3: Mean accuracy and standard error for 3 experiment runs, computed over multiple different splits for each dataset. The best results in each row are shown in **bold red font**. Note that, when compared with the Base model, all gains statistically significant. + P stands for using the schema pruning model and + E for the schema expansion model.

5.2 Results

Synthetic Benchmark Results. Our results for this benchmark are presented in the top part of Table 3. A first observation is that performance on the i.i.d. split for the baseline parsers is significantly better than on the domain-based splits. Interestingly, our expansion and pruning components still provide a significant boost over baseline performance in this setting (up to 43.7% absolute accuracy / 83.2% relative). However, the baseline parsers are practically unusable in the domain-based splits. In this case, our approach provides a very significant accuracy gain, rendering them useful (*up to 55.0% absolute / 327.4% relative*).

SQUALL Benchmark Results. Our results for this benchmark are presented in the bottom part of Table 3. Similar to the synthetic benchmark, we observe that both parsers perform reasonably well on the i.i.d. split, but significantly underperform in our repartitioned benchmark. This is consistent with earlier observations by Suhr et al. (2020) and Lee et al. (2021). Furthermore, we observe that our expansion component helps boost the accuracy of both parsers significantly (up to 5.1% absolute / 13.8% relative) and the pruning component provides some small further improvements on top of that. However, we notice that the pruning component is not as helpful for SMBOP as it is for SEQ2SEQ, which we provide detailed analysis in §5.4. Drilling down a bit further, we observe that most gains are due to the data categories we defined in §3.2. Perhaps most importantly, we get a *47.0% absolute accuracy gain (1,468.8% relative)* for SMBOP on the "`Date` Expressions" category alone. This can be largely attributed to our schema expansion component, where by incorporating prior domain knowledge we are effectively reducing the original column operations problem to a column matching problem, which is significantly easier. As a result, we get significant improvements on both "Expression" data categories. We do not observe the same for "Accessor" categories, which we address in the following section.

5.3 When is Schema Expansion Helpful?

From Table 3, schema expansion does not seem to help much for "Accessor" expressions (i.e., Base + P performs as well as or slightly better than Base + E + P on those categories). In order to further understand the contribution of schema expansion, we conducted an ablation study where we compare the proposed Base + P + E with three more approaches: (1) E Expressions: the schema expansion component only uses "Expression" templates, (2) E Accessors: the schema expansion component only uses "Accessor" templates, (3) P Oracle: the schema pruning model is replaced with an oracle model that always only keeps the columns that are used in the gold SQL queries (so the parser only has to figure out how to use them, rather than also figuring out which ones to use). Note that (3) will be discussed in the following section. We present the results for this ablation study in Table 4. We observe that expanding "Expressions" but not "Accessors" boosts performance on the "Expressions" categories, and similarly for "Accessors". More importantly though we see that using either one alone performs worse than using both types of expansion, indicating that they both provide value and that they work well together.

5.4 When is Schema Pruning Helpful?

It is evident from Table 3 that schema pruning is useful both on its own (i.e., Base + P), but also on top of schema expansion (i.e., Base + E + P). For SMBOP, we observe that Base + P is more or less on par with Base. Though this may seem inconsistent with the SEQ2SEQ results at first, it is not actually surprising because SMBOP keeps the most relevant columns in the beam during bottom-up decoding, and thus it is implicitly already using a schema pruning component. Furthermore, we observe that schema pruning is especially useful on top of schema expansion for the column operation data categories ("Expressions" and "Accessors"). This is because in the corresponding examples we end up with a significantly larger number of expanded columns that labeled as negatives when training the pruning model.

Dataset Split	Seq2Seq				SmBop			
	Base + E + P	E Accessors	E Expressions	P Oracle	Base + E + P	E Accessors	E Expressions	P Oracle
Repartitioning	39.3 ± 0.2	37.4 ± 0.1	37.6 ± 0.1	**52.9** ± 0.3	42.1 ± 0.2	40.6 ± 0.2	40.4 ± 0.2	**57.9** ± 0.4
`Date` Expressions	43.2 ± 1.0	2.8 ± 0.7	39.6 ± 1.3	**81.0** ± 2.5	46.3 ± 0.6	16.3 ± 1.3	41.1 ± 1.0	**75.4** ± 1.7
`Score` Expressions	33.9 ± 1.7	14.2 ± 3.6	25.7 ± 1.7	**48.8** ± 1.7	51.6 ± 1.3	36.5 ± 4.1	48.4 ± 1.6	**74.2** ± 1.7
`Date` Accessors	26.0 ± 0.5	27.6 ± 0.6	26.6 ± 0.7	**33.1** ± 1.5	24.8 ± 0.3	23.5 ± 0.8	22.6 ± 2.1	**32.4** ± 1.0
`Score` Accessors	18.1 ± 0.9	20.6 ± 1.5	9.9 ± 0.8	**29.2** ± 1.1	25.3 ± 2.0	30.7 ± 2.8	15.3 ± 3.2	**58.1** ± 3.2

Table 4: Mean accuracy and standard error for 3 runs of our ablation studies on SQUALL repartitioning split for domain generalization, with the best results in each row colored **red**.

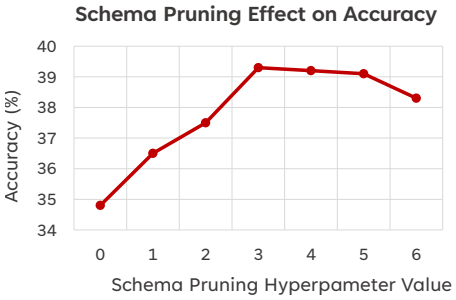

Figure 3: Accuracy (%) while varying the schema pruning model hyperparameter of §4. Pruning more than necessary has a significant negative impact on accuracy, while pruning less does not.

Title: Adam Szalai International Goals

score before goal → Home Score, Away Score

final score → Result, Home Result, Away Result

Goal	Date	Opponent	Score	Home Score	Away Score	Result	Home Result	Away Result
1	10/08/2010	San Marino	2-0	2	0	8-0	8	0
2	10/08/2010	San Marino	4-0	4	0	8-0	8	0
3	12/08/2010	Finland	0-1	0	1	1-2	1	2

Question: *How many games did he score in, where his team lost?*

Program: `SELECT COUNT(DISTINCT "Date") FROM t WHERE "Home Result" < "Away Result"`

Figure 4: Example that showcases some of the challenges that are not addressed by our approach, but which are accounted for in the evaluation benchmarks that we propose. In this case, the `"Score"` and `"Result"` columns have domain-specific semantics that are hard for the model to learn, and the question also depends on the title of the table, which current models do not take into account.

Schema pruning then filters most of these irrelevant columns before training the underlying parser, resulting in a more robust training procedure. Finally, in Table 4 we observe that P Oracle performs really well, indicating that investing in a good schema pruning model would be meaningful for improving generalization performance.

Schema Pruning Decision Threshold. As discussed in §4, the proposed schema pruning component requires setting a decision threshold hyperparameter. We already described the way we do this in §4, but it is also worth analyzing the impact of this decision on the overall parser accuracy. This is because, intuitively we expect that too aggressive pruning will likely cause cascading errors, while too conservative pruning would not be very effective and end up being equivalent to not using any pruning at all. To this end, we conducted a study for how the parser accuracy varies as a function of the schema pruning model hyperparameter which was discussed in §4. We performed this experiment using the SEQ2SEQ model which is more affected by the pruning component, over our repartitioned SQUALL benchmark. The results are shown in Figure 3. It is evident that aggressive pruning has a more significant negative impact on accuracy then conservative pruning.

5.5 Limitations

The proposed method is of course not without any limitations and in this section we would like to put attention on some of them. While schema expansion does help significantly when tackling out-of-domain generalization on column operations, there are a lot of cases that it cannot directly handle as currently designed. For example, consider the question-table pair shown in Figure 4. In this case the original table contains a `"Score"` column and a `"Result"` column. The interpretation of these columns is very domain-specific and in this case, `"Score"` refers to the score in a game right before the player of that row scored a goal, while `"Result"` refers to the final score of the game. Our schema expansion component cannot help with resolving distinctions of this kind. Arguably, one might say this is a challenge inherently related to column matching, but putting details aside, our approach coupled with the proposed benchmarks does help show that column operations pose a significant challenge for existing text-to-SQL parsers, and this paper provides a reference point that future work can build upon. Also, note that while constructing expansion templates requires some effort and may initially seem like a limitation of our approach, we have shown that this effort can be small relative to the amount of training data that would need to be annotated otherwise.

6 Conclusion

In this paper, we introduced and focused on *column operations*, an important challenge related to out-of-domain generalization for text-to-SQL parsing. We proposed two new evaluation benchmarks—one based on a new synthetic dataset and one based on a repartitioning of the SQUALL dataset—and showed that current state-of-the-art parsers significantly underperform when it comes to this form of generalization. We then proposed a simple way to incorporate prior domain knowledge to the parser via a new component called *schema expansion* that allows us to reduce the column operations

challenge to column matching; an arguably easier challenge. We also introduced a *schema pruning* component allowing us to scale schema expansion, and showed that when paired together, these two components can boost the performance of existing text-to-SQL parsers by a significant amount (up to 13.8% relative accuracy gain / 5.1% absolute in our experiments). Through column expansion, we created a new table schema that is more friendly to downstream parsers. Our work uses heuristics based schema expansion and works well when limited to columns that have specified types (e.g., scores or timespans), but our synthetic experiments suggest much larger potential on this problem. We hope this work could motivate future research on creating a parser-friendly table ontology. Future work could explore learning approaches that use models to automatically expand any table schema, for example, by showing appropriate prompts to ask pre-trained language models to tackle it (Brown et al., 2020; Petroni et al., 2019).

Acknowledgments

We thank the anonymous reviewers for their helpful comments, Jason Eisner for his detailed feedback and suggestions on an early draft of the paper, and Jacob Andreas, Ziyu Yao, Yuchen Zhang, and Sam Thomson for helpful discussions.

References

Ion Androutsopoulos, Graeme D Ritchie, and Peter Thanisch. 1995. Natural language interfaces to databases–an introduction. *Natural language engineering*.

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of Empirical Methods in Natural Language Processing*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Proceedings of Advances in Neural Information Processing Systems*.

David Chen and Raymond Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the Association for the Advancement of Artificial Intelligence*.

Wenhu Chen, Yu Su, Yilin Shen, Zhiyu Chen, Xifeng Yan, and William Yang Wang. 2019. How large a vocabulary does text classification need? a variational approach to vocabulary selection. In *Conference of the North American Chapter of the Association for Computational Linguistics*.

Hal Daumé III and Jagadeesh Jagarlamudi. 2011. Domain adaptation for machine translation by mining unseen words. In *Proceedings of the Association for Computational Linguistics*.

Hal Daumé III and Daniel Marcu. 2006. Domain adaptation for statistical classifiers. *Journal of artificial Intelligence research*.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-grounded pretraining for text-to-sql. In *Conference of the North American Chapter of the Association for Computational Linguistics*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics*.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the Association for Computational Linguistics*.

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond iid: three levels of generalization for question answering on knowledge bases. In *Proceedings of the World Wide Web Conference*.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Mueller, Francesco Piccinno, and Julian Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the Association for Computational Linguistics*.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2018. Mapping language to code in programmatic context. In *Proceedings of Empirical Methods in Natural Language Processing*.

Zhanming Jie and Wei Lu. 2014. Multilingual semantic parsing: Parsing multiple languages into semantic representations. In *Proceedings of International Conference on Computational Linguistics*.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, et al. 2020. Measuring compositional generalization: A comprehensive method on realistic data. In *Proceedings of the International Conference on Learning Representations*.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the International Conference of Machine Learning*.

Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. KaggleDBQA: Realistic evaluation of text-to-SQL parsers. In *Proceedings of the Association for Computational Linguistics*.

Michael Minock, Peter Olofsson, and Alexander Näslund. 2008. Towards building robust natural language interfaces to databases. In *International Conference on Application of Natural Language to Information Systems*.

Saeid Motiian, Quinn Jones, Seyed Iranmanesh, and Gianfranco Doretto. 2017. Few-shot adversarial domain adaptation. In *Proceedings of Advances in Neural Information Processing Systems*.

Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to generate pseudo-code from source code using statistical machine translation. In *International Conference on Automated Software Engineering*.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the Association for Computational Linguistics*.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? In *Proceedings of Empirical Methods in Natural Language Processing*.

Ohad Rubin and Jonathan Berant. 2021. SmBoP: Semi-autoregressive bottom-up semantic parsing. In *Conference of the North American Chapter of the Association for Computational Linguistics*.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? *Proceedings of the Association for Computational Linguistics*.

Tom Sherborne, Yumo Xu, and Mirella Lapata. 2020. Bootstrapping a crosslingual semantic parser. In *Findings of the Association for Computational Linguistics: EMNLP*.

Tianze Shi, Chen Zhao, Jordan Boyd-Graber, Hal Daumé III, and Lillian Lee. 2020. On the potential of lexico-logical alignments for semantic parsing to SQL queries. In *Findings of the Association for Computational Linguistics: EMNLP*.

Alane Suhr, Ming-Wei Chang, Peter Shaw, and Kenton Lee. 2020. Exploring unexplored generalization challenges for cross-database semantic parsing. In *Proceedings of the Association for Computational Linguistics*.

Alon Talmor and Jonathan Berant. 2019. Multiqa: An empirical investigation of generalization and transfer in reading comprehension. In *Proceedings of the Association for Computational Linguistics*.

Bailin Wang, Mirella Lapata, and Ivan Titov. 2021. Meta-learning for domain generalization in semantic parsing. In *Conference of the North American Chapter 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 Association for Computational Linguistics*.

Jingjing Xu, Hao Zhou, Chun Gan, Zaixiang Zheng, and Lei Li. 2021. Vocabulary learning via optimal transport for neural machine translation. In *Proceedings of the Association for Computational Linguistics*.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the Association for Computational Linguistics*.

Pengcheng Yin, Graham Neubig, Wen tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the Association for Computational Linguistics*.

Pengcheng Yin, John Wieting, Avirup Sil, and Graham Neubig. 2021. On the ingredients of an effective zero-shot semantic parser. *arXiv preprint arXiv:2110.08381*.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Caiming Xiong, et al. 2020. Grappa: Grammar-augmented pre-training for table semantic parsing. In *Proceedings of the International Conference on Learning Representations*.

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

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

A Appendix

A.1 Synthetic Dataset

Table 5 presents all formulas used to construct our synthetic benchmarks. To evaluate out-of-domain generalization, we ensure formulas in each domain do not overlap with other domains.

Domain	Formulas Used for Synthetic Data Generation
Finance	"total income" = "stock" + "salary" "salary" = "base" + "bonus" "account" = "checking account" + "saving account" "total income" = "taxable income " + "exclusions" "salary" = "weekly salary" * "week" "salary" = "monthly salary" * "month" "salary" = "yearly salary" * "year" "tax" = "salary" * "tax rate" "interest" = "principle" * "interest rate"
Sports	"total score" = "home score" + "away score" "total win" = "home win" + "away win" "total games" = "winning games" + "losing games" "aggregate score" = "first lag score" + "second lag score" "total field attempts" = "field goal attempts" + "three pointer attempts" "field goals made" = "field goal attempts" * "field goal percentage" "running time" = "distance" / "speed"
Health	"total case" = "exposed case" + "non-exposed case" "vaccinated number" = "first dose number" + "second dose number" "total case" = "positive case" * "positive rate" "total bed count" = "bed occupancy rate" * "occupied bed count" "actual deaths" = "mortality rate" * "actual cases" "actual deaths" = "death rate" * "period" "live birth" = "birth rate" * "period" "population" = "population density" * "area"

Table 5: Formulas used in three domains for constructing synthetic benchmarks.