Addressing Limitations of Encoder-Decoder Based Approach to Text-to-SQL

Octavian Popescu
IBM Research
o.popescu@us.ibm.com

Irene Manotas
IBM Research
irene.Manotas@ibm.com

Ngoc Phuoc An Vo
IBM Research
ngoc.phuoc.an.vo@ibm.com

Hangu Yeo
IBM Research
hangu@us.ibm.com

Elahe Khorasani
IBM Research
elkh@ibm.com

Vadim Sheinin
IBM Research
vadims@us.ibm.com

Abstract

Most attempts on Text-to-SQL task using encoder-decoder approach show a big problem of dramatic decline in performance for new databases. Models trained on Spider dataset, despite achieving 75% accuracy on Spider development or test sets, show a huge decline below 20% accuracy for databases not in Spider. We present a system that combines automated training-data augmentation and ensemble technique. We achieve double-digit percentage improvement for databases that are not part of the Spider corpus.

1 Introduction

Text-to-SQL is a task to automatically translate a given a natural language question into an SQL formula for a specific database (DB).

In 2018, Spider dataset (Tao et al., 2018) was released as a large-scale complex and cross-domain semantic parsing and Text-to-SQL dataset. Since then numerous models have been developed and evaluated using Spider dataset. Although there is a large increase in accuracy, growing from 20% to 70 + % between 2019 and 2021 of top ranked systems, later studies (Suhr et al., 2020a; Shi et al., 2020; Zhong et al., 2020), show that the accuracy of these models drops significantly, in some cases well below 20%, for databases outside Spider dataset.

In this paper, we present various experiments for Text-to-SQL task using DB schemas that are inside and outside of Spider dataset. We identify systematic types of errors and analyze their possible causes. From our error analysis, we propose solutions to address the limitations of the underlying encoder-decoder architecture and of the Spider dataset. We use training data augmentation technique in order to create a series of models for the same DB schema. By coupling each model SQL result with the information extracted from the query at the NLP step, we interpolate a final SQL formula via Ensemble technique. The system achieves very good results compared to its individual components and can be run on any new DB schema, in or outside of the Spider dataset, even when there exists a set of very limited number of training examples, on the order of several tens.

This paper is organized as follows. In Section 2, we review studies that defined a "standard" encoder-decoder architecture for Text-to-SQL. In Section 3, we define the problem of encoder-decoder approach based on declined performances on new DB schemas and new types of questions, that leads to the challenge of achieving good accuracy. Next, we describe our system in Section 5. The experiment results in Section 6 and conclusions in Section 7.

2 Related Work

The Text-to-SQL task is not a new task in NLP and there are many references on its complexity and achievements obtained recently (Navid et al., 2017; Popescu et al.; Yao et al., 2010). In (Tao et al., 2018), the Spider corpus with thousands of training examples was introduced in 2018. The Spider challenge has been introduced as well, where any team can have their system evaluated on an unknown corpus, which is a part of the Spider corpus, but not publicly available.

In (Cai et al., 2018; Gehring et al., 2017; Yin et al., 2016; Rabinovich et al., 2017), among others,

Proceedings of the 29th International Conference on Computational Linguistics, pages 1593–1603
October 12–17, 2022.

a sequence-to-sequence approach was introduced, opening the way for this type of architecture. Instead of translating into SQL, (Guo et al., 2019; Zhang et al., 2019; Bogin et al., 2019), the system translates into a representation that captures the semantics of the query, called Intermediate Representation (IR). From IR to SQL is a deterministic process: a context free grammar is used to convert one into another. The authors build on the work of (Sun et al., 2019; Cheng et al., 2019) that used Abstract Syntax Tree (AST).

A system that uses the the IR approach, but extends the context free grammar to include values, is Valuenet(VL) (Brunner and Stockinger, 2021). The code is available from (Brunner, 2021) . Valuenet obtained an accuracy of 60% on Spider development corpus, which put it among the best systems in the Spider challenge in 2020.

The RAT system (Wang et al., 2019) is based on the relation-aware self-attention mechanism, to address schema encoding, schema linking, and feature representation within a Text-to-SQL encoder. They augmented their system with BERT (Devlin et al., 2018), showing that using transformers brings a significant increase in the accuracy. But, most importantly, they also showed that "... all the known information about the schema, it is insufficient for appropriately encoding a previously unseen schema in the context of the question...". In (Suhr et al., 2020b), a detailed analysis of systems that performed well on Spider corpus confirmed a large drop in accuracy for DB schemas outside of the Spider corpus.

Given that creating a new training is an expensive process, an important part of research for Text-to-SQL task has been dedicated to improving the training methods, in order to reduce this cost. Paraphrasing (Ronak Kaoshik, 2021), dialog (Artzi and Zettlemoyer, 2011; Gur et al., 2018), or a combination of these (Herzig and Berant, 2019) were used to cope with this problem.

(Kalajdjieski et al., 2020) surveyed a series of methods to automatically create training corpora. Some of these models are pretrained using the Masked Language Modeling (MLM) task by either masking tokens from the utterance input or tokens from the schema input (Deng et al., 2021). This work demonstrated that jointly pre-training on utterances and table contents (e.g., column names and cell values) can benefit downstream tasks such as table parsing and semantic parsing. In (Xu et al., 2017) a general method, which avoids the dependency of order in deep learning system, is presented.

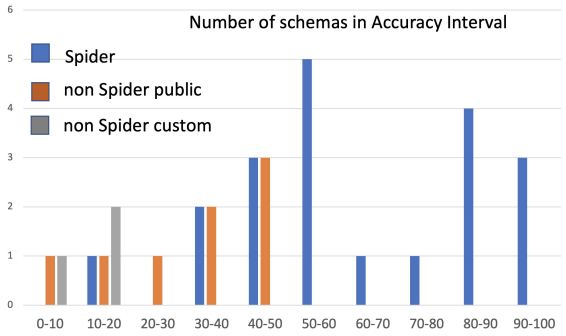

Figure 1: Accuracy for Spider vs. non Spider DB schemas

Our own observation aligns very well with the observations in the previous papers, and confirms the difficulty to cope with new unseen DB schema only by means of unique training corpus and/or model. However, our solution differs from what has been been proposed so far. The systems that used general pretraining methods, like RAT+GAP, did not show that the same performance is observed for out of Spider schemas, but only for the ones in Spider corpus. The systems that confronted the systematic errors that are produced in the case of new DB schemas, took a different path from ours, like (Suhr et al., 2020b), and did not exploit systematically the relationship between values and filters as a means to control the correctness of SQL formula. Their line of research assumed that this state of fact is due only to the lack in training. However, our experiments support the idea, that, while indeed some errors are traceable to the particularities of a certain training corpus, one important cause of many errors is ignoring the way the DB schema relationships could be expressed in natural language, see Section 3. By explicitly providing the linkage between values and schema definition, coupled with specific training generation, and interpolating over an ensemble of different models, our system shows improvement for DB schemas that are inside or outside of the same corpus.

3 Problem Definition

While the overall results for top performers on the Spider corpus are high, the variation for individual DB in the testing corpus can be large. However, when the same system is applied to the DB schemas outside of the Spider corpus we obtain a different distribution, Figure 1. The *Spider* his-

togram shows that accuracy for majority of test DB schemas fell into the range of 40-90%. The data was obtained running the default VL, system trained on the Spider training corpus and tested on the Spider development corpus. The *non Spider general* come from (Suhr et al., 2020a). None of these DB schemas have an accuracy above 50%. The *non Spider custom* refers to the training and test corpora for our own databases, described in section 4. For these databases, the accuracy is in the same category as for the lowest for *non Spider general*, namely below 20%. This category contains just one DB schema for the Spider corpus. Looking at what percentage of schemas are in three main categories, which roughly correspond to *bad, ok, good*, the difference between Spider vs. non Spider DB schemas is very large. This shows that indeed there is a very strong skew in the distribution towards the left intervals of accuracy for new DB schemas.

The reason for this skew for new DB schemas that is exemplified above was pointed out in (Suhr et al., 2020b) and (Zhong et al., 2020). Without proper reference to the linkage between query tokens to specific table/columns, based only on the examples seen in training, the deep learning model is not able to discern between similar queries in the context of a new DB schema. In Section 4 we provide details on these types of errors and their causes.

4 Baseline and Error Analysis

Spider dataset (Tao et al., 2018) is a large-scale cross-domain semantic parsing and Text-to-SQL annotated dataset. **ValueNet** (VL) (Brunner and Stockinger, 2021) is an open source deep learning system built using an encoder-decoder architecture. The Valuenet trained on Spider dataset represents our baseline.

Our outside Spider DB schemas are HR, WH and BI (*reference to articles kept anonymous for now*). For each of these schemas we have from 70 to 200 training examples.These training examples are split into two different corpora *ANS* - for training, and (2) *nANS* - for test.

VL obtained an accuracy of about 60% on the Spider development, but much lower results for the new DBs. In table Table 1 we present the results obtained by the baseline for the new DBs, separately for what we are going to us as training (ANS) and for test respectively(nANS).

Table 1: Accuracy for Default Valuenet+Spider corpus

DB	#ANS	acc	#nANS	acc
HR	121	24%	81	19%
WH	87	20%	42	15%
BI	109	4%	112	2%

4.1 Error Analysis

4.1.1 EA1. Absence of certain types of query/SQL in training

For questions having more than one value such as `Who bought most Apple products in Bestbuy?`, or `In which year did Mary shop in both Bestbuy and Radioshack?`, many times the VL model misses one of the values, or creates an incorrect filter relationships in the final SQL.

4.1.2 EA2. Limitations on Complexity

Queries having compound logical operators, such as `(filter1 OR filter2) AND filter3`, are not present in the corpus. When input questions need to generate queries like this, the Spider code cannot represent such type of queries correctly, it outputs a truncated representation, so the SQL is never correctly inferred.

4.1.3 EA3. Temporal questions

Currently, time is a string in Spider corpus, so the training examples contains `like` operator. However, time operations, like difference, conversion etc., are needed for queries like `... after 7 days...` or `... two days after April 1st ...`

4.1.4 EA4. Pre-processing information

By default, VL uses a Google API for NLP preprocessing of the query. As such, only general information about values, which is outside the scope and definition of the schema, is fed to the network. For example, for the token `Apple`, which in the context of that DB refers to the name of the manufacturer, the information provided is the web page describing this company, which does not help a lot in inferring the correct SQL.

4.1.5 EA5. Linking Values and Schema

Crucial relationship between values to the underlying schema is missing as input to the VL encoder-decoder. For instance, using the fact that the token `Apple` is related to the table `Manufacturers`

and the column `name` significantly increases the chances that the inferred SQL is correct.

4.1.6 EA6. Checking the consistency of SQL vs. syntax restrictions

English syntax provides hard restrictions on the constituency of phrases that are translated into SQL. A determiner, such as an adjective, has to be applied in the SQL formula to the value representing the head of the English phrase. For a query like `what is the age of pet of the youngest owner`, the correct SQL links `age` to a column of the table `pets` while `youngest` is an aggregation function on the table `students`, not vice versa. Many times the incorrect SQL is generated by the network because of inconsistency of SQL operators association.

5 Hybrid System (HS)

We created a new system that processes and extracts the relevant information from the input question and links tokens to specific tables and columns from the corresponding DB schema. We replaced Valuenet's pre-processing and Name Entity Recognition (NER) modules with our Disambiguation Dictionary Module (DDM) to provide crucial information of the relationship between values and schema. We also introduced an ensemble of neural network models to improve the performance of the inferred SQL queries. From the original Valuenet architecture we kept only the encoder-decoder for IR with the semQL grammar.

The hybrid system has a different data flow from Valuenet. First, an English query is processed by our Natural Language Processing module (NLP). In our approach, we correlate technologies that increase the performance by double figure percentage: (i) we devise a seed training data augmentation technique (STDA), that on the bases of the initial training corpus, we call seeds, is able to generate a larger training set. In a typical scenario, a few tens of seed examples lead to several hundreds of training examples; (ii) with different sets of automatically created training data, we train different models and use an ensemble technique to analyze the output of each of these models.

5.1 DDM query rewriting

The DDM component from NLP module, describes the columns, values, relationships between columns, and synonyms for columns and values, (Vadim et al., 2018; Popescu et al., 2019; Vo et al., 2019; Yeo et al., 2021). It creates a set of schema-dependent lexical rules using the information from schema annotations and a set of schema-independent template rules. From the set of matched rules a set of structured data items (DTI) is created. A DTI defines an item `[table name].[column name]` with operators such as `select`, `filter`, `aggregation`, etc., that need to be applied to the corresponding item. During the query processing process, a natural language question is disambiguated, annotated, and matched against the lexical rules. Using the information from DDM's output, and the input question in English, a query representation following the Spider format is created so it could be incorporated as an input to the neural network. The user's input query undergoes a double rewriting before the text goes to the network, as described below.

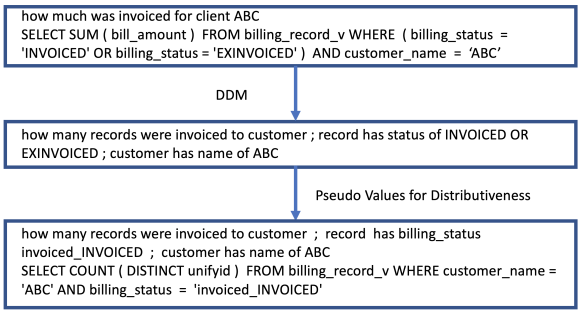

Figure 2: Rewriting of Queries in Hybrid System

- **DDM style question rewriting**. The input question is rewritten and augmented with the information provided by the NLP module, especially by the DDM module. The rewritten question contains explicit, English, references to tables and columns in the DB schema. For instance, for the question `how much was invoiced for client XYZ`, we rewrite the question as follow: the DDM's output for this query explicitly indicates that the token `invoiced` is a filter containing two values, `INVOICED_type1` and `INVOICED_type2`, from the table `billTable` and the column `status`. Also, a separate filter containing the value `XYZ` from the same table, but from a different column `name` is extracted. Two sentences are created accordingly: `bill has status of INVOICED_type1` OR `INVOICED_type2` and `customer has name of XYZ`. In the original query the values are replaced with their corre-

sponding columns, `how much amount was bill to customer` and the three sentences are adjoined in order to create the query that will be fed to the network: `how much amount was invoiced to customer; bill has status of INVOICED_type1 OR INVOICED_type2 ; customer has name of XYZ`.

The DDM rewriting of the user query adds the schema information into simple English sentences. In this way, the encoder-decoder have explicitly access the connection between the English tokens and their correspondent in the DB table and columns. See first two boxes in Figure 2.

- **pseudo-values for logically compound filters**. The SQL corresponding to the initial query in Figure 2 exemplifies a case where the priority of logical operators is taken into account. The Spider code does not properly handle these types of SQL constructs. We found two ways to automatically rewrite the SQL query such that this problem is handled properly. (1) to rewrite the SQL and the English question as a series of sub-questions such that each question in the query does not contains filters that need to be distributed into several logical conditions. Thus, we combine the individual results of the individual SQL conditions into a single result corresponding to the original SQL query; or (2) to use the pseudo-values to represent composite SQL conditions in the WHERE clause. Since some queries have values that are part of a composite filter, we can use a new pseudo-value representing the composite filter instead. For instance, for the query shown in Figure 2, we used the pseudo-value `invoiced_INVOICED` to represent the filters on both values present in the query for the same column. In the rewritten SQL instead of a compound filter we use this pseudo-value filter, which is linked through the `AND` operator to the other independent filter from the query, `customer_name='XYZ'`. See last box in Figure 2.

The DDM style question rewriting and the pseudo-values are applied both during training and inference time. During training, the neural network has access to the gold SQL query, and the SQL is rewritten using pseudo-values, if necessary. During inference, only the English question is rewritten, and the SQL is inferred. If the SQL is correct, this SQL will have the same form as the gold SQL query, and a rewritten procedure that replaces the pseudo-values back to the original value is applied to obtain the final SQL.

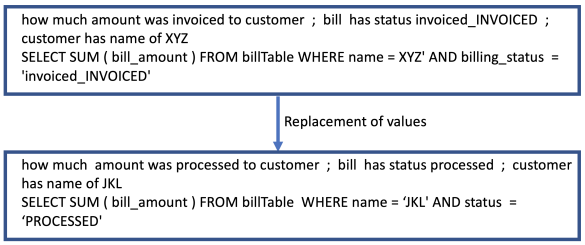

Figure 3: Replacing Values for Training Data Augmentation

The **HS**, by rewriting the queries before the IR is generated, is an effective solutions for the issues discussed in Error Analysis section 4.1.

5.2 Seed Training Data Augmentation (STDA)

The number of training examples for a given schema has a large impact on the accuracy of the trained deep learning model. However, for each DB schema only a small number of training examples is available, usually on the order of tens to a couple of hundreds at most. We call this small set of training examples the *seeds* set. To increase the number of training examples, we apply a data augmentation mechanism in which we take the seeds set, and automatically replace the corresponding values, in queries having multiple values, with alternative values from the same table and column as in the original example. We replace values in both the English question and the SQL query for training. In Figure 3 we show an example of how using a *seed* example, we can create a new training example.

Table 2: Seed Training Data Augmentation

Schema	Size of seeds	Size of STDA
HR	90	1500
WH	80	2000
BI	100	3300

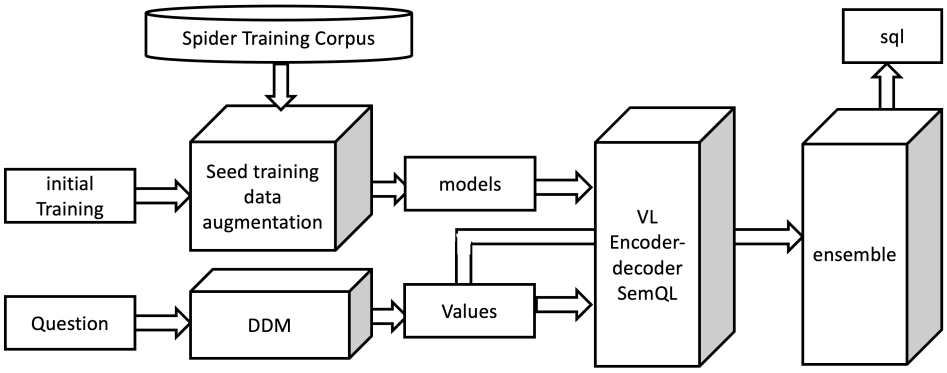

Figure 4: Hybrid System Architecture

5.3 Ensemble

We used the technique described in subsection 5.2 to create different training subsets from the original *seeds* training data for each DB schema. We created overlapping subsets, each one containing between 80 and 90 percent of the data available in the seeds set. We created a series of distinct models by training the deep learning model on the concatenation of the original Spider training set and each of the new generated training datasets for each DB schema.

A new query, that was not originally in the *seeds* set, is tested with all trained models. Each model infers an SQL that usually is not the same for all trained models. We chose the SQL query that obeys a set of hard coded restrictions created for each query. The restriction is that the filters in the SQL query observe the relationship between values and the DB schema as seen by the output of the rule-based system. These relationships consider values and numerical operators only, not the the body of the SQL formula. Therefore, we control only the compatibility between values in filters and the schema definition, and not the actual form of the SQL query e.g., join paths which inferred by the neural network. Figure 5 shows the inferred output from different models in the ensemble for the same query, the query at the bottom of the figure is the final selected SQL query.

Assembling the Output SQL. The ensemble component selects one SQL query that complies with the set of restrictions determined previously, this is done by assembling the output from a set of models created with the training datasets generated with STDA technique. The pseudo code shown in Algorithm 1 describes the steps that lead to the final SQL formula after interpolating the output from the models in the ensemble, as described above. In a nutshell, the algorithm considers the SQL query from each model, M_i, which is checked against the restrictions determined by DDM's output which are kept in $RSet$. The procedure consists of checking filters, $Filter\_M_i\_sql$, and operators, $Operators\_M_i\_sql$ one by one, independently of

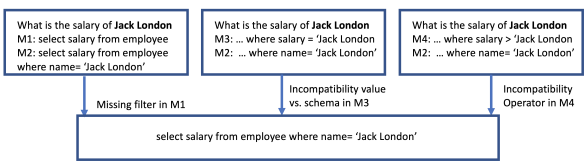

Figure 5: Interpolation over Multiple Models

Algorithm 1 Restrictions on Ensemble outputs

Require: Q ▷ English user query
Require: $M_1, M_2, ..., M_n$ ▷ DL models
Require: $RSet$ ▷ Restriction Set, DDM output
  while M_i **do**
    $Filter\_M_i\_sql$ ▷ extract filters from SQL
    $Operator\_M_i\_sql$ ▷ extract operators from SQL
    $EnsembleFailure \leftarrow True$ ▷ ensemble has no output
    if $Filter\_M_i\_sql$ not in $RSet$ **then**
      $rewrite\_filter$
    end if
    if $Operator\_M_i\_sql$ not in $RSet$ **then**
      $rewrite\_operator$
    end if
    if $Operator\_M_i\_sql = RSet$ and $Filter\_M_i\_sql = RSet$ **then**
      $EnsembleFailure \leftarrow False$
      return M_i_sql
    end if
  end while
  if $EnsembleFailure$ **then**
    return emptySQL

one another.

The architecture of the hybrid system is presented in Figure 4.

6 Experiments and Results

In this section we present the experiments and the results we obtained using the hybrid system against the baseline obtained by the original Valuenet trained on the Spider corpus. Besides the three schemas presented in Section 3, we also consider here the following schemas from the the Spider corpus: *pets_1* and *dog_kennels*. We considered specifically these two DB schemas because the default system scored less than median on them. In Table 1, for the HR, WH, and BI DB schemas the training was the *ANS* corpora, and *nANS* part was kept for testing, see Section 4. That is, we use the *ANS* corpus for direct training, or it was subject to STDA as described in Section 6.2 which led to creation of different models to which the ensemble module described in 6.3 was applied. In the latest case, we created a series of 5 models for each schema by splitting the training into five parts, in $4:1$ ratio (each model was using 80% of the original seeds for training augmentation) The evaluation of the accuracy considers the result of the SQL from our system vs. the result of the gold SQL formula. A system inferred SQL is considered correct only if all values and only those ones are returned by the gold SQL as well (execution accuracy) In all the tables , # represents the size of corpus, *acc* represents the accuracy, *HS* is our system with DDM module, *STDA* is the seed training module and *ens*, the ensemble module. The training for our schema was always combined with the Spider training corpus to obtain a model. So, a a model for a new DB having 100 training examples is actually build from 7000(Spider)+100 examples.

In Table 3 we present the results of the HS system without STDA, that is, we used 100% of the seeds training.

Table 3: Accuracy of hybrid system no STDA

DB	#train	acc
HR	121	36%
WH	87	26%
BI	87	38%
Pets_1	30	32%
Dog_kennels	62	48%

The results for *pets_1* and *dog_kennels* are different than the ones reported in Section 4, because in this experiment these schemas were removed from Spider development corpus.

We also carried out a cross validation experiment on the 5 models obtained for each schema via STDA. The results are in Table 4. The results are very high for each 5th part used for testing, but still low for the actual test, *nANS*, as seen in the second column of Table 5 .

Table 4: Accuracy for cross validation

DB	#seed	#STDA	#acc 5th part
HR	90	1500	78%
WH	70	2000	72%
BI	80	3300	80%
Pets_1	30	1500	68%
Dog_kennels	66	1500	72%

Finally, we used the 5 models from the cross validation for each schema with STDA with Ensemble and we obtained the results shown in Table 5. The ensemble gets significant improvement over the individual models, see third column.

Table 5: Accuracy hybrid system+ensemble

DB	HS+STDA acc	HS+STDA+Ens acc
HR	50%	62%
WH	40%	48%
BI	62%	66%
Pets_1	35%	41%
Dog_kennels	55%	57%

Putting together the results obtained, We indicate what is the average accuracy gain/loss for each type of system and each schema, *ANS* and *nANS* in Table 6.The benefit in using the STDA was in double digit percent improvement for all schemas. The ensemble brings an extra 5 to 10% improvement. The ensemble was very precise, having more than 98% precision, with a recall of 47%.

In 4.1 we analyzed certain patterns of errors. To see how much the hybrid system can improve on **EA1**, we compiled a specific corpus for multivalue question/queries examples for HR DB schema. This corpus has 64 questions that have more than one value, such as `What Apple product has the largest stock in Bestbuy?` The

Table 6: Accuracy Gain over Baseline

DB	BS	HS STDA	Ens
HR	22%	+17%	+43%
WH	17%	+12%	+29%
BI	3%	+15%	+63%
Pets_1	30%	+6%	+11%
Dog_kennels	45%	+5%	+12%

Table 7: EA1-multivalue queries improvement

system	accuracy
defaul VL	13%
default VL+tr	16%
HS no STDA	38%
HS+STDA	54%
HS+STDA+ens	78%

training corpus for HR, consisting of 57 questions, contained only 19 questions with multiple values. The results we obtained applying the hybrid system only on the multiple values corpus are in Table 7. The *default VL* is the the out-of-the-box VL system, *default VL+tr* is the same system trained on the extra training, *HS* , *STDA* and *ens* stand for hybrid system, seed training data augmentation and ensemble techniques respectively.

The ensemble algorithm works particularly well for this type of corpus. It is because the majority of wrong SQL had the wrong reference to table and column, which makes them recoverable due to the DDM information used in ensemble.

EA2 issue was related to the logically compound filters. Because Spider code cannot correctly process those types of SQL elements, it is not fair to compare systems that employ any of our techniques vs. a system does not have any mechanism for dealing with compound logical operators. Currently, we do not known of any other system that manages this problem.

The efficiency of our solution is further proved by the experiments using **temporal questions**. The Spider corpus have a poor representation of temporal questions, less than 8% in training and less than 7% in dev, and, as pointed out in **EA3** the SQL associated contains string , not date evaluation. We manually created a corpus containing temporal questions for HR and WH. This corpus contains questions with one data value, like, `how many products were sold on 2/2/2014`, we refer to them as *tmp_1v*, and

Table 8: Temporal Questions corpus

temporal corpus	#HR	#WH
tmp_1v train	28	35
tmp_1v test	14	22
tmp_mv train	44	70
tmp_mv test	32	42

Table 9: EA3 - Temporal Questions Results

system	HR tmp_1v	WH tmp_mv
default VL	18%	0%
default VL+tr	24%	0%
HS+STDA+ens	58%	44%

questions that more than one data value, like `how many products are sold with prices greater than 200 dollars between 2017/01/01 and 2017/12/31`, we refer to as *tmp_mv*.

For *temp_1v* the baseline system had a performance similar to the one for non temporal questions. However, for *tmp_mv* the situation change: the baseline system achieved 0% accuracy, and the situation did not improved when the available temporal training was provided to the system to build a new model. On the other hand, our system HS+STDA+Ens obtained a good accuracy, see Table 9. The fact that VL system obtains 0% is due to the inability of the system to correlate a value with DB table and column, which the technique introduced here for STDA+ens remedies. In Appendix we provided a sample of user's questions and the output from the baseline system vs. our system.

7 Conclusion and Further Research

In this paper we present a hybrid system for the Text-to-SQL task. The architecture of this system was driven by seeking solutions to a problem that we found when we tried to run an encoder-decoder system with out-of-the-box configuration, namely a large drop in accuracy for our DB schemas. The problem was reported in the results published by other teams as well. Our solution was to construct a system that does schema linkage and employs a training data augmentation technique. The system showed a good accuracy for DB schemas in and out of the Spider corpus. We obtained a double figure improvement of the accuracy, in some cases close to 30%, and not less than 9%. The lowest increase was for a schema that was considered difficult in

the Spider corpus.

Our experiments strongly suggests that a perfect training corpus cannot be built such that the accuracy on any new schema will be high. However, by implementing schema linkage and specific training, automatically created, one can realize a high jump in accuracy for deep learning networks.

Our future research will focus on further improving the training with queries that are not properly represented in the Spider corpus and on the methodology for training data augmentation.

8 Appendix

Here we show examples of the questions and inferred SQL for the HR and WH schemas, respectively. For each question we present the result of the default VL system, first SQL, vs. the result of our system, HS+STDA+Ensemble, the second SQL. For each question, we show the error made by the default system and the name of the system that inferred the correct SQL. The questions are shown exactly as they were written by users.

8.1 WH DB schema examples of inferred SQL.

- who manufactured iphone **: wrong table; correction via Ensemble**
  - SELECT T1.name FROM manufacturers AS T1 JOIN products AS T2 ON T1.manufacturer_id = T2.manufacturer_id WHERE T1.name = 'IPHONE'
  - SELECT T1.name FROM manufacturers AS T1 JOIN products AS T2 ON T1.manufacturer_id = T2.manufacturer_id WHERE T2.type = 'IPHONE'

- what did Richard buy **: wrong path; correction via STDA**
  - SELECT T1.name FROM customers AS T1
    WHERE T1.customer_id = Richard and T1.name = 'Richard'
  - SELECT T1.type FROM products AS T1 JOIN sales_details AS T13 ON T1.product_id = T13.product_id JOIN sales AS T14 ON T13.sales_id T̄14.sales_id JOIN customers AS T2 ON T14.customer_id = T2.customer_id WHERE T2.name = 'Richard'

- who bought at bestbuy **: wrong path, wrong column; correction via STDA+Ensemble**
  - SELECT T1.name FROM customers AS T1 JOIN sales AS T13 ON T1.customer_id = T13.customer_id JOIN sales_details AS T14 ON T13.sales_id = T14.sales_id JOIN products AS T2 ON T14.product_id = T2.product_id WHERE T2.price = BESTBUY
  - SELECT DISTINCT T1.name FROM customers AS T1 JOIN sales AS T13 ON T1.customer_id = T13.customer_id JOIN shops AS T2 ON T13.shop_id = T2.shop_id WHERE T2.name = 'BESTBUY'

- what is sold at bestbuy **: wrong column, correction via Ensemble**
  - SELECT T1.name FROM vendors AS T1 JOIN sales AS T13 ON T1.vendor_id = T13.vendor_id JOIN sales_details AS T14 ON T13.sales_id = T14.sales_id JOIN products AS T2 ON T14.product_id = T2.product_id WHERE T2.price = BESTBUY
  - SELECT distinct T2.type FROM shops AS T11 JOIN sales AS T22 ON T11.shop_id = T22.shop_id JOIN sales_details AS T1 ON T22.sales_id = T1.sales_id JOIN products AS T2 ON T1.product_id = T2.product_id WHERE T11.name = 'BESTBUY'

- show me apple product names **: wrong table, correction via STDA+Ensemble**
  - SELECT T1.name FROM manufacturers AS T1 WHERE T1.name = 'APPLE'
  - SELECT T1.type FROM products AS T1 JOIN manufacturers AS T2 ON T1.manufacturer_id = T2.manufacturer_id WHERE T2.name = 'APPLE'

References

Yoav Artzi and Luke Zettlemoyer. 2011. Bootstrapping semantic parsers from conversations. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019. Representing schema structure with graph neural networks for text-to-sql parsing. *arXiv preprint arXiv:1905.06241*.

Brunner. 2021. Valuenet source code.

Ursin Brunner and Kurt Stockinger. 2021. Valuenet: A natural language-to-sql system that learns from database information.

Ruichu Cai, Boyan Xu, Xiaoyan Yang, Zhenjie Zhang, Zijian Li, and Zhihao Liang. 2018. An encoder-decoder framework translating natural language to database queries.

Jianpeng Cheng, Siva Reddy, Vijay Saraswat, and Mirella Lapata. 2019. Learning an executable neural semantic parser. *Computational Linguistics*, 45(1):59–94.

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

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*.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional sequence to sequence learning.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. *arXiv preprint arXiv:1905.08205*.

Izzeddin Gur, Semih Yavuz, Yu Su, and Xifeng Yan. 2018. Dialsql: Dialogue based structured query generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Melbourne, Australia. Association for Computational Linguistics.

Jonathan Herzig and Jonathan Berant. 2019. Don't paraphrase, detect! rapid and effective data collection for semantic parsing. *arXiv preprint arXiv:1908.09940*.

Jovan Kalajdjieski, Martina Toshevska, and Frosina Stojanovska. 2020. Recent advances in sql query generation: A survey.

Yaghmazadeh Navid, Wang Yuepeng, Dillig Isil, and Thomas Dillig. 2017. Query synthesis from natural language. *International Conference on Object-Oriented Programming*.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a theory of natural language interfaces to databases.

Octavian Popescu, Ngoc Phuoc An Vo, Vadim Sheinin, Elahe Khorashani, and Hangu Yeo. 2019. Tackling complex queries to relational databases. In *Asian Conference on Intelligent Information and Database Systems*, pages 688–701. Springer.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*.

Prakash R Shaurya Agarawal Naman Jain Mayank Singh Ronak Kaoshik, Rohit Patil. 2021. Acl-sql: Generating sql queries from natural language.

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*.

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

Alane Laughlin Suhr, Kenton Lee, Ming-Wei Chang, and Pete Shaw. 2020b. Exploring unexplored generalization challenges for cross-database semantic parsing.

Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, and Ming Zhou. 2019. Semantic parsing with syntax and table aware sql generation. *56th Annual Meeting of the Association for Computational Linguistics*.

Yu Tao, Zhang Rui, Yang Kai, Yasunaga Michihiro, Wang Dongxu, Li Zifan, Ma James, Li Irene, Yao Qingning, Roman Shanelle, Zhang Zilin, and Radev Dragomir. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *EMNLP*.

Sheinin Vadim, Khorasani Elahe, Yeo Hangu, Xu Kun, An Vo Ngoc, Phuoc, and Octavian Popescu. 2018. Quest: A natural language interface to relational databases. *LREC*.

Ngoc Phuoc An Vo, Octavian Popescu, Vadim Sheinin, Elahe Khorasani, and Hangu Yeo. 2019. A natural

language interface supporting complex logic questions for relational databases. In *International Conference on Applications of Natural Language to Information Systems*, pages 384–392. Springer.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*.

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

Ziyu Yao, Yu Su, Huan Sun, and Wen tau Yih. 2010. Model-based interactive semantic parsing: A unified framework and a text-to-sql case study. *IJCNLP*.

Hangu Yeo, Elahe Khorasani, Vadim Sheinin, Ngoc Phuoc An Vo, Octavian Popescu, and Petros Zerfos. 2021. Programmatic database language generation for big data applications. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 2848–2856. IEEE.

Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. 2016. Neural enquirer: Learning to query tables with natural language.

Rui Zhang, Tao Yu, He Yang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-based sql query generation for cross-domain context-dependent questions. *arXiv preprint arXiv:1909.00786*.

Ruiqi Zhong, Tao Yu, and Klein Dan. 2020. Semantic evaluation for text-to-sql with distilled test suites.