NL2pSQL: Generating Pseudo-SQL Queries from Under-Specified Natural Language Questions

Fuxiang Chen[1,4], Seung-won Hwang[2], Jaegul Choo[3], Jung-Woo Ha[1] and Sunghun Kim[1,4]

[1]Clova AI Research, NAVER
[2]Yonsei University
[3]Korea University
[4]Hong Kong University of Science and Technology

fchenaa@cse.ust.hk , seungwonh@yonsei.ac.kr , jchoo@korea.ac.kr , {jungwoo.ha , sung.kim.n}@navercorp.com

Abstract

Generating SQL codes from natural language questions (NL2SQL) is an emerging research area. Existing studies have mainly focused on clear scenarios where specified information is fully given to generate a SQL query. However, in developer forums such as Stack Overflow,[1] questions cover more diverse tasks including table manipulation or performance issues, where a table is not specified. The SQL query posted in Stack Overflow, Pseudo-SQL (pSQL), does not usually contain table schemas and is not necessarily executable, is sufficient to guide developers. Here we describe a new NL2pSQL task to generate pSQL codes from natural language questions on under-specified database issues, in short, NL2pSQL. In addition, we define two new metrics suitable for the proposed NL2pSQL task, Canonical-BLEU and SQL-BLEU, instead of the conventional BLEU. With a baseline model using sequence-to-sequence architecture integrated with denoising autoencoder, we confirm the validity of our task. Experiments show that the proposed NL2pSQL approach yields well-formed queries (up to 43% more than a standard Seq2Seq model). Our code and datasets are publicly available at http://github.com/clovaai/nl2psql.

1 Introduction

Converting natural-language questions to SQL codes (NL2pSQL) is an active area of research in natural language processing. However, earlier work (Hemphill et al., 1990; Brad et al., 2017; Zelle and Mooney, 1996; Tang and Mooney, 2000; Popescu et al., 2003; Lawrence and Riezler, 2016; Li and Jagadish, 2014; Roy et al., 2013; Yaghmazadeh et al., 2017; Zhong et al., 2017; Yu et al., 2018c; Finegan-Dollak et al., 2018) mainly focused on fully specified questions where the given question and the table can be uniquely answered by a generated SQL query. For example, a question such as "who is the manufacturer for the order year 1998?" can only be uniquely answered when the table that can answer the question is attached to the question. This requires the generated SQL query to be executable, in order to verify its correctness.

Meanwhile, real-life NL2pSQL questions from developer forums such as Stack Overflow (SO) (StackOverflow, 2018) are mostly under-specified to generate an executable answer. Actual questions deal with more diverse issues, including table creation, backup, or performance, and often present with a less detailed context (such as missing table name or schema). For example, a SO question[2] such as "How to reset AUTO_INCREMENT in MySQL?" can only be answered by a non-executable template "ALTER TABLE [tablename] AUTO_INCREMENT = 1", where human developers are expected to read and then feed the missing context. We call these questions *Under-specified NL2pSQL*, distinguished from the conventional NL2pSQL task on fully specified questions.

Our key contribution is to address 1) the novel problem of generating human-readable pseudo SQL code to diverse ranges of under-specified developer questions with our own curated dataset and to propose 2) a new approach as well as 3) evaluation metrics suitable for the proposed task. Our dataset contains crawled posts in SO between the year of 2008 and 2017. The proposed task addresses the following new opportunities and challenges.

First, our new dataset covers diverse scenarios with 362 distinct SQL keywords, such as *USING, UNIQUE, SELECT, UPDATE, TRUNCATE, etc*,

[1]https://www.stackoverflow.com

[2]https://stackoverflow.com/questions/8923114

Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, pages 2603–2613,
Hong Kong, China, November 3–7, 2019. ©2019 Association for Computational Linguistics

unlike specified scenarios such as data lookup using simple *SELECT* queries. Meanwhile, 19% of the answers in SO do not contain the *SELECT* keyword at all.

Second, we eliminate the requirement of a generated code to be executable, for which we propose new evaluation metrics without such a requirement. An existing metric is lexical comparisons with the human-generated templates using BLEU scores (Papineni et al., 2002). This metric based on counting matching n-gram may penalize mismatches in non-essential tokens such as variable names (Lin et al., 2018; Zhong et al., 2017), while mismatches of SQL keywords should not be tolerated.

Our contribution includes the following:

- **Dataset:** We publicly release the NL2pSQL dataset containing 1,244 pairs as well as the denoising dataset for NL2pSQL containing 406,384 pairs.[3]

- **NL2pSQL:** We propose an end-to-end natural NL2pSQL model without the restriction of an executable SQL query trained and evaluated for under-specifed questions. Existing sequence-to-sequence models are known to generate repetitive tokens, but our denoising autoencoder can effectively fix errors in SQL statements towards readable codes. Our model performs a two-step translation by cascading two sequence-to-sequence models, one for generating initial SQL statements from natural language input, and the other to fix ill-formed SQL statements.

- **Evaluation metric:** While existing NL2pSQL metrics penalize the mismatch of any terms, we propose novel metrics allowing the mismatches for non-essential terms, yet requiring matches for important SQL keyword: Specifically, Canonical-BLEU abstracts the identifier names of the SQL statements before calculating the BLEU scores while SQL-BLEU calculates the BLEU scores using only the SQL keywords. Different forms of SQL queries may exist for a given natural language, and we note that no single metric is capable to accurately discern one form from the other. Thus, two proposed metrics complement each other.

[3] https://github.com/clovaai/nl2psql

2 Related Work

Natural Language to SQL. Bridging the gap between natural language and SQL queries has been a long-studied area (Warren and Pereira, 1982; Popescu et al., 2003, 2004; Li et al., 2006; Giordani and Moschitti, 2012). With the emergence of deep learning, neural network-based approaches have gained popularity including models using sequence-to-sequence and reinforcement learning (Zhong et al., 2017; Xu et al., 2018; Iyer et al., 2017; Dong and Lapata, 2016; Yu et al., 2018a). These models assume fully-specified questions and require human-annotated executable SQL queries for training and evaluation. Thus, they are not applicable to under-specified questions, which are the majority of issues in real-world software developer forums.

An alternative approach is designing semantic parsers, to exploit rich prior knowledge in the form of features and grammars (Zettlemoyer and Collins, 2005). It aims at mapping natural language utterances to an executable logical form such as SQL. They require rich resources – in the form of either grammars or training data to extract such grammars. As existing semantic parsers on SQL build on pairs of SQL and their execution results, which cannot exist in our problem context (Xu et al., 2018; Zhong et al., 2017; Sun et al., 2018; Yu et al., 2018c,b) of under-specified questions, we adopt a generative approach in our model.

Program Synthesis Program synthesis or automatic code generation is a relatively new field and has gained much attention lately. In previous work (Ling et al., 2016; Rabinovich et al., 2017; Yin and Neubig, 2017; Balog et al., 2017; Murali et al., 2018), the generated code is targeted at programming languages such as Java and Python, or they used a reduced specification such as a domain-specific language (DSL). Meanwhile, Beltramelli utilized GUI screenshots to generate markup language codes (Beltramelli, 2017). Our work differs in the sense that we are not constrained to a reduced specification but deal with a new domain of SQL statements, instead of markup languages or fully-fledged programming languages such as Java and Python.

Denoising and Program Repair Denoising is a popular research area in computer vision (Buades et al., 2005; Elad and Aharon, 2006; Dabov et al., 2007; Zhang et al., 2017; Meiniel et al., 2018) and

natural language processing (Kouno et al., 2015; Napoles et al., 2017). In software engineering, denoising tasks focus on bug fixing in fully-fledged programming languages such as Java and C (Long and Rinard, 2015; Mechtaev et al., 2016; Gupta et al., 2017). Our work is different in that we are combining the denoising aspect into the auto-generated SQL queries to complement the limitation of neural generation, which has not been previously considered.

3 Proposed Approach

3.1 NL2pSQL Dataset

To obtain human-written pseudo SQL statements with their corresponding natural language description, we turn to Stack Overflow (SO). We use the public SO data from the Stack Exchange Data Dump website for convenience purposes as the data are already structured in an XML form, e.g., 'Posts.xml', for easy extraction. Our goal is to curate question and code pairs: For example, in the SO Question 6308594, [4] the natural language input is *"how can I copy data from one column to another in the same table"*, and the code is *"update table set columnB = columnA"*.

In particular, we collect all the answers where they have a single code block and their corresponding questions are labeled with an *SQL* tag from 'Posts.xml'. Note that this xml file contains all the SO questions and answers since its establishment from August 2008 to September 2017. We also used the data from 'Postlinks.xml' (described in the later paragraph).

A code block can contain multiple lines of codes. We observed that indentation for code readability made numerous SQL statements be represented with multiple lines. A SQL statement indented with a different style will have a trivial effect when being read and executed by a SQL parser. Thus, we transformed all the multi-line SQL statements into a single line by replacing the line break with a white space. In this study, we only consider the SQL statement with a maximum token (delimited by a white space) length of 100.

Among questions with SQL answers, we avoid error debugging questions but focus on implementation questions with code answers. To this end, we first collect all the QA pairs tagged with SQL. We then extract the title from the QA pairs for classifying the implementation questions, as the title description has been reported as sufficient (Iyer et al., 2016). We only keep QA pairs where their question titles start with "how",[5] and not containing the following keywords *bug, error, fix* within the title content to exclude questions on error debugging. We manually inspected these QA pairs to ensure all of them to be implementation-related questions. As a result, 1,225 QA pairs in total were found.

Another issue is redundancy: 'Postlinks.xml' may also contain recurring question pairs from SO, asking the same questions in lexically different forms (or paraphrased) (Chen and Kim, 2015). We found 19 such recurring questions, and we union them to the previously mentioned 1,225 QA pairs. A total of 1,244 implementation question QA pairs were collected.

We observed that the queries in our dataset mostly involve complex conditions such as the combination of multiple joined tables with ordering and grouping clauses that are not supported by existing NL2pSQL approaches. A majority of the natural text contains 11 to 40 tokens, with SQL queries containing 11 and 15 tokens. To gain more insights on the difficulty of SQL queries, we measured Halstead complexity metrics (Halstead et al., 1977) on the training, development, and test sets and found most queries fall under the complexity score of 0-5 and well-split with near-identical distributions across the three splits.

3.2 Model: NL2pSQL with Denoising

While semantic parsing approaches require the machine-executable code for training (as outlined in Section 2), we focus on a new generative model without executability requirements for our proposed scenario. Essentially, our model consists of two layers of autoencoder architecture (Figure 1). The first layer is for generating SQL code given a natural language as input (NL2pSQL) through an autoencoder. The input to the first autoencoder is the word embedding of the natural language.

However, this model suffers a well-known problem of low-quality generation, such as token repetitions (Holtzman et al., 2018), or "ill-formed". Though our approach does not require a code segment to be machine-executable, generated codes

[4]https://stackoverflow.com/questions/6308594

[5]We are aware that the same intention can be paraphrased, and we can use paraphrased patterns to increase recall (and the data size). However, we focus on maintaining a high-precision set to start with.

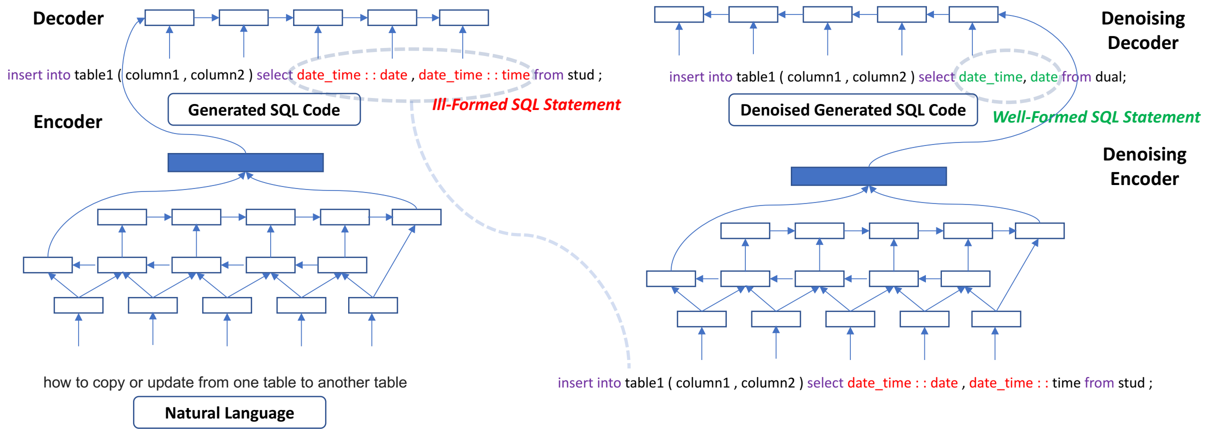

Figure 1: Overview of NL2pSQL with Denoising Autoencoder. Our model performs a two-step translation by cascading two sequence-to-sequence models. The input layers are the natural-language word embeddings in the 1st autoencoder and the SQL statement (generated by the first autoencoder) word embeddings in the 2nd autoencoder.

should be sufficiently "well-formed" to ensure human readablity. For example, a generated code with repetitive tokens will be considered ill-formed and transformed into its well-formed counterpart (or, denoising). Our contribution is thus to denoise the ill-formed queries by employing a subsequent autoencoder architecture. The input to the second autoencoder is the word embeddings of the generated SQL statements from the first autoencoder. This denoising functionality is included in the second layer with the goal of producing well-formed SQL statements.

For training the denoiser, we propose to synthesize a pair of ill- and well-formed SQL statements from the given dataset: <a *well-formed SQL statement*, an *ill-formed SQL statement*> as the input to the denoising autoencoder module. Formally, we define a well-formed SQL statement as a valid SQL statement recognized by an SQL parser whereas an ill-formed SQL statement is detected as an erroneous SQL statement construct.

We first collect well-formed SQL statements among SQL statements written by developers in SO using MySQL parser (GuduSoftware, 2018), as MySQL is the most popular SQL dialect among developers (StackOverflowSurvey, 2018). Next, we generate its ill-formed counterpart, by perturbing each of the collected well-formed SQL statements. Specifically, we randomly decide to inject into or remove from the well-formed SQL statement a token for k iterations to generate multiple ill-formed statements for a given well-formed SQL statement. In our model, we set k to be four.

To the best of our knowledge, this is the first NL2pSQL model that is not restricted to the executability from specified questions. This work is also the first in attempting to fix (denoise) ill-formed SQL statements through a neural network-based autoencoder architecture.

3.3 Evaluation Metrics

To evaluate the generated SQL statements, we first check if the generated SQL statements are well-formed (Section 3.3.1). In addition, we propose two new BLEU variants. Previous studies have reported that the BLEU score may be inappropriate in measuring the correctness of the generated code against the target code (Lin et al., 2018; Zhong et al., 2017). This is because a generated code may be written differently but still perform the intended function. The first example in Table 6 illustrates this problem. The generated SQL statement performs the semantics as required in the natural language but is structurally different as compared to the ground truth (GT). By using the vanilla BLEU, it produces a much heavily penalized score of 0.065. Thus, in our study, we use two modified versions of BLEU, namely Canonical-BLEU and SQL-BLEU, as will be described in Section 3.3.2.

3.3.1 Well-Formed SQL Statements

To evaluate the effectiveness of our denoising autoencoder, we compare the relative number of well-formed SQL statements between the generated SQL statements before and after denoising. Whether a particular SQL statement is well-formed is determined by using a SQL parser.

3.3.2 Canonical-BLEU and SQL-BLEU

Here, we first motivate the need for introducing two new BLEU variants, namely Canonical-BLEU and SQL-BLEU, before describing them in more detail. Canonical-BLEU first abstracts the identifiers of the SQL statement before calculating the BLEU score, while SQL-BLEU only considers the SQL keywords within the SQL statement when computing the BLEU score.

We note that two queries with the same semantics may have syntactically different forms. For example, a join statement (**S1**) such as "*select a.car from a,b where a.id=b.id*" can be equivalent to a nested select statement (**S2**) such as "*select a.car from a where a.id in (select b.id from b where b.name='Mary')*". When **S1** is generated and the ground truth is a similar join statement, it will have a correct and high SQL-BLEU score, yet having lower computing complexity. When the ground truth is similar to **S2**, SQL-BLEU and Vanilla BLEU would penalize more heavily than our Canonical-BLEU. In this respect, both SQL-BLEU and Canonical-BLEU complement each other and should be used together.

Canonical-BLEU Generated SQL statements may have different identifiers such as variable and table names, as compared to the original (ground truth) SO SQL statements. The difference in the identifiers' name between the generated SQL statements and the ground truth should not be heavily penalized if their SQL forms are semantically the same. To mitigate this issue, we first "canonicalize" all the identifier names (e.g., MY_COLUMN) into indexed placeholders (e.g., var_1). This transformation is performed both on the ground truth SQL statements and the generated SQL statements, and their BLEU scores are calculated. We call this Canonical-BLEU.

SQL-BLEU In a SQL statement, several words are considered keywords that serve as particular functions. For example, *SELECT* and *UPDATE* are both keywords in the SQL language. *SELECT* is used for retrieving a record from a relational database table whereas *UPDATE* is used to modify the content of a record. In SO, developers are free to adopt and use different column or table names. However, despite that, if the SQL code is about the retrieval of records, there should have a *SELECT* keyword within the SQL statement. Furthermore, we analyze the syntactically correct answers from SO between the period of 2008 and 2017 and found that 362 distinct SQL keywords (e.g., *YEAR, DATABASES, USING, UNIQUE, SELECT, UPDATE, TRUNCATE, etc*) are used, and 19% of the answers (that are well-formed) do not contain the *SELECT* keyword. We also found that multiple different variations of retrieval-type answers (i.e., 348 different keywords are used together with *SELECT*) use different combinations of keywords such as *WHEN, EXIST, INNER JOIN, IN, GROUP BY, etc*. Thus, we decided to use SQL keywords as another form of measure.

We first transform the generated SQL statements into their token types using a SQL parser. Each word will have its corresponding token type. For example, the word *SELECT* will be tagged as a *keyword* by the SQL parser. We remove all words that are non-keywords and only those words tagged as keywords are left in the SQL statements (we preserve their word ordering) to calculate the BLEU score. This transformation is performed both on the ground truth SQL code as well as the generated SQL statements and their BLEU scores are calculated. We call this SQL-BLEU.

4 Experiment Setup

4.1 Data Description

The used dataset is randomly split into train, dev and test sets in the ratio 60:20:20 to be mutual exclusive.

4.2 Baseline and Backbone Models

We stress that our work is focused on under-specified NL2pSQL and is designed for users to read/adjust the missing context for their specific usage. This is the first work on under-specified NL2pSQL and is mostly a resource and new problem/metric definition paper. Note that existing NL2pSQL work is all fully-specified and cannot be used for comparison as they involve semantic parsing and query execution on a valid database schema, which is absent in our context. In addition, making the system fully-specified is beyond the scope of this paper.

We conduct experiments for NL2pSQL generation with denoising on two neural machine translation models, Seq2Seq and CopyNet, which have shown competitive performance on natural-language-to-code translation (Lin et al., 2018). In addition, we carry out experiments in both token and character levels on both of these models, denoted as Seq2Seq(Token), Seq2Seq(Char), Copy-

Net(Token), CopyNet(Char). For each model, we perform a two-step translation by cascading two sequence-to-sequence models, for pSQL generation and denoising respectively. Each model is trained with the following hyperpameters: [6]

- Seq2Seq (Cho et al., 2014) encodes questions and decodes into queries using recurrent neural networks (RNNs). The encoder dimension is 200 by using a bi-directional RNN with concatenation. The dimension of the decoder RNN is set as 100. We optimized our objective function with Adam (Kingma and Ba, 2015), using the default momentum hyperparameters. We set the initial learning rate as 0.0001 and the mini-batch size as 32. We also used variational RNN dropout (Gal and Ghahramani, 2016) with a dropout rate of 0.4. During beam-search decoding (Hwang and Chang, 2007), we set the beam size as 100, and we ran for 100 epochs to train the model. For beam size, it has been reported that a larger beam size will lead to a higher translation quality at the expense of the decoder speed (Freitag and Al-Onaizan, 2017). We wanted to find a balance between translation quality and decoding speed. We tested different beam sizes and an entire decoding process to generate a SQL statement takes only a few seconds when the beam size is set to 100.

- CopyNet (Gu et al., 2016) extends the model above, to selectively "copy" some input sequence into output, using the same hyperparameters discussed above.

- The information retrieval (IR)-based baseline is a Lucene-based code retrieval tool, widely adopted for code search such as Sourcerer (Linstead et al., 2009). It retrieves top-3 matches, based on answer titles, matching the given question.

All the experiments were performed based on NAVER Smart Machine Learning (NSML) (Kim et al., 2018; Sung et al., 2017).

[6] Following description is for generation, and when the same model is used for denoising, we make a minor change: mini-batch size (set as 16 rather than 32), and the beam size (set as 10 instead of 100), due to a relatively large size of the dataset. Character-level encoding for denoising is unreported, as it is empirically inferior to that of token-level, as argued in prior work (Lin et al., 2018) as well.

5 Results

This section shows the quantitative (Sections 5.1 and 5.2) as well as qualitative analysis (Section 5.3) from our experiments.

5.1 Well-Formed SQL Statements

Table 1 shows the change in the number of SQL statements determined as well-formed by the SQL parser, after going through our denosing autoencoder module trained with a Seq2Seq and a CopyNet model.

Our module certainly increases the number of well-formed SQL statements. For example, when paired with a *Seq2Seq denoising autoencoder* for the top-1 prediction, the number of well-formed statements increases from 47 to 66 (19% increase). Such increase ranges from 12% to 19%, from 14% to 19%, from 30% to 43%, and from 0% to 7%, in Seq2Seq (Token), CopyNet (Token), Seq2Seq (Char), and CopyNet (Char), respectively. When paired with a *CopyNet Denoising autoencoder*, such increase is less significant. When comparing token- and character-level generation after denoising, we can see that token-level generation shows higher increase, as shown in **bold**.

5.2 Comparison of Canonical and SQL-BLEU

Here, we report the mean scores of both Canonical-BLEU and SQL-BLEU over all the generated SQL statements in the test set (including their denoised version) using the Seq2Seq and CopyNet autoencoder models in both the token and character levels.

5.2.1 Mean SQL-BLEU

Tables 2 and 3 show the changes in the mean SQL-BLEU scores on the first three generated SQL statements, after using denoising autoencoders trained with a Seq2Seq and a CopyNet model, respectively.

For example, when paired with a Seq2Seq denoising autoencoder for the top-1 prediction, the mean SQL-BLEU scores has a negligible decrease of 0.01 (from 0.28 to 0.27). Such decrease ranges from 0.01 to 0.01, from 0.01 to 0.01, from 0.02 to 0.03, and from 0.02 to 0.02, in Seq2Seq(Token), CopyNet (Token), Seq2Seq (Char), and CopyNet(Char), respectively. When paired with a *CopyNet Denoising autoencoder*, we observed similar negligible decrease in the

Table 1: % of well-formed, generated SQL queries using a Seq2Seq/CopyNet & after denoising autoencoder models. P1, P2, and P3 correspond to the first three predictions. Columns (2, 4, 6, 8) & (3, 5, 7, 9) show the increased in well-formed queries after going through a Seq2Seq & CopyNet denoising autoencoder respectively.

	P1		**P2**		**P3**		**Top 3 (%)**	
	Seq2Seq	CopyNet	Seq2Seq	CopyNet	Seq2Seq	CopyNet	Seq2Seq	CopyNet
Seq2Seq (Token)	47 → **66**	47 → 50	37 → **49**	37 → 40	33 → **52**	33 → 35	59 → **80**	59 → 64
CopyNet (Token)	49 → **65**	49 → 51	40 → **54**	40 → 40	33 → **52**	33 → 33	61 → **80**	61 → 64
Seq2Seq (Char)	31 → **74**	31 → 35	29 → **64**	29 → 31	33 → **63**	33 → 33	40 → **79**	40 → 43
CopyNet (Char)	29 → **35**	29 → 10	27 → **34**	27 → 15	32 → **32**	32 → 14	40 → **45**	40 → 31

Table 2: Negligible change in mean SQL-BLEU score & changes in delta after Seq2Seq denoising in ().

	P1	**P2**	**P3**
Seq2Seq (T)	0.28 (-0.01)	0.30 (-0.01)	0.29 (-0.01)
CopyNet (T)	0.30 (-0.01)	0.30 (-0.01)	0.28 (-0.01)
Seq2Seq (C)	0.26 (-0.03)	0.26 (-0.02)	0.26 (-0.02)
CopyNet (C)	0.18 (-0.02)	0.18 (-0.02)	0.17 (-0.01)
IR	0.29	0.29	0.27

Table 3: Negligible change in mean SQL-BLEU score & changes in delta after CopyNet denoising in ().

	P1	**P2**	**P3**
Seq2Seq (T)	0.28 (-0.01)	0.30 (-0.01)	0.29 (0.00)
CopyNet (T)	0.30 (-0.01)	0.30 (-0.01)	0.28 (0.00)
Seq2Seq (C)	0.26 (0.00)	0.26 (-0.01)	0.26 (-0.01)
CopyNet (C)	0.18 (-0.02)	0.18 (-0.02)	0.17 (-0.01)
IR	0.29	0.29	0.27

Table 4: Negligible change in mean Canonical-BLEU score & changes in delta after Seq2Seq denoising in ().

	P1	**P2**	**P3**
Seq2Seq (T)	0.27 (0.00)	0.31 (-0.01)	0.31 (0.00)
CopyNet (T)	0.27 (0.00)	0.30 (+0.01)	0.31 (+0.01)
Seq2Seq (C)	0.23 (-0.01)	0.26 (-0.02)	0.29 (-0.01)
CopyNet (C)	0.29 (+0.02)	0.30 (+0.02)	0.30 (+0.03)
IR	0.23	0.23	0.24

Table 5: Negligible change in mean Canonical-BLEU score & changes in delta after CopyNet denoising in ().

	P1	**P2**	**P3**
Seq2Seq (T)	0.27 (0.00)	0.31 (+0.01)	0.31 (+0.01)
CopyNet (T)	0.27 (0.00)	0.30 (+0.01)	0.31 (0.00)
Seq2Seq (C)	0.23 (+0.01)	0.26 (0.00)	0.29 (-0.01)
CopyNet (C)	0.29 (-0.01)	0.30 (0.00)	0.30 (-0.01)
IR	0.23	0.23	0.24

mean SQL-BLEU after denoising. Compared to IR approach, token level approaches mostly yield higher SQL-BLEU scores.

5.2.2 Mean Canonical-BLEU

Table 4 and 5 show the change in the mean Canonical-BLEU scores on the first three generated SQL statements, after using denoising autoencoders trained with a Seq2Seq and a CopyNet model respectively.

For example, when paired with a Seq2Seq denoising autoencoder for the top-1 prediction, the mean Canonical-BLEU scores has a negligible decrease in Seq2Seq (Token), from 0.00 to 0.01, or 0.01 decrease. Such decrease ranges from 0.00 to 0.02, from 0.00 to 0.01, from 0.01 to 0.02, and from 0.02 to 0.03, in Seq2Seq(Token), CopyNet (Token), Seq2Seq (Char), and CopyNet(Char), respectively. When paired with a *CopyNet Denoising autoencoder*, we observed similar negligible decrease in the mean SQL-BLEU after denoising. Compared to the IR approach, we observed that most of the generative approaches yield higher Canonical-BLEU scores.

In Section 5.1, we previously reported up to 43% more well-formed SQL statements in the denoised version than the non-denoised version. This suggests that the two metrics (*Well-Formed SQL Statements and the two variants of BLEU – Canonical BLEU and SQL BLEU*) we proposed are complementary, in that Canonical-BLEU and SQL-BLEU alone cannot distinguish well-formed SQL statements. We also note that for denoising, while enhancing one metric, it does not hurt another.

5.3 Qualitative Analysis

Table 6 shows the output examples generated by our models in the unseen data (the test set), including the denoised version. We only show the generated outputs from the Seq2Seq model as they yield the best results in our experiments. Each example consists of the natural language input, the ground truth, the top prediction of the generated output from the Seq2Seq model, and its denoised version.

The first example is about copying or updating from one table to another. In the generated SQL statement, it performs the intended copying function but is syntactically incorrect in the use of

date_time. The denoised version fixes it by replacing *date_time : : date* and *date_time : : time* with proper column names. Furthermore, the ground truth is coupled with comments, and its SQL statement is much longer than the generated version. The retrieved statement by the IR baseline has similar semantics on observing the same type of an insertion query with our approach. Although both of them are similar, the IR baseline appears more complicated as it contains more columns for insertion and a nested table structure. We believe that the generated SQL statement is a much better choice in this case.

The second example is about counting the number of rows. The generated SQL statement shows the counting-related code. Interestingly, the IR approach produces the exact query. In the last example, it is about inserting a record from another table. Similar to the IR approach, the generated SQL statement is able to churn out a query performing an insertion operation from another table. Both generated SQL statements from the last two examples do not contain any syntax errors, and the denoised autocoders did not change the SQL statements. In the last example, it is difficult to include "*top500*" which requires verbal explanation of defining one's own function in the "*order by*" clause. This part was omitted, but all the other semantics are fully preserved.

6 Conclusions

This paper addressed the problem of generating human-readable pseudo SQL queries from natural-languge questions without code-executability restriction, and propose a new dataset. Despite excluding executability, our task covers much more diverse issues including table manipulation and performance-related problems beyond simple table lookup. This coverage allows our task to help software engineers to deal with various practical implementation and operation issues. In addition, we describe a new NL2pSQL model and two evaluation metrics suitable for dealing with our task. Our result shows that adding a denoising autoencoder module enables our model to generate a more number of well-formed queries, to complement a known weakness of neural code generation.

Table 6: Generated examples

NL	how to copy or update from one table to another table
GT	- - 1) insert new record insert into old_table (id , column) select n . id , n . column from new_table n left join old_table o on n . id = o . id where o . id is null - - 2) update existed record update o set column = n . column from old_table o join new_table n on n . id = o . id
Seq2Seq	insert into table1 (column1 , column2) select date_time : : date , date_time : : time from stud ;
Seq2Seq (D)	insert into table1 (column1 , column2) select date_time , date from dual ;
IR	insert into test1 . dbo . bcd (col1 , col2 , col3) select col1 , col2 , col3 from test . dbo . abc
NL	how can i count the number of rows in a database for different time periods
GT	declare @ interval int set @ interval = 5 select datepart (hh , datetimecolumn) , datepart (mi , datetimecolumn) / @ interval * @ interval , count (*) from thetable group by datepart (hh , datetimecolumn) , datepart (mi , datetimecolumn) / @ interval * @ interval
Seq2Seq	select count (*) from (select distinct * from table_name) t
Seq2Seq (D)	select count (*) from (select distinct * from table_name) t
IR	select count (*) from (select distinct * from table_name) t
NL	how can i select top 500 rows from table1 and insert them to table2
GT	insert into emptytable select top 500 * from populatedtable order by what - - need this to define top 500
Seq2Seq	insert into destination (col1 , col2 , col3 , col4 , col5) select ' static_value1 ' , ' static_value2 ' , col6 , col7 , col8 from source_table
Seq2Seq (D)	insert into destination (col1 , col2 , col3 , col4 , col5) select ' static_value1 ' , ' static_value2 ' , col6 , col7 , col8 from source_table
IR	insert into temp (time , name) select getdate ( ) , name from name

7 Acknowledgement

Hwang was partly supported by IITP funded by MSIT (2017-0-0177; XAI).

References

Matej Balog, Alexander L Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. 2017. Deepcoder: Learning to write programs. In *International Conference on Learning Representations*.

Tony Beltramelli. 2017. pix2code: Generating code from a graphical user interface screenshot. *arXiv preprint arXiv:1705.07962*.

Florin Brad, Radu Iacob, Ionel Hosu, and Traian Rebedea. 2017. Dataset for a neural natural language interface for databases (nnlidb). *arXiv preprint arXiv:1707.03172*.

Antoni Buades, Bartomeu Coll, and J-M Morel. 2005. A non-local algorithm for image denoising. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 60–65. IEEE.

Fuxiang Chen and Sunghun Kim. 2015. Crowd debugging. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 320–332, New York, NY, USA. ACM.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. Association for Computational Linguistics.

Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. 2007. Image denoising by sparse 3-d transform-domain collaborative filtering. *IEEE Transactions on image processing*, 16(8):2080–2095.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany. Association for Computational Linguistics.

Michael Elad and Michal Aharon. 2006. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image processing*, 15(12):3736–3745.

Catherine Finegan-Dollak, Jonathan K Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-sql evaluation methodology.

Markus Freitag and Yaser Al-Onaizan. 2017. Beam search strategies for neural machine translation. *arXiv preprint arXiv:1702.01806*.

Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027.

Alessandra Giordani and Alessandro Moschitti. 2012. Translating questions to sql queries with generative parsers discriminatively reranked. *Proceedings of COLING 2012: Posters*, pages 401–410.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1631–1640. Association for Computational Linguistics.

GuduSoftware. 2018. Sql parse, analyze, transform and format all in one.

Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish Shevade. 2017. Deepfix: Fixing common c language errors by deep learning. In *AAAI*, pages 1345–1351.

Maurice H Halstead et al. 1977. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., New York, NY.

Charles T Hemphill, John J Godfrey, and George R Doddington. 1990. The atis spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.

Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. 2018. Learning to write with cooperative discriminators. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1638–1649. Association for Computational Linguistics.

Seung-won Hwang and Kevin Chang. 2007. Probe minimization by schedule optimization: Supporting top-k queries with expensive predicates. In *IEEE TKDE*.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973, Vancouver, Canada. Association for Computational Linguistics.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2073–2083. Association for Computational Linguistics.

Hanjoo Kim, Minkyu Kim, Dongjoo Seo, Jinwoong Kim, Heungseok Park, Soeun Park, Hyunwoo Jo, KyungHyun Kim, Youngil Yang, Youngkwan Kim,

et al. 2018. Nsml: Meet the mlaas platform with a real-world case study. *arXiv preprint arXiv:1810.09957*.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization.

Kazuhei Kouno, Hiroyuki Shinnou, Minoru Sasaki, and Kanako Komiya. 2015. Unsupervised domain adaptation for word sense disambiguation using stacked denoising autoencoder. In *Proceedings of the 29th Pacific Asia Conference on Language, Information and Computation: Posters*, pages 224–231.

Carolin Lawrence and Stefan Riezler. 2016. Nlmaps: A natural language interface to query openstreetmap. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations*, pages 6–10.

Fei Li and Hosagrahar V Jagadish. 2014. Nalir: an interactive natural language interface for querying relational databases. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 709–712. ACM.

Yunyao Li, Huahai Yang, and HV Jagadish. 2006. Constructing a generic natural language interface for an xml database. In *International Conference on Extending Database Technology*, pages 737–754. Springer.

Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. 2018. Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation LREC 2018, Miyazaki (Japan), 7-12 May, 2018.*

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, Fumin Wang, and Andrew Senior. 2016. Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 599–609. Association for Computational Linguistics.

Erik Linstead, Sushil Bajracharya, Trung Ngo, Paul Rigor, Cristina Lopes, and Pierre Baldi. 2009. Sourcerer: mining and searching internet-scale software repositories. *Data Mining and Knowledge Discovery*, 18(2):300–336.

Fan Long and Martin Rinard. 2015. Staged program repair with condition synthesis. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 166–178. ACM.

Sergey Mechtaev, Jooyong Yi, and Abhik Roychoudhury. 2016. Angelix: Scalable multiline program patch synthesis via symbolic analysis. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*, pages 691–701. IEEE.

William Meiniel, Jean-Christophe Olivo-Marin, and Elsa D Angelini. 2018. Denoising of microscopy images: A review of the state-of-the-art, and a new sparsity-based method. *IEEE Transactions on Image Processing*, 27(8):3842–3856.

Vijayaraghavan Murali, Letao Qi, Swarat Chaudhuri, and Chris Jermaine. 2018. Neural sketch learning for conditional program generation. In *International Conference on Learning Representations*.

Courtney Napoles, Keisuke Sakaguchi, and Joel Tetreault. 2017. Jfleg: A fluency corpus and benchmark for grammatical error correction. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 229–234. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Proceedings of the 20th international conference on Computational Linguistics*, page 141. Association for Computational Linguistics.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157. ACM.

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 (Volume 1: Long Papers)*, pages 1139–1149. Association for Computational Linguistics.

Senjuti Basu Roy, Martine De Cock, Vani Mandava, Swapna Savanna, Brian Dalessandro, Claudia Perlich, William Cukierski, and Ben Hamner. 2013. The microsoft academic search dataset and kdd cup 2013. In *Proceedings of the 2013 KDD cup 2013 workshop*, page 1. ACM.

StackOverflow. 2018. Stack overflow.

StackOverflowSurvey. 2018. Stack overflow survey 2018.

Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, and Ming Zhou. 2018. Semantic parsing with syntax- and table-aware sql generation. In *Proceedings of the*

56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 361–372. Association for Computational Linguistics.

Nako Sung, Minkyu Kim, Hyunwoo Jo, Youngil Yang, Jingwoong Kim, Leonard Lausen, Youngkwan Kim, Gayoung Lee, Donghyun Kwak, Jung-Woo Ha, et al. 2017. Nsml: A machine learning platform that enables you to focus on your models. *arXiv preprint arXiv:1712.05902*.

Lappoon R Tang and Raymond J Mooney. 2000. Automated construction of database interfaces: Integrating statistical and relational learning for semantic parsing. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, pages 133–141. Association for Computational Linguistics.

David HD Warren and Fernando CN Pereira. 1982. An efficient easily adaptable system for interpreting natural language queries. *Computational Linguistics*, 8(3-4):110–122.

Xiaojun Xu, Chang Liu, and Dawn Song. 2018. SQLNet: Generating structured queries from natural language without reinforcement learning. ICLR.

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. Type-and content-driven synthesis of sql queries from natural language. *arXiv preprint arXiv:1702.01168*.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018a. Typesql: Knowledge-based type-aware neural text-to-sql generation. *arXiv preprint arXiv:1804.09769*.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018b. Syntaxsqlnet: Syntax tree networks for complex and cross-domaintext-to-sql task.

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

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055.

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

Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. 2017. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155.

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