ABSTRACT
Learning the mapping between natural language (NL) and programming language, such as retrieving or generating code snippets based on NL queries and annotating code snippets using NL, has been explored by lots of research works [2, 19, 21]. At the core of these works are machine learning and deep learning models, which usually demand for large datasets of <NL, code> pairs for training. This paper describes an experimental study of StaQC [50], a large-scale and high-quality dataset of <NL, code> pairs in Python and SQL domain, systematically mined from the Stack Overflow forum (SO). We compare StaQC with two other popular datasets mined from SO on the code summarization task, showing that StaQC helps achieve substantially better results, improving the current state-of-the-art model by an order of 8% ~ 9% in BLEU metric.

CCS CONCEPTS
• Information systems → Web searching and information discovery; • Software and its engineering → Documentation; • Computing methodologies → Neural networks;

KEYWORDS
Code Summarization; Question-Code Pairs; Web Mining; Deep Neural Networks; Stack Overflow

1 INTRODUCTION
Stack Overflow (SO) [41] as a website has helped software development greatly over the past few years. The understanding and re-usability of crowd sourced programming solutions can help improve the overall process of software development. Over the past few years, great research has been performed towards generation or retrieval of code snippets from a natural language description [1, 24, 28, 38, 53] and summarizing code snippets using natural language [1–3, 10, 19, 21, 47]. The underlying machine learning models for these tasks fall into the category of deep learning models [13], which are inherently data hungry. Generating large datasets containing high quality code snippets paired with a natural language description from SO becomes an important task in building many such data-hungry downstream applications for software engineering.

Figure 1 shows an example question “Elegant Python function to convert CamelCase to snake_case?” on SO. In the figure, $S_i$ ($i = 1, 2, 3, 4$) and $C_j$ ($j = 1, 2, 3, 4$) denote sentence blocks and code blocks respectively [50].
In fact, multi-code answer posts are very common in SO, which makes the low-precision and low-recall issues even more prominent when we need to create large-scale <NL, code> datasets. To address these problems, Yao et al. [50] proposed a novel Bi-View Hierarchical Neural Network model (i.e., BiV-HNN) to systematically mine question-code pairs from multi-code question posts, with high precision and recall. The BiV-HNN model is able to capture features from both the textual context as well as the programming content of each code snippet, which are combined into a deep neural network architecture to predict whether this code snippet is a standalone solution or not. The model substantially outperforms heuristic approaches by more than 15% in F1 and accuracy, and is then applied to automatically mine code solutions for SO questions in Python and SQL domains, which results in StaQC, comprising ~148K Python and ~120K SQL <NL, code> pairs. According to Yao et al. [50], models trained on StaQC for code retrieval task [4, 21, 25] achieved impressive improvement in Mean Reciprocal Rank [46] of about 6% over the baseline dataset [21]. The experiment showed a glimpse into the potential of StaQC to improve state-of-the-art performance on downstream tasks of software engineering without considerable modifications to the model, which is an interesting research question to explore, "Can StaQC show similar gains in performance on other downstream applications in Software Engineering?"

In this paper, we investigate the task of Code Summarization, which is to automatically generate a natural language summary for a given code snippet. We test StaQC [50] against CODE-NN, a heuristically mined dataset presented in [21] and CoNaLa [52], which also utilizes machine learning models to systematically collect question-code pairs from SO, on Code Summarization task. We compare the performances of two code summarization models [19, 21] for SQL and Python programming languages across different datasets. Through experiments, we show that StaQC helps achieve significantly better results on SQL code summarization, improving the current state-of-the-art model by 8% ~9% in BLEU metric. We also show that StaQC consistently performs better on Python code summarization, when compared to CoNaLa.

**Paper organization.** The remainder of this paper is organized as follows. Section 2 presents statistical analysis of the CODE-NN dataset, the CoNaLa dataset and the StaQC dataset. Section 3 provides brief descriptions on the code summarization models. Section 4 presents the experimental setup and results for the code summarization models. Finally, Section 5 concludes the paper and points out potential future directions.

## 2 STATISTICAL ANALYSIS OF THE DATASETS

### 2.1 CODE-NN Dataset

The CODE-NN dataset is heuristically mined from SO by Iyer et al. [21], containing <NL, code> pairs of SQL and C# domains. In order for a straightforward comparison with StaQC (which covers SQL and Python domains), in this paper, we focus on the SQL dataset in CODE-NN. The authors first extracted SO question posts tagged by "sql", "database" or "oracle" from the archived Stack Exchange Dump to be in SQL domain. Among them, the authors only considered questions whose accepted answer post contains exactly one code snippet. The CODE-NN dataset was then generated by pairing the question title with the one code snippet in its accepted answer post. The dataset is further processed using a semi-supervised bootstrap approach to filter titles that bear no relation to the corresponding code snippet [21]. The final cleaned SQL dataset contains 32,337 <question, code> pairs. Complete statistics of the CODE-NN dataset are provided in Table 1.

### 2.2 StaQC Dataset

For larger scale and higher quality, the StaQC dataset is systemically mined from SO by Yao et al. [50], covering Python and SQL domain. Similar to CODE-NN, the authors identified SQL and Python posts by their tags on SO. They further cleaned them by using a supervised binary classifier to select only the "how-to-do-it" questions, since answers to other types of questions are not very likely to be standalone code solutions. From this cleaned dataset, answer posts containing exactly one code snippet are directly mined similar to CODE-NN.

To extract code solutions from multi-code answer posts (i.e., answer posts containing multiple code snippets), Yao et al. [50] proposed a novel Bi-View Hierarchical Neural Network (BiV-HNN) model. The BiV-HNN model consists of two different modules that capture features from the textual contexts and the code content of a code snippet, and combines them into a deep neural network architecture, which finally predicts whether a code snippet is a standalone solution or not. The model was shown to outperform both the widely adopted heuristic methods (e.g., pairing the question title with the first code snippet or all code snippets in its answer post) and traditional classifiers (e.g., Logistic Regression and Support Vector Machines) by more than 15% higher F1 and accuracy in identifying code solutions. The final mined StaQC dataset contains 119,519 SQL domain and 147,546 Python domain <question, code> pairs, making it the largest-to-date dataset for SQL Domain. Figure 2 shows an example of two code solutions in the StaQC dataset, which are mined from one multi-code SQL answer post. Complete statistics of the StaQC dataset are provided in Table 1.

### 2.3 CoNaLa Dataset

Similar to StaQC, CoNaLa [52] is also extracted by a machine learning model from SO, resulting in the largest-to-date dataset of 598,237 question-code pairs in Python domain. As in the two previous work [21, 50], Yin et al. [52] collected SO Python questions and filtered down to only "how-to" questions from them. Different from [50], they considered every line (or fragment) in a code block as a candidate code solution, and built a logistic regression classifier to decide whether the candidate aligns well with the question title or not. The classifier utilizes two kinds of features to capture both the syntactic and the semantic information in a code candidate, leading
Table 1: Statistics of datasets.

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>Dataset</th>
<th># of QC pairs</th>
<th>Question Average length</th>
<th># of tokens</th>
<th>Code Average Length</th>
<th># of tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL</td>
<td>CODE-NN</td>
<td>32,337</td>
<td>9</td>
<td>10,086</td>
<td>27</td>
<td>141,018</td>
</tr>
<tr>
<td></td>
<td>StaQC</td>
<td>119,519</td>
<td>9</td>
<td>35,722</td>
<td>37</td>
<td>740,837</td>
</tr>
<tr>
<td>Python</td>
<td>CoNaLa</td>
<td>598,237</td>
<td>9</td>
<td>25,582</td>
<td>8</td>
<td>394,777</td>
</tr>
<tr>
<td></td>
<td>StaQC</td>
<td>147,546</td>
<td>9</td>
<td>59,906</td>
<td>39</td>
<td>1,171,117</td>
</tr>
</tbody>
</table>

Figure 2: An example of two code solutions in StaQC [50], which were mined from a multi-code answer post.

A dataset with rich surface variations is beneficial for the development of complex deep learning models [13]. When a model does not observe certain data patterns in the training phase, it becomes less capable to predict them during testing. StaQC can alleviate this issue by enabling a model to learn from alternative code solutions to the same question. Owing to its large scale and diversity, Yao et al. [50] showed that a model trained on StaQC dataset can outperform the one trained on the CODE-NN dataset by 6% Mean Reciprocal Rank [46] on the code retrieval task. In this paper, we further demonstrate the strength of StaQC through experiments on the code summarization task.

3 CODE SUMMARIZATION MODELS

3.1 Background

Code Summarization is a task to generate a natural language summary given a code snippet. Current state-of-the-art architectures for code summarization model are built upon the recent advancements in deep recurrent neural network (RNN) [12] architectures. Specifically, Long Short-Term Memory (LSTM) [11, 18] has become the core component of many tasks involving sequential data due to its capability to learn long-term dependencies. It has also been used to better handle long dependencies in source code (e.g., a Python function is used far away from its definition) [19]. The details of RNN and LSTM are shown in Figure 3a and 3b respectively.

3.1.1 Recurrent Neural Networks. RNNs are intimately related to sequences and lists because of their chain-like architecture. As illustrated in Figure 3a, At each time step t, the RNN cell takes as input, the current input token as well as the previous hidden state outputted by its previous time step t − 1 and updates the current hidden state namely, \( h_t = \tanh(W_x t + U h_{t-1} + b) \) where \( W, U \) and \( b \) are the trainable parameters and \( \tanh \) is the activation function [19]. Though, in theory, RNNs are capable to capture long-distance dependencies, in practice, they fail due to the gradient vanishing/exploding problems [6, 36].

3.1.2 Long Short-Term Memory. LSTM introduces a structure called "memory cell" to solve the problem. Basically, a LSTM unit is composed of three multiplicative gates which control the proportions of information to forget and to pass on to the next time step. Figure 3b gives the basic structure of an LSTM unit. The LSTM is trained to selectively "forget" information from the hidden states, thus allowing room to take in more important information [18].
Table 2: Question - # of code distributions.

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>Dataset</th>
<th># of code solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>SQL</td>
<td>StaQC</td>
<td>86,788</td>
</tr>
<tr>
<td></td>
<td>StaQC-multi</td>
<td>11,266</td>
</tr>
<tr>
<td>Python</td>
<td>CoNaLa</td>
<td>2,264</td>
</tr>
<tr>
<td></td>
<td>StaQC</td>
<td>108,653</td>
</tr>
<tr>
<td></td>
<td>StaQC-multi</td>
<td>23,403</td>
</tr>
</tbody>
</table>

3.1.3 Language Models. Language models have been successfully applied to solve a variety of problems in NLP, e.g., machine translation [5, 48] and etc., leading to state-of-the-art performances.

LSTM has been widely used to solve semantically related tasks like speech recognition [14], sequence tagging [9, 20, 32], machine translation [5, 48] and etc., leading to state-of-the-art performances.

LSTM’s have become an important building block for language models due to their capability to remember long range dependencies [23, 26, 33, 42, 43].

\[
P(x) = P(x_1|x_2)...P(x_n|x_1...x_{n-1})
\]

3.2 Codenn

Codenn summarization model, published in [21], uses an end-to-end generation system to perform content selection and surface realization jointly. The core component of Codenn is a LSTM-based recurrent neural network with an attention mechanism [14, 30], which models the probability of a natural language summary conditioned on the given code snippet, as shown in Figure 4. Formally, given a NL summary \( n = n_1, ..., n_l \), each word \( n_i \) is represented by
a 1-hot vector $n_i \in \{0, 1\}^{|N|}$, where $N$ is the vocabulary size. The model computes the probability of $n$ as a product of the conditional next-word probabilities as shown in Eq 2:

$$p(n|n_1, ..., n_{i-1}) = \prod_{i=1}^{l} p(n_i|n_1, ..., n_{i-1})$$

with

$$p(n_i|n_1, ..., n_{i-1}) = W \tanh(W_1 h_i + W_2 t_i)$$

where $W \in \mathbb{R}^{N \times |H|}$ and $W_1, W_2 \in \mathbb{R}^{H \times H}$ are trainable parameters, and $H$ is the embedding dimensionality of the summaries. $h_i$ is the LSTM hidden state at time step $i$, $t_i$ is the contribution from the attention model on the source code. The generation of each word is guided by a global attention model [30], which computes a weighted sum of the embeddings of the code snippet tokens based on the current LSTM state. A code snippet $c$ is represented as a set of 1-hot vectors $c_1, ..., c_K \in \{0, 1\}^{|C|}$, where $C$ is the vocabulary of all tokens in the code snippets. The attention model computes,

$$t_i = \sum_{j=1}^{k} \alpha_{i,j} \cdot c_j$$

where $F \in \mathbb{R}^{1 \times |H|}$ is a token embedding matrix and each $\alpha_{i,j}$ is the attention weight for each code token $c_j$ w.r.t current hidden state $h_i$:

$$\alpha_{i,j} = \frac{\exp(h_i^T c_j F)}{\sum_{j=1}^{k} \exp(h_i^T c_j F)}$$

### 3.3 DeepCom

DeepCom [19] is built upon advances in Neural Machine Translation (NMT). Typical NMT aims to automatically translate from one language to another language [5, 44]. Intuitively, Hu et al. [19] considered generating NL summaries or comments as a variant of the NMT problem, where source code written in a programming language needs to be translated to text in natural language. Compared with Codenn which only builds a LSTM based language model for NL summaries, the NMT model builds language models for both source code and summaries.

Specifically, DeepCom models both the code snippets and the text summaries as sequences, and uses the Sequence-to-Sequence (Seq2Seq) approach to learn the translation between them. Seq2Seq has been widely used for machine translation [44], text summarization [39], dialogue system [45], etc. The model consists of three components: an LSTM encoder, a LSTM decoder, and an attention component. Figure 5 illustrates the detailed Seq2Seq model.

At each time step $t$, the encoder reads one token $x_t$ of the source code sequence, then updates and records the current hidden state $s_t$. The source code features are finally encoded into a context vector $c$ through an attention mechanism [5]. Specifically, DeepCom defines individual $c_i$ for predicting each target word $i$ as a weighted sum of all hidden states $s_1, ..., s_m$ in encoder:

$$c_i = \sum_{j=1}^{m} \alpha_{ij} s_j$$
We experimented with the DeepCom model [19] for Python domain. A vocabulary for both code snippets and natural language is created, which was chosen from \{128, 512\}, and the dropout rate [40], which are kept the same as mentioned in [21], except for the LSTM hidden dimension, where the vocabulary sizes for code and natural language questions are set to 50,000 and 30,000 respectively. We used simple "space" tokenization to create vocabulary for both code snippets and natural language. The best model was selected as the one achieving the highest BLEU score on DEV set.

### 4.2 SQL-Domain Experimental Setup

For the experiments on the Codenn model and the DeepCom model for SQL domain, we keep the implementation details exactly the same as mentioned in [21] and [19], only changing the training dataset for each experimental setting. In the first and second setting, we use CODE-NN and StaQC (SQL) datasets as training data respectively. To emphasize the importance of surface variation in StaQC, we just added the 41,826 <natural language question, code solution> pairs, automatically mined from Stack Overflow SQL multi-code answer posts, to the CODE-NN dataset (i.e., CODENN-multi = CODE-NN + StaQC-multi), which becomes the training data in our third experimental setting. For all SQL domain experiments, we used the DEV (valid) set and the EVAL (test) set in [21] for development and evaluation, respectively. All questions and code snippets occurring in the these two sets were removed from training data.

For the experiments on the Codenn model, all the hyper-parameters are kept the same as mentioned in [21], except for the dropout rate [40], which was chosen from \[0.4, 0.7\] for each experimental setting, to avoid over-fitting and achieve a better performance. For the experiments on DeepCom, almost all the hyper-parameters are as mentioned in [19] except for the LSTM hidden dimension, which was chosen from \[128, 256, 512\], and the dropout rate [40], which was chosen from \[0.4, 0.7\] for each experimental setting. Hu et al. [19] use a vocabulary size of 30,000 for code snippets as well as NL summaries. Instead, we experimented with vocabulary sizes of \[30,000, 50,000\] for code snippets, in order to accommodate the considerable difference in token numbers of code and summaries, as shown in Table 1. We used simple "space" tokenization to create vocabulary for both code snippets and natural language. The best model was selected as the one achieving the highest BLEU score on DEV set.

### 4.3 Python-Domain Experimental Setup

We experimented with the DeepCom model [19] for Python domain. In the first and second experimental settings, we use StaQC-python and CoNaLa datasets as training data respectively. As the CoNaLa dataset (~600k) is more than \(3\times\) times the size of StaQC Python dataset (~148k), we perform a third experiment using CoNaLa-reduced dataset (~148k), which is a random subset of CoNaLa and has the same size as the StaQC-python dataset, in order to make a direct comparison. We randomly sampled 10% of the CoNaLa dataset as the development (DEV) set across all experiments. We test all models with the CoNaLa test set [52] (denoted as "EVAL"). All questions and code snippets occurring in the DEV and EVAL set were removed from the training data.

For all experiment, we set the the maximum length of a code sequence to 100 and the maximum question length to 30. The vocabulary sizes for code and natural language questions are set to 50,000 and 30,000 respectively. We used simple "space" tokenization to create vocabulary for both code and natural language. We experimented with different hyper-parameters: Number of LSTM layers [2, 3], hidden dimension \[128, 256, 512\] and dropout rate \[0.3, 0.8\] in each setting. The best model was selected as the one achieving the highest BLEU score on the DEV set.

### 4.4 Experimental Results

Table 3 shows the BLEU score of each SQL Code summarization model on EVAL and DEV set across different experimental settings. Across different model architectures, we can consistently observe that models trained on StaQC and CODENN-multi showed improved performance over models trained on CODE-NN. For the Codenn summarization model, we can observe an improvement of 2% in BLEU metric over the CODE-NN dataset. Specifically for DeepCom, StaQC helps achieve a substantial 8%~9% improvement in BLEU metric on the current state-of-the-art model [19].

By comparing the results of experimental setting 1 (using the CODE-NN dataset) to experimental setting 3 (using the CODENN-multi dataset), we can clearly see the importance of the mined multi-code questions. The rich surface variations in the mined multi-code questions helps improve the performance across all models. Note that the performance gains shown here are still conservative,
A Comprehensive Study of StaQC for Deep Code Summarization

since we adopted almost the same hyper-parameters and a small evaluation set, in order to see the direct impact of StaQC. Using more challenging evaluation sets and by conducting systematic hyper-parameter selection, we expect models trained on StaQC to be more advantageous.

Table 4 presents the BLEU scores for different experiments on the Python Code Summarization task. The DeepCom model trained on StaQC [50] consistently performs similar or slightly better than the model trained on CoNaLa [52], even though CoNaLa is $3 \times 3$ size of StaQC. We further make a direct comparison between StaQC and CoNaLa-reduced, which are of the same size, and observe an improvement of $\approx 2.5\%$ in BLEU metric provided by StaQC over CoNaLa-reduced. In the future, we plan to conduct more qualitative study over different experiments.

5 CONCLUSION

This paper explores an experimental study of StaQC [50], a large-scale and high-quality dataset of <natural language question, code snippet> pairs in Python and SQL domain, in comparison with two other existing datasets, CODE-NL [21] and CoNaLa [52]. We show that the systematically mined StaQC can greatly help downstream tasks aiming to associate natural language with programming language, by performing experiments on the code summarization task. In the future, we plan to perform more qualitative analysis across these datasets and also perform more experiments on other downstream code-language tasks like code generation.

ACKNOWLEDGMENTS

This research was sponsored in part by the Army Research Office for Government purposes notwithstanding any copyright notice implied, of the Army Research Office or the U.S. Government. The opinions contained herein are those of the authors and should not be attributed to the Army Research Office.

REFERENCES


