SLQA: Sequence Labeling Based Question Answering for Mining Aligned Natural Language and Programming Language Data

Anonymous ACL-IJCNLP submission

Abstract

A high-quality parallel natural language and programming language (NL-PL) corpus is essential for many downstream tasks such as code retrieval, code summary, and code synthesis. Traditional approaches to acquire this parallel corpus struggle primarily with the issues of transferbility and scalability as querying programming languages require training models separately for each programming language or handcrafting specialized representations. In particular, approaches that incorporate the programming language in the model have to incorporate a lot of domain knowledge to be able to generalize well to other languages (Yin et al., 2018). On the other hand, approaches that incorporate only natural language (Yao et al., 2018) do not fully exploit the semantic similarity between syntax of different programming languages (eg: print in Python is similar to println in Java)

We bridge this gap by treating the problem as a question answering task, and propose Sequence Labeling based Question Answering (SLQA) for this purpose. Specifically, for a question answering pair in Stack Overflow, the model analyzes the natural language and programming language in the code block of the answer post to infer whether the code block answers the question. Furthermore, since our model uses Byte Pair Encoder (BPE) tokenization, our code retrieval model exploits the subword similarity in syntax of various programming languages, allowing it to scale to multiple languages without any domain knowledge.

1 Introduction

Although semantic natural language (NL) annotation of programming language (PL) involves semi-structured data, acquiring this parallel data has not been easy. Availability of high-quality aligned natural language-programming language (NL-PL) pairs is essential for not only the implementation of downstream tasks such as code retrieval (Allamanis et al., 2015; Wei and Ch, 2015; Husain et al., 2020), code summarization (Allamanis et al., 2016; Iyer et al., 2016), code synthesis (Locascio et al., 2016; Desai et al., 2016; Quirk et al., 2015; Yin and Neubig, 2017; Clement et al., 2020), but also evaluation of these models. Traditionally, efforts in acquiring this parallel corpus was done through mining structured yet highly contextualized resources such as official documentation of programming languages found on sources like Github or unstructured but more generalized sources like StackOverflow. Current approaches (Yin et al., 2018; Iyer et al., 2016; Yao et al., 2018) struggle primarily with the issues of transferbility and scalability as querying PLs require training models separately for each programming language (Yao et al., 2018) or handcrafting specialized representations (Iyer et al., 2016; Yin et al., 2018).

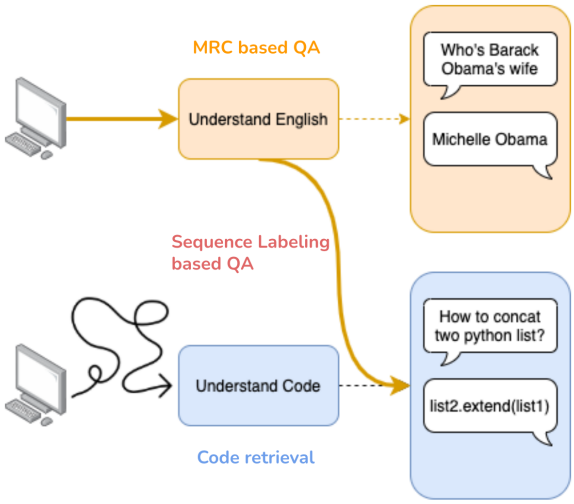

Figure 1: Our work compared to MRC based QA and code retrieval. Our model is able to mine aligned natural language and programming language pairs without understand the semantic meaning of code.

On the other hand, recent modelling efforts

in machine reading comprehension (MRC) have shown some exciting results. In particular, state of the art Question Answering (QA) models such as BiDAF (bid) and DrQA (Chen et al., 2017) have illustrated interesting approaches to identify the correct answer span within a textual context. So, in order to allow the model to effectively find NL-PL pairs, we are inspired by machine reading comprehension based question answering (Rajpurkar et al., 2016; Yang et al., 2018) and treat this problem as a question answering problem on Stack Overflow: for a given Stack Overflow post, we consider its question title as the question and the code blocks in the answer that can solve the question as the answer. While span prediction can be applied to NL-PL tasks, word level span prediction models will still require training a different model for each programming language. Sequence Labeling based Question Answering (SLQA) circumvents this problem by treating each code-block as a separate token and instead use sequence labelling to indicate whether that code-block is part of the answer span. Specifically, we use the IOB tagging format technique (Ramshaw and Marcus, 1995), which allows the model to predict multiple token spans as answers. The contributions of this work are as follows

- SLQA - A transformer based model that uses sequence labeling to identify question-code pairs
- Empirical results that show SLQA is the first NL-PL model that has shown effective transfer learning from one programming language to many other programming language
- We further present a dataset with about 800,000 NL-PL parallel corpus

2 Relevant Work

2.1 Structured Code Documentation as NL

Previous work acquired such aligned NL-PL pairs mining using weak or no supervision over official documentation. Using week or no supervision is intentional as acquiring NL-PL pairs at scale requires mining large corpus. CodeSearchNet(Husain et al., 2020) and PyMT5(Clement et al., 2020) a collection of about 2 million and 7.7 million function-documentation pairs respectively. Availability of these datasets propelled models that used natural language to query programming ones (Feng et al., 2020; Yan et al., 2020). However, unsurprisingly this dataset is noisy due to the difference in language used between the documentation and natural language. Language used in documentation is fundamentally different from natural language in that it is often written by the same author who has written the code and therefore does not differ much from the vocabulary in the code (Husain et al., 2020). Furthermore, to truly query a programming language database requires a semantic mapping between general and simple natural language and programming language, whereas documentation language is highly contextual and specific.

2.2 Unstructured Code Forums as NL

While official documentation sites like Github serve as structured resources to mine these pairs, a less structured but more relevant resource is StackOverflow. This is often done by using the question title as the query, and part of the answer as the corresponding code snippet based on a handcrafted approach (Allamanis et al., 2015; Zilberstein and Yahav, 2016). Neural Code Search (NCS) (Sachdev et al., 2018) adopted an unsupervised approach that uses Aroma (Luan et al., 2019) and StackOverflow's data for code and natural language respectively to map pairs of that are semantically similar.

More recently, neural networks were used for this task. Among them, Code-NN (Iyer et al., 2016) uses posts that has only one code block to generate code summaries based on the question title. Conala (Yin et al., 2018), on the other hand, used a combination of handcrafted rules and neural network model for mining Python and Java forums, where the handcrafted approach relies on a prior knowledge of the language, and the neural network model trains a classification model using just the code snippet to identify the line of code that is most relevant. StaQC (Yao et al., 2018) incorporated question title and local textual context and used a binary model to determine whether a code block is aligned with the surrounding textual context.

Due to the fact that Stack Overflow's data is largely unstructured, efforts in modelling using this data struggle with two main issues. First, if code information is used (Yin et al., 2018), they need to model a separate model for each programming language. Second, because StackOverflow's answer body is interspersed with code blocks along with textual data, local textual context based models (Yao et al., 2018) suffer from inability to retain alignment among the overall textual data.

2.3 Related work in other domains of NLP

In traditional MRC based QA tasks ((Rajpurkar et al., 2016), (Yang et al., 2018), etc.), there is often one correct answer span for a given question and passage. State of the art QA models such as BiDAF (bid) and DrQA (Chen et al., 2017) have used transformer based models to predict the beginning and end of the span within the passage that answers the question. Since StackOverflow's answer posts is natural language corpus interspersed with code-blocks, we have chosen to adopt sequence labelling approach (Erdogan, 2010), in particular we will be using IOB tagging. IOB tagging is widely used in fields such as named entity recognition (Lample et al., 2016), where multiple instances of data need to identified as part of an entity. Furthermore, we believe using code language along with natural language could be useful in training to track similar, but not same, words (eg: print in python is similar to println in Java). To this end, we also used Byte-Pair Encoding (BPE)(Gage, 1994) tokenization to represent our data. BPE is a sub-word tokenizing approach which is useful to map similar words

3 Preliminaries

In this section, we provide a task definition and describe our annotation approach.

3.1 Task Overview

Given a Stack Overflow question title and its accepted answer, we aim at finding consecutive code blocks in the accepted answer that can solve the question. Some solutions will be consist of multiple code blocks, and other solutions might only have a single code block. The ultimate goal of the task is to identify all potential solutions in the answer post.

3.2 Manual Annotation

Different from the annotation approach proposed in previous works (Yin et al., 2018; Yao et al., 2018), we did not use a classifier to determine whether a question is a how-to question or not, instead, our annotation is consist of two parts: first, we annotated questions with binary categories to label whether they are how-to question; second, we annotate the answer body of a how-to question with IOB labels.

During the pilot annotation study, we had all three of the annotators to annotate the same set of sample Python data extracted from the Stack Overflow data dump. We went through five iterations of the pilot study before finalizing the annotation protocol. In the final iteration, the average Cohen's kappa for how-to questions is around 0.717, and the average Cohen's kappa for IOB tagging is around 0.785.

Lang2Code	how-to		non-how-to	
	#	%	#	%
Python	1323	71.4	529	28.6
SQL	70	70	30	30
R	80	80.8	19	19.2
Git	75	75.8	24	24.2
Java	53	53.5	46	46.5
Linux	69	69.7	30	30.3

Table 1: The annotated how-to-question statistics of Lang2Code-human

After the annotation protocol has been finalized, each annotator was assigned a set of sample data for annotation. The data are mutually exclusive so that each post is only annotated once.

3.2.1 How-To Questions

A how-to question classifier is a binary classification model that classifies whether a given question post (title and question body) is a how-to question or not. This was important to ensure our dataset does not include highly specific debugging questions or very vague question posts. While we did not adopt a how-to question classifier during the annotation stage, we implemented one based on the annotated data for processing the larger Stack Overflow data dump in the future. The dataset includes 2351 hand-annotated labels for the classifier from Python, Java, SQL, R and Git related questions. The specific split of the languages are shown in Table 1. About 1100, 500 and 700 were used as training, validation and test sets respectively.

The primary evaluation metric we were concerned with while evaluating models was the precision score. As we will be using this model to classify large corpus of data whether they are how-to question or not, which we will further use to annotate answer posts, we wanted to minimize the false positive rate. We tried multiple models and have chosen pre-trained BERT-based model as our final classification model has it has performed the best in terms of precision with a precision score of 92.40% on the test set.

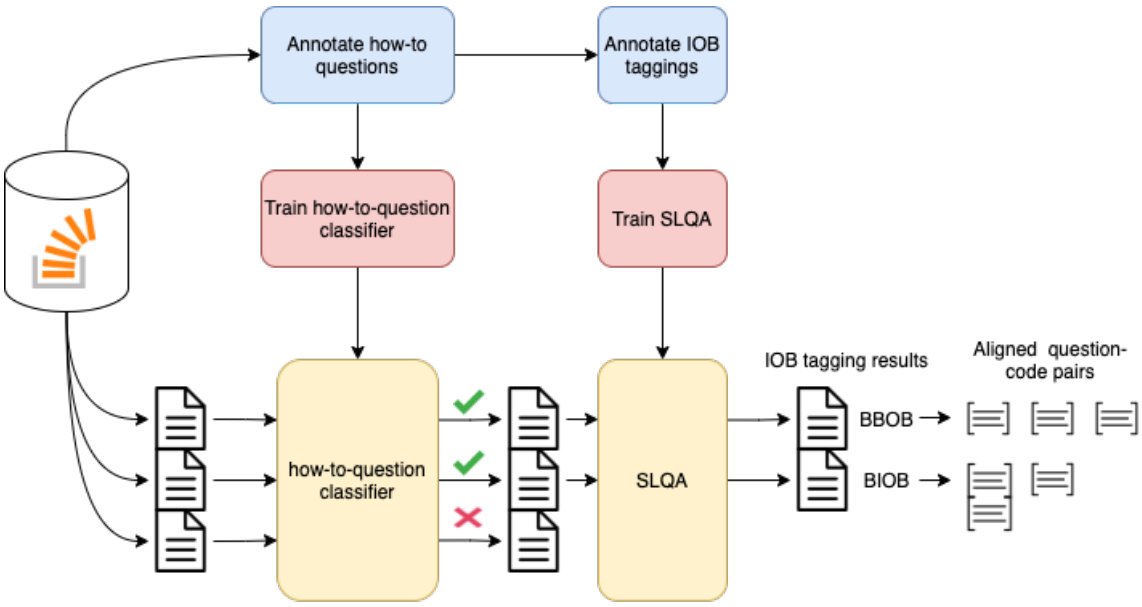

Figure 2: Overall pipeline of our work

3.2.2 IOB Tagging

Once we determined that a question is a how-to question, we annotate the code blocks in the accepted answer with IOB tagging indicating which code block(s) answer the question.

Different from the traditional IOB tagging approach in NER, where the tagging is done at the word level (Curran and Clark, 2003), we modified the approach to tag only the standalone code blocks but not the surrounding text or the in-line code blocks. We also decided to ignore in-line code blocks as our pilot annotation study found that most of the solutions reside in standalone code blocks rather than in-line code blocks.

For the standalone code blocks, the general idea behind our annotation protocol is that: 1) all code blocks that forms a standalone solution by itself are labelled as "B"; 2) for code blocks that together form a joined solution, the first code block is labelled as "B", and all subsequent code blocks are labelled as "I"; 3) all code blocks that neither form a solution by themselves nor constitute a joined solution are labelled as "O".

4 Datasets

Our experiments are mainly based on two datasets, which will be referred to as StaQC-human and Lang2Code-human respectively in the following sections. StaQC-human refers to the manually annotated dataset provided by the StaQC study, and Lang2Code-human refers to the dataset annotated by ourselves. We are not using CoNaLa datasets in this study because their annotation is not performed at code block level.

4.1 StaQC-human

StaQC-human contains 2,169 and 2,056 manually annotated question-code (QC) pairs for Python and SQL. Since they focus solely on standalone solutions, they annotate a code block as a solution if and only if the code block can solve the question by itself, which means that the dataset does not contain any solution that is consists of multiple code blocks.

Given that their annotation is at single code block level, their training, validation, and test sets are also generated in a similar manner. Instead of splitting data at post level, they chose to perform the split at single code block level, that is, code blocks from the same post may end up be in different sets. We think this approach can be questionable. On the one hand, it may cause potential data leakage as consecutive code blocks will share the same text block between them. On the other hand, they failed to take advantage of the information resides in the post as a whole. Therefore, we decided to reorganize their data at post level. We kept only the posts that have all their code blocks annotated shuffled the data at post level before splitting it into 80% training, 10%validation, and 10% test sets. The results are summarized in Table 2.

		# of posts	stand-alone solution code blocks		joint solution code blocks		non-solution code blocks	
			#	%	#	%	#	%
Lang2Code	Python-train	1058	1328	46.1	423	14.7	1131	39.2
	Python-valid	132	185	48.3	58	15.1	140	36.6
	Python-test	133	178	48.0	44	11.9	149	40.2
	SQL-test	70	91	49.5	33	17.9	60	32.6
	R-test	80	88	40.7	44	20.4	84	38.9
	Git-test	75	63	29.9	107	50.7	41	19.4
	Java-test	53	47	34.1	68	49.3	23	16.7
	Linux-test	69	87	42.4	63	30.7	55	26.8
StaQC-human	Python-train	1139	1354	45.9	0	0.0	1592	54.1
	Python-valid	142	181	47.9	0	0.0	197	52.1
	Python-test	143	164	44.6	0	0.0	204	55.4
	SQL-train	1056	1632	61.6	0	0.0	1016	38.4
	SQL-valid	132	201	61.8	0	0.0	124	38.2
	SQL-test	133	190	58.5	0	0	135	41.5

Table 2: The statistics of Lang2Code-human and StaQC-human. StaQC-human is reorganized at post level using the original data.

4.2 Lang2Code-human

Lang2Code-human is sampled from Stack Overflow's raw data dump (cite) and manually annotated by three annotators. We focused on Python as the primary programming language, which is later used as the training data and part of the test data. Since we also want to test whether our model will scale to multiple languages without training on every single languages, we also annotated five additional languages to use as test data:

1. Java
2. R
3. SQL
4. Git
5. Linux

In terms of sampling approach, the Stack Overflow's dataset consists more than 2 million posts. We initially chose to use view count as the weight for generating weighted Python samples from the raw data dump to filter out low quality question posts. However, after we finished annotating the weighted sample of size 450, we found that the posts with relatively high view counts tend to have a large proportion of "B"s. In order to obtain a more balanced dataset, we then generated an unweighted Python sample of size 1,500. The samples of size 500 for the other five programming languages were also generated using the unweighted approach to secure a relatively balanced result.

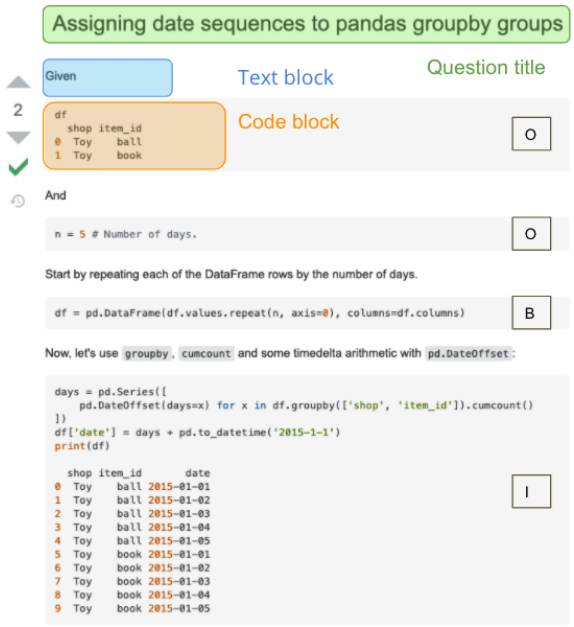

Figure 3: Illustration of our QA task

5 Model

This section provides an overview of the SLQA model and explains the variants of the model.

5.1 Intuition

Assuming that a post has the following structure: $\{Q, T_1, C_1, T_2, C_2, ..., T_N, C_N\}$, where Q represents the question title, T_i represents text blocks in the accepted answer, and C_i represents code blocks in the accepted answer. Our task is to automati-

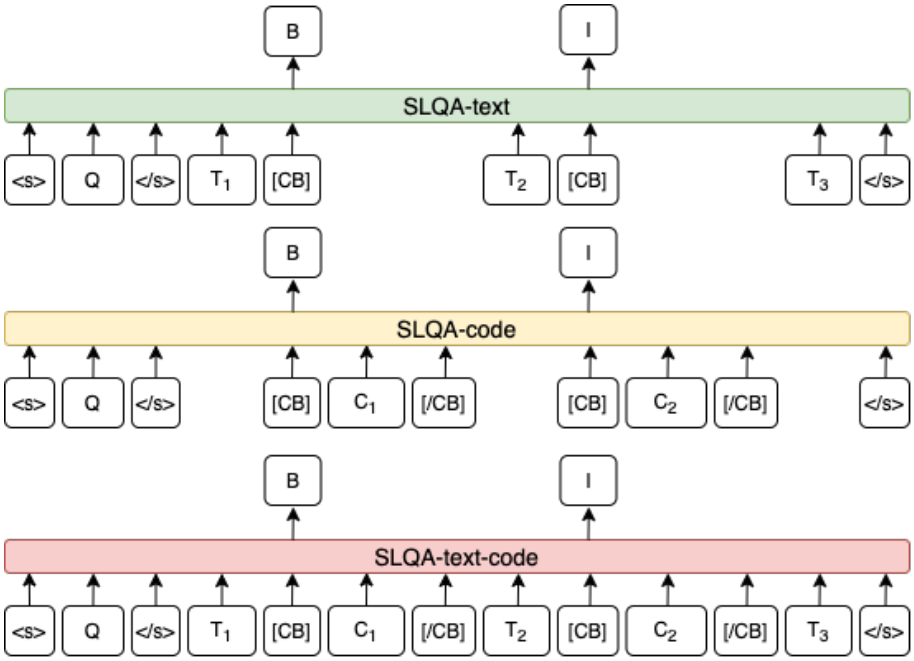

Figure 4: Three variants of SLQA

cally assign an IOB label to each code block. In this study, we worked on what is left blank in the StaQC study and modeled the sequence of code blocks simultaneously to take advantage of the entire text and code information contained in the answer post.

We started off by analyzing the flow of an example answer post as shown in Figure 3. In this post, the first text block T_1 contains a single word "given" and the second text block T_2 contains a single word "and", which indicate that the code blocks C_1 and C_2 will likely contain merely the context of the solution instead of the solution itself. In the third text block T_3, there are keywords "start by", which suggests that C_3 might contain part of the solution. Then we notice that the last text block T_4 contains the keyword "now", thus it is reasonable to infer that the last code block C_4 will possibly contain the second half of the solution. Therefore, even without looking at the actual code, we should be able to guess that "OOBI" are likely to be the correct IOB labels for this specific answer post.

5.2 Variants of SLQA

In this study, we propose three input-level variants of the SLQA model: SLQA-text, SLQA-code, and SLQA-text-code. The intuition of inferring IOB labels solely based on textual information give rise to the SLQA-text model, which takes only text blocks in the answer post as input, and all code blocks are masked with the [CB] token.

6 Experiments

In this section, we discuss our experiments and demonstrate the strength of our proposed SLQA-text-code model.

6.1 Experiment Setup

Data. As mentioned in section 3, the datasets that we will be using for the experiments are StaQC-human and Lang2Code-human. The detailed statistics of the two datasets are presented in Table 2.

Preprocessing. Using the vocabulary and tokenization methods provided by BPE, we preprocessed both text and code in accepted answer as if they are plain text.

Implementation. SLQA is constructed through fine-tuning RoBERTa-large. We applied early stopping with patience equals 5, and we designed the batch size to be 16. Each of the three variants of SLQA are trained and tested five times using different random splits of the datasets, and the final results are obtained by averaging all five results.

Evaluation Metrics. To evaluate and compare different models, we adopted precision, recall, F_1, and Exact Match as our evaluation metrics, which are defined in Nakayama (2018).

6.2 Baselines

6.2.1 Bi-View Hierarchical Neural Network

Bi-View Hierarchical Neural Network (BiV-HNN) is a bidirectional Gated Recurrent Unit (Bi-GRU) based binary classification model proposed in the StaQC study, which takes as input the bidirectional GRU-based RNN encoded vectors, including question title, standalone code block, and text blocks surrounding the code block, and outputs a binary prediction of whether the code block is a solution to the question title.

Since we reorganized the original StaQC-human dataset at post level, we needed to train the BiV-HNN model again with the newly split training, validation, and test sets. For consistency, we adopted the same hyperparameters and retained the original tokenization and code normalization.

We were able to reproduce the BiV-HNN model on both Python and SQL data, and the results are summarized in Table 3. However, since the original work adopted different vocabulary, tokenization, and code normalization methods on Python and SQL separately, it is not possible to perform transfer learning on the two languages using BiV-HNN.

6.2.2 Select-All and Select-First

The other two commonly used baselines are Select-All and Select-First. Select-All refers to the heuristic of treating all code blocks in the accepted answer as standalone solutions, that is, all code blocks are labelled as "B". Select-First means choosing the first code block as the standalone solution and ignore the rest, which is equivalent to labelling the first code block as "B" and the rest as "O".

6.3 Experiments on StaQC-human

Through the experiments on StaQC-human, there are mainly three things we want to understand. Firstly, we want to compare the performance of SLQA and baseline models on StaQC-human-Python and StaQC-human-SQL respectively. We believe that by taking advantage of global information, SLQA and its variants should be able to achieve better results than the existing baselines.

Secondly, we would like to evaluate SLQA's ability to transfer learning across languages. In the context of StaQC-human, this means 1) train and validate the model on Python, and then evaluate on SQL; and 2) train and validate the model on SQL, and then evaluate on Python. Since we did not apply any specific code normalization and used only the vocabulary and tokenization method provided by BPE, it is possible to conduct transfer learning experiments to examine whether SLQA is able to apply the knowledge learned on one programming language to another programming language.

Finally, we want to compare the performance of the three variants of SLQA. For training and evaluation on the same programming language, we think SLQA-text-code has the most advantage because the model will acquire knowledge from both the text and the code information, and such knowledge is directly applicable to the test set. In terms of transfer learning, we think SLQA-code will be the lowest because the syntax, structure, and features of the original programming language and the new programming language tend to be very different, which is a big challenge for the model. The best performer should be SLQA-text as the model only learns textual information, which should not be significantly different from language to language. As for SLQA-text-code, we think it will perform comparably to SLQA-text because it also contains all text information.

6.4 Experiments on Lang2Code-human

Similar to the experiments we conducted on StaQC-human, we designed our experiments on Lang2Code-human to first evaluate the performance of SLQA and the Select-All and Select-First baseline models. Secondly, we performed transfer learning from Python to the other five programming languages to establish a transfer learning benchmark. Finally, we would like to compare the performance of the three variants of SLQA, our hypothesis about their performance stays the same for the same reasons stated in section 5.3.

7 Results

First we see from table 3 that for the experimental results on Python, SLQA's model outperformed BiV-HNN in each mode (Text, Code, Text+Code). This is partly due to the fact that full-text information can also help the model understand the data better. The best performance is SLQA-text-code, with an 8% improvement over BiV-HNN.

Comparing the three variants of SLQA, we find that SLQA-text-code is the best, followed by SLQA-text, and finally SLQA-code. Here we see that SLQA-code performed significantly worse than SLQA-text compared to the BiV-HNN variant where Text-HNN and Code-HNN were not too

	Model	Precision	Recall	F1	Accuracy
	Select First	64.3	56.1	60	66.6
	Select-All	44.6	**100**	61.7	44.6
Python->Python	Text-HNN	71.7	79.5	75.4	76.8
Python->Python	Code-HNN	71.7	77.8	74.6	76.4
Python->Python	BiV-HNN	76.2	83.8	79.6	81
Python->Python	SLQA-text	78.5	86	82	83
Python->Python	SLQA-code	72.8	87	79.1	79.1
Python->Python	SLQA-text-code	**85.3**	90.2	**87.6**	**88.4**
SQL->Python	SLQA-text	74.6	90.1	81.6	81.9
SQL->Python	SLQA-code	84.5	24.3	37.3	63.6
SQL->Python	SLQA-text-code	79.5	82.3	80.9	82.3

Table 3: Experiment results evaluated on StaQC-human Python test set. Python->Python means the model is trained and validated on Python training & validation set, and evaluated on Python test set. SQL->Python means the model is trained and validated on SQL training & validation set, and evaluated on Python test set.

	Model	Precision	Recall	F1	Accuracy
	Select First	73.7	51.3	60.5	60.6
	Select-All	58.8	**100**	74	58.8
SQL->SQL	Text-HNN	76.5	86.6	81.2	76.5
SQL->SQL	Code-HNN	75.3	88.8	81.5	76.3
SQL->SQL	BiV-HNN	83.2	96	89.1	86.1
SQL->SQL	SLQA-text	86.3	94	90	87.7
SQL->SQL	SLQA-code	84.2	89	86.5	83.5
SQL->SQL	SLQA-text-code	85.2	97.7	**91**	**88.4**
Python->SQL	SLQA-text	88	82.8	85.2	83.1
Python->SQL	SLQA-code	77.3	80.1	78.3	74
Python->SQL	SLQA-text-code	**90.4**	88.3	89.2	87.2

Table 4: Experiment results evaluated on StaQC-human SQL test set.

	Model	Precision	Recall	F1	Exact Match
Python	Select First	62.1	41.6	49.8	55.3
Python	Select All	48.5	**91.4**	63.4	52.6
Python->Python	SLQA-text	63.9	76.6	69.7	69.2
Python->Python	SLQA-code	65.1	77.5	70.6	68
Python->Python	SLQA-text-code	**70.2**	82.3	**75.7**	**73.5**
SQL	Select First	50.7	33.3	40.2	46.2
SQL	Select All	50	**87.6**	63.7	57.1
Python->SQL	SLQA-text	65.2	75.8	70	70.2
Python->SQL	SLQA-code	63.1	58.1	59.8	61.2
Python->SQL	SLQA-text-code	**70.6**	82.7	**76.1**	**76.3**
R	Select First	53.2	39.3	45.2	57
R	Select All	40.7	**82.2**	54.5	49.5
Python->R	SLQA-text	56.2	74.5	64	67.3
Python->R	SLQA-code	58	73.6	64.7	65.8
Python->R	SLQA-text-code	**58.4**	77.2	**66.5**	**67.6**
Git	Select First	36.5	25.7	30.2	43.1
Git	Select All	30.8	**61.9**	41.1	49.3
Python->Git	SLQA-text	53.4	57.7	55.4	64.8
Python->Git	SLQA-code	46.8	49.3	48	55.9
Python->Git	SLQA-text-code	**58.9**	58.8	**58.7**	**67.8**
Java	Select First	45.3	32	37.5	49.3
Java	Select All	34.1	62.7	44.1	53.6
Python->Java	SLQA-text	49.3	59.2	53.8	65.2
Python->Java	SLQA-code	50	50.4	50	58
Python->Java	SLQA-text-code	**59.4**	**67**	**62.9**	**71.5**
Linux	Select First	53.6	33.6	41.3	45.9
Linux	Select All	42.9	**80**	55.9	53.2
Python->Linux	SLQA-text	61.3	71	65.8	68.3
Python->Linux	SLQA-code	62.4	60.9	61.5	58.2
Python->Linux	SLQA-text-code	**66.6**	74	**70.1**	**69.7**

Table 5: Experiment results evaluated on Lang2Code-human test set. The test set includes the Python test set and the other 5 coding language test set. The SLQA model is trained on Python training set and evaluated on the 6 test set to test the transfer learning ability.

different. This is probably because the input code of Code-HNN was normalized, so the model could find the pattern more easily (e.g., counting the number of VARs, etc.). In contrast, in SLQA-code the unprocessed raw code was directly input to the model. While this made it difficult the SLQA-code model to extract patterns, the fact that it did not require any domain knowledge is a huge plus. The similar pattern was observed with experiments results on SQL 4

For the transfer learning experiments of Python->SQL and SQL->Python, we find that SLQA-code performed poorly, which means that it is difficult to make the model predict language A in language B without giving the model additional information, which is consistent with our hypothesis. We also notice that the experimental results of Python->SQL were relatively much better than those of SQL->Python, which we believe is related to the complexity of the language, probably the language features of Python are more complex and some of them are well compatible with the language features of SQL.

We also found that both SLQA-text-code and SLQA-text perform well in transfer learning. Both of the transfer learning results for SLQA-text-code were even better than results for BiV-HNN which were trained on the original langauge. This indicates that SLQA-text-code has a strong ability to reason by semantic information of text and code.

Next, we apply the SLQA variant to the Lang2Code-human dataset. We find that SLQA-text and SLQA-text-code perform well on top of all test sets, especially SLQA-text-code, which on average exceeds the F1 score of heuristic baseline by about 15%. The increase of SLQA-text-code over the heuristic baseline on the Python dataset over the heuristic baseline is very close to its increase in other test sets. This indicates that SLQA achieves similar results on test sets in other languages as on the Python test set, which validates the strong migration ability of SLQA-text-code across programming languages.

8 Conclusion

Traditional approaches to constructing NL-PL pairs had to choose between using programming language attributes or natural language attributes. The former required domain knowledge as well as struggled to transfer learning to other languages, while the latter did not exploit the sub-word level syntactic similarity within different programming languages. We showed that SLQA, a model that incorporates both the programming language as well as natural language attributes performs better than the current state of the art. We also showed that this kind of model can be transferred to other languages with no re-training or domain language required.

Acknowledgments

The acknowledgments should go immediately before the references. Do not number the acknowledgments section. **Do not include this section when submitting your paper for review.**

References

Miltiadis Allamanis, Hao Peng, and Charles Sutton. 2016. A convolutional attention network for extreme summarization of source code.

Miltos Allamanis, Daniel Tarlow, Andrew Gordon, and Yi Wei. 2015. Bimodal modelling of source code and natural language. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2123–2132, Lille, France. PMLR.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions.

Colin B. Clement, Dawn Drain, Jonathan Timcheck, Alexey Svyatkovskiy, and Neel Sundaresan. 2020. Pymt5: multi-mode translation of natural language and python code with transformers.

James Curran and Stephen Clark. 2003. Language independent NER using a maximum entropy tagger. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 164–167.

Aditya Desai, Sumit Gulwani, Vineet Hingorani, Nidhi Jain, Amey Karkare, Mark Marron, and Subhajit Roy. 2016. Program synthesis using natural language. In *Proceedings of the 38th International Conference on Software Engineering*, pages 345–356.

Hakan Erdogan. 2010. Sequence labeling: Generative and discriminative approaches. ICMLA.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. Codebert: A pre-trained model for programming and natural languages.

Philip Gage. 1994. A new algorithm for data compression. *C Users J.*, 12(2):23–38.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2020. Codesearchnet challenge: Evaluating the state of semantic code search.

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, Berlin, Germany. Association for Computational Linguistics.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

Nicholas Locascio, Karthik Narasimhan, Eduardo DeLeon, Nate Kushman, and Regina Barzilay. 2016. Neural generation of regular expressions from natural language with minimal domain knowledge.

Sifei Luan, Di Yang, Celeste Barnaby, Koushik Sen, and Satish Chandra. 2019. Aroma: code recommendation via structural code search. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–28.

Hiroki Nakayama. 2018. seqeval: A python framework for sequence labeling evaluation. Software available from https://github.com/chakki-works/seqeval.

Chris Quirk, Raymond Mooney, and Michel Galley. 2015. Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 878–888, Beijing, China. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text.

Lance Ramshaw and Mitch Marcus. 1995. Text chunking using transformation-based learning. In *Third Workshop on Very Large Corpora*.

Saksham Sachdev, Hongyu Li, Sifei Luan, Seohyun Kim, Koushik Sen, and Satish Chandra. 2018. Retrieval on source code: a neural code search. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 31–41.

Yi Wei and Nirupama Ch. 2015. Building bing developer assistant.

Shuhan Yan, Hang Yu, Yuting Chen, Beijun Shen, and Lingxiao Jiang. 2020. Are the code snippets what we are searching for? a benchmark and an empirical study on code search with natural-language queries. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 344–354.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering.

Ziyu Yao, Daniel S. Weld, Wei-Peng Chen, and Huan Sun. 2018. Staqc. *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*.

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to mine aligned code and natural language pairs from stack overflow.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada. Association for Computational Linguistics.

Meital Zilberstein and Eran Yahav. 2016. Leveraging a corpus of natural language descriptions for program similarity. In *Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pages 197–211.