Descriptions to Code: Generating and retrieving concise code snippets

Witheld for review

ABSTRACT

Tools capable of code generation have the capacity to augment programmers capabilities drastically. However, the field of software engineering is currently left with primitive tools with limited search and completion functionality. In this paper we compare current methods from both code retrieval and machine-learned code generation models for the task of code generation, and propose a new approach combining these two techniques. We use existing standard benchmark collections, including DJango and CoNaLa. In addition, we develop new human generated datasets with a focus on short, realistic snippets of code. We also study methods for generating large-scale synthetic code collections needed to train neural translation models. We experiment with domain transfer methods across the various code collections. We find that current retrieval based algorithms are currently significantly more effective than neural translation-based approaches. We also analyze our findings and demonstrate the limitations of current models, suggesting new areas for future improvements.

CCS CONCEPTS

• **Information systems → Information retrieval**;

KEYWORDS

Code generation, Code Retrieval, Neural Machine Translation

ACM Reference Format:

Witheld for review. 2019. Descriptions to Code: Generating and retrieving concise code snippets. In *Proceedings of SIGIR (SIGIR'19).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Programmers require parallel knowledge of many different programming languages, libraries and techniques to effectively write code. The sheer amount of structured information required is often too much to keep in one's head and means online searches for library examples or clarifications on syntax are frequent. This process lengthens the time from idea to product and can also make the programmer lose focus as they veer off on tangents to search for forgotten syntax.

Code generation sets out to solve this by allowing the programmer to express their ideas in a natural English statement and have the code be generated via an algorithm. In doing so the programmer can focus on higher level tasks. We tackle the problem of mapping programmer's intent in English to a snippet of Python code delivering the desired intent. Current code generation approaches show promising results with models consistently improving on standard metrics such as BLEU and exact match accuracy. Despite this, it is currently unclear what improvements mean to the intended users of the systems.

We aim to gain deeper insight into what it means to generate relevant code using proven models in the field such as Retrieval and Neural Machine Translation (NMT). To do this we source a new test set made of 55 questions from StackOverflow annotated by multiple programmers, and use it to gain new insight into the results from in-domain BLEU scores. We evaluate our results across multiple standard datasets (DJANGO, Docstring, CoNaLa) and complement these by building our own synthetic one specifically targeted at the problem. In light of our findings we create a new model combining the two previous approaches by enriching description with the top retrieved result from our retrieval model. We note an improvement to BLEU scores across in-domain and our real world StackOverflow questions. To scrutinize further we create a study involving multiple programmers tasked to rate the relevance of a generated or retrieved code snippet given a description. This comparison of all studied models across all datasets along with human control subjects suggest that well established retrieval methods deliver desirable real world qualities to programmers that BLEU scores fail to reflect. Qualitative feedback from the study tells us that the well formed nature of retrieval results are preferable to outputs by Neural Translation models.

2 RELATED WORK

Retrieval models are well established in the field of Code Improvement. Many attempts emphasize helping programmers debug programs, and remove duplicate or similar code by identifying close matches from different sections in source code. Early approaches [3] rely on highly structured formal methods to convert queries such as "find all functions that contain a variable arr" into a specialized query language to search for exact matches. Mishne et al. [4] propose code snippet retrieval by forming unstructured queries over code and use a "fuzzy" approach to help programmers find similar snippets to their query. These approaches attempt to search over the target code to find relevant results. Sindhgatta [7] employs a different approach by querying over code authors' annotations to retrieve relevant snippets. This last approach is most similar to our retrieval model.

Most recent work treats Code Generation as a Machine Translation task and applies translation models, such as encoder-decoder networks. First attempts involved input and output as linear sequences [8]. Words are tokenized and passed to an embedding layer, and are then encoded into a fixed size latent space with one or more memory modules. Tokens in the target language are then generated from this latent space. This allows for variable length sequences to

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SIGIR'19, July, Paris, France
© 2019 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

be taken in and generated. Following this, others suggest a way of generating code to take advantage of the programming language's inherent structure, representing the output as a series of construction decisions. Rabinovich et al. [6] take it a step further by using Abstract Syntax Trees (AST) in the decoding process. Their network generates a tree structure in a top-down manner by adding atomic operations which build the tree keeping information from parent and sibling nodes. This technique is well suited for the task since programming languages follow a tree structure with intermediate nodes holding functions, and leaf nodes holding primitives like Strings and Ints. However, the work mentioned above does not include any context such as already existing code or variables when feeding the source. When writing code that integrates with a larger program, it usually depends on variables or methods in said source. Iyer et al. [2] attempt to solve this by appending already existing code into the input vector, and changing distinct variable names to universal positional tokens combined with AST decoders. This yields good results as existing variables and methods are taken into account.

3 GENERATION AND RETRIEVAL METHODS

3.1 Task Definition

We define the task of Code Generation from Natural Language as follows. Given a topic description, T, the goal is to generate a single most relevant snippet, S, of code that satisfies the topic description.

To perform this task we common formulate it as follows: **Input:** NL tokens from T are split into a sequence $\mathbf{t}_i$. $_i$ denoting the position of the token in the sequence.
Output: Code tokens split into a sequence $\mathbf{s}_i$

We note that the result can come from either retrieval (existing code) or be generated by a generative model, such as the translation models we experiment with. The output are short snippets – equivalent to a small line (or lines) of code. This is roughly analogous to generating or retrieval code at the level of sentence or paragraph.

3.2 Retrieval

For retrieval we use Cosine Similarity (1) with a Bag-of-Words (BoW) representation, text is weighted using TF-IDF weighting. We perform passage retrieval on the query language by fitting the tokenized query text into the vocabulary space and ranking descriptions based on their score.

$$\cos(\mathbf{t}, \mathbf{s}) = \frac{\mathbf{ts}}{\|\mathbf{t}\|\|\mathbf{s}\|} = \frac{\sum_{i=1}^{n} \mathbf{t}_i \mathbf{s}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{t}_i)^2}\sqrt{\sum_{i=1}^{n} (\mathbf{s}_i)^2}} \quad (1)$$

3.3 Neural Machine Translation

We evaluate a Seq2Seq baseline. Tokens embeddings are learned through training and passed sequentially to the encoder. We cap the number of input and output tokens at 50. The encoder and decoder are n-layer LSTM's paired withk an attention mechanism to produce a conditional distribution over the next source code token given all the previous tokens. We use Softtmax_Cross_Entropy loss as our output sequence cost as we look for an exact match to the label. Our model settings are: batch size 256, rnn size 128, num layers 2, dropout 0.2, steps 24k, scaled luong attention.

3.4 Retrieval + NMT

We combine the exact model configurations as above into this hybrid approach. This model takes the top result of the Retrieval model and concatenates the code tokens with the original description and a separator token between: "<RETRIEVAL_ENDS>". The new sequence behaves exactly as in the previous model and has embeddings learned at training time. Training is done sequentially. First we fit the retrieval model, this is a fast operation. Then we use it to enrich the description. Since we train both models on the same set, we take the second best result from the Retrieval model as the first would be an exact match to the output code.

4 EXPERIMENTAL SETUP

In this section we describe the collections of code used, the code description topics, and the collection of results.

4.1 Code collections

4.1.1 Django. The Django dataset [5] was produced by a single engineer tasked to annotate the entire DJANGO source code line by line (18k+ lines). The original aim for the dataset was to map from code to pseudo-code. This leads to relatively detailed descriptions of each line which map simply to code. Barone and Sennrich [1] argue that the fact that a single source and annotator result in narrow descriptions that are not representative of real code.

4.1.2 Docstring. This is a large and varied collection of Python "docstrings" (documentation string) [1]. Docstrings are designed to allow programmers to document the operation and interface of codes object they build. The dataset consists of over 100k pairs of docstrings with the associated function code scraped from GitHub. The code originates from the top-level methods of projects, this reduces inter-dependencies often present in large code repositories, and presents self-contained methods which are of optimal use when performing machine translation.

4.1.3 CoNaLa. CoNaLa [9] is sourced from StackOverflow questions and answers. The dataset consists of over 2k hand-written short answers to programming questions. These are high quality code/description pairs. However, the limited dataset size poses problems for data hungry methods such as Neural Translation. The authors provide an additional automatically annotated set of 600k+ pairs. During evaluation of the later dataset we decide to use only the 2k hand written pairs.

4.1.4 Synthetic. Our approach when creating this dataset was to create a generic set of code/description pairs which mapped similar intentions to a single code snippet. We search highly rated projects from GitHub and StackOverflow Posts to gather 400 questions and relevant snippets to annotate. We then augment the data X100 by combining hand made substitutions and word synonyms. We also combine simple single self-contained statements into multi-line snippets by adding "and" between two simple descriptions. We obtain a total of 53k pairs. This will be made available.

4.1.5 Human. To validate our results, The programmers were allowed to use a search engine or not and return single or multi-line statements.

4.2 StackOverflow Code Description topics

We create a new collection of evaluation code description topics. The topics were gathered by manually selecting 55 descriptions from the popular programming website StackOverflow. We select topics relating to standard python functions such as list, dictionary, string operations, library imports and simple function calls. The topics and the manual assessments will be made publicly available (on Github) for use as a new benchmark collection for code generation.

4.2.1 Manual Relevance Assessments. To assess the real-world effectiveness, we collect manual relevance judgments. Each annotator, an undergraduate CS student, assessed the relevance for the 55 code descriptions. We create the set of judgments by pooling the results from each of the methods described earlier. In particular, we fully evaluate the top 1 result from each system. The result is a collection of 770 code/description relevance judgments. In additional, we also kept assessor qualitative comments on the observed code.

The participants are tasked to rate the relevance of the generated snipped given the description on a scale from 1 to 5 where 1 is "unusable" and 5 is "exact translation". These ratings are then binarized at a relevance threshold of 3 to create binary judgments. We note their rating agreement with Krippendorff's alpha using the interval metric of 0.712.

An example code description with sample top 1 results for each assessed method is shown below.

Description:

```
execute a command 'command' in
the terminal from a python script
```

Human 1:

```
import subprocess
subprocess.Popen(command)
```

Human 2:

```
os.system(command)
```

Retrieval CoNaLa:

```
os.system('dir c:\\')
```

NMT CoNaLa:

```
os.delete('some')
```

Retrieval+NMT CoNaLa:

```
os.system(<unk> <unk>)
```

4.3 Pre-processing

We pre-process data in the same way for each model. The English descriptions are tokenized by spaces and the associated code is separated using Python3's built-in code tokenizer, breaking up the code into the more atomic units. Tokenizing code by spaces results in excessive complex and single use tokens of little use for generation.
All vocabulary is used for Retrieval, but is capped at 20k for NMT with most frequent tokens kept, this is a memory limitation. All tokens out of vocabulary are set to "<UNK>".

4.4 Metrics

Dev BLEU: We evaluate the BLEU score between target sequences for a subset of the training data. Unseen to the model but still in-domain.
Stack BLEU: We perform the same BLEU calculation as above over the labels for 55 code descriptions from StackOverflow. These labels are hand written by a human annotator.
Precision@1: We use precision at the top ranked result. This allows us to equally compare retrieval and generation methods.
For all of the results we perform statistical significance testing using a paired t-test and report significance at a 95 percent confidence interval.

5 RESULTS

In this section we examine the results of our experiments on the various collections and methods. We start with baseline retrieval and neural translation models trained on various corpora and evaluated using BLEU. We then focus on results of our new proposed Retrieval+NMT model. Finally, we use the manual assessments as a way to measure the real-world effectiveness of the methods and we contrast this with BLEU-based evaluation.

5.1 Baseline Retrieval and NMT results

The section below presents results of the retrieval and NMT models trained on a variety of corpora. We also compare to a human coder baseline.

Model	Dev BLEU	Stack BLEU
Retrieval Django	45.9	4.3
Retrieval Docstring	21.8	1.2
Retrieval CoNaLa	11.8	5.1
Retrieval Synthetic	75.6	9.7
NMT Django	51.2	3.4
NMT Docstring	1.7	0.3
NMT CoNaLa	16.3	2.3
NMT Synthetic	92.3	2.2
Human	/	36.5

Fig2: In-domain (Dev) and Test (StackOverflow) scores on multiple datasets.

In-domain sets are common in the field of code generation to benchmark the effectiveness of models. We find Neural Translation outperforms Retrieval for most datasets for in-domain sets. However, our StackOverflow set is much more difficult shown by the vastly lower scores, and yields the opposite results as the retrieval method finds more relevant code than NMT by a significant margin. Humans show vastly improved results, though the fact that they don't score near 100 BLEU points suggests BLEU is limited. Results like these show the difficulty for a model to perform well in real world situations.

5.2 Retrieval + NMT model results

In this section we focus on results of our new proposed retrieval + NMT model results.

Dev BLEU	NMT	Retrieval+NMT	Delta
Django	51.2	53.5	+4.5%
Docstring	1.7	1.9	+11.8%
CoNaLa	16.3	13.0	-20.2%
Synthetic	92.3	98.0	+6.1%
Stack BLEU			
Django	3.4	4.6	+34.3%
Docstring	0.3	1.7	+466.6%
CoNaLa	2.3	3.8	+65.2%
Synthetic	2.2	5.5	+150.0%

Fig3: BLEU Score comparison of NMT and Retrieval+NMT algorithms on Dev and Stack sets

Adding retrieval capabilities to Neural Translation yields improvements in almost every dataset. Most notable improvements are found in the Stack BLEU scores pushing marginally past our previous benchmarks. We theorize passing the actual description with the retrieval model's best guess allows the hybrid translation model to estimate appropriate syntax and target length while changing relevant tokens to match the description. We note a strong dependency on datasets for scores with the new model finding similar difficulty generalizing on the Docstring dataset opposed to more focused datasets like Django, Conala and our Synthetic one.

5.3 Manual assessment

We now compare Stack BLEU results to human rated effectiveness.

Model	Stack BLEU	Prec@1
Retrieval Django	4.3	20.9%
Retrieval Docstring	1.2	18.2%
Retrieval CoNaLa	5.1	20.9%
Retrieval Synthetic	9.7	15.5%
NMT Django	3.4	6.4%
NMT Docstring	0.3	4.6%
NMT CoNaLa	2.3	3.6%
NMT Synthetic	2.2	4.6%
Retrieval+NMT Django	4.6	0.9%
Retrieval+NMT Docstring	1.7	3.6%
Retrieval+NMT CoNaLa	3.8	4.5%
Retrieval+NMT Synthetic	5.5	7.3%
Human	36.5	90.0%

Fig4: Stack BLEU & Precision@1 for all models on their StackOverflow descriptions

As participants rate the effectiveness of the generated code for all 55 descriptions, we observe statistically significant improvements (>95% confidence) for retrieval over other methods. Taking a closer look at the outputs we notice the translation models despite picking up key words in the description, they often void the output by inserting irrelevant tokens that make the outputs unsuitable to users. Human coders manage very high relevance despite differences in the tokens used (highlighted by lower BLEU score). Since BLEU gives a score based on n-gram similarity, these findings reinforce the view that other metrics akin to the polymorphic nature of code will highlight better success in a model's predictions. Participants reported having difficulty distinguishing between similarly bad predictions, suggesting user ratings show most telling results when the generated code is near high quality. Below is an example of a result with a high BLEU score, but is marked as non-relevant. It shows that BLEU does not capture the redundancy well. And that translation models pick up on important details like "numpy", while retrieval models may miss key aspects, but return well-formed code.

Description:
```
import numpy
```
Retrieval:
```
import code
```
Retrieval+NMT:
```
return numpy . array ( numpy . array
( numpy . array ( numpy . array (
```

6 CONCLUSION

Code Generation proves a difficult task, even for current Neural Translation architectures. We find that generalization to out-of-domain sets in real-world scenarios is often poor. We experimented with enriching current translation methods with retrieved code and show that this yields improvements on BLEU. Further, we find that commonly used in-domain BLEU don't show the full picture, with real world descriptions from StackOverflow showing significant declines in effectiveness. Experiments show that users find retrieval-based approaches more relevant because they return well-formed code. This suggests that future work in code generation should give greater emphasis to this aspect in the model and evaluation.

REFERENCES

[1] Antonio Valerio Miceli Barone and Rico Sennrich. 2017. A parallel corpus of Python functions and documentation strings for automated code documentation and code generation. *arXiv:1707.02275 [cs]* (July 2017). http://arxiv.org/abs/1707.02275 arXiv: 1707.02275.

[2] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2018. Mapping Language to Code in Programmatic Context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, 1643–1652. http://www.aclweb.org/anthology/D18-1192

[3] Jun Jang Jeng and Betty H. C. Cheng. 1993. Using formal methods to construct a software component library. In *Software Engineering âĂŤ ESEC '93*, Gerhard Goos, Juris Hartmanis, Ian Sommerville, and Manfred Paul (Eds.). Vol. 717. Springer Berlin Heidelberg, Berlin, Heidelberg, 397–417. https://doi.org/10.1007/3-540-57209-0_27

[4] Gilad Mishne and Maarten De Rijke. 2004. Source Code Retrieval using Conceptual Similarity. In *Proc. 2004 Conf. Computer Assisted Information Retrieval (RIAO âĂŽ04*. 539–554.

[5] Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to Generate Pseudo-Code from Source Code Using Statistical Machine Translation (T). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, Lincoln, NE, USA, 574–584. https://doi.org/10.1109/ASE.2015.36

[6] Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract Syntax Networks for Code Generation and Semantic Parsing. *arXiv:1704.07535 [cs, stat]* (April 2017). http://arxiv.org/abs/1704.07535 arXiv: 1704.07535.

[7] Renuka Sindhgatta. 2006. Using an information retrieval system to retrieve source code samples. In *Proceeding of the 28th international conference on Software engineering - ICSE '06*. ACM Press, Shanghai, China, 905. https://doi.org/10.1145/1134285.1134448

[8] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 3104–3112. http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf

[9] Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to Mine Aligned Code and Natural Language Pairs from Stack Overflow. *arXiv:1805.08949 [cs]* (May 2018). http://arxiv.org/abs/1805.08949 arXiv: 1805.08949.