CommitBERT: Commit Message Generation Using Pre-Trained Programming Language Model

Tae-Hwan Jung
Kyung Hee University
nlkey2022@gmail.com

Abstract

In version control using Git, the commit message is a document that summarizes source code changes in natural language. A good commit message clearly shows the source code changes, so this enhances collaboration between developers. To write a good commit message, the message should briefly summarize the source code changes, which takes a lot of time and effort. Therefore, a lot of research has been studied to automatically generate a commit message when a code modification is given. However, in most of the studies so far, there was no curated dataset for code modifications (additions and deletions) and corresponding commit messages in various programming languages. The model also had difficulty learning the contextual representation between code modification and natural language.

To solve these problems, we propose the following two methods: (1) We collect code modification and corresponding commit messages in Github for six languages (Python, PHP, Go, Java, JavaScript, and Ruby) and release a well-organized 345K pair dataset. (2) In order to resolve the large gap in contextual representation between programming language (PL) and natural language (NL), we use CodeBERT (Feng et al., 2020), a pre-trained language model (PLM) for programming code, as an initial model. Using two methods leads to successful results in the commit message generation task. Also, this is the first research attempt in fine-tuning commit generation using various programming languages and code PLM. Training code, dataset, and pretrained weights are available at https://github.com/graykode/commit-autosuggestions.

1 Introduction

Commit message is the smallest unit that summarizes source code changes in natural language. Figure 1 shows the git diff representing code modification and the corresponding commit message. A good commit message allows developers to visualize the commit history at a glance, so many teams try to do high quality commits by creating rules for commit messages. For example, Conventional Commits [1] is one of the commit rules to use a verb of a specified type for the first word like 'Add' or 'Fix' and limit the length of the character. It is very tricky to follow all these rules and write a good quality commit message, so many developers ignore it due to lack of time and motivation. So it would be very efficient if the commit message is automatically written when a code modification is given.

```
    output_mode=None,
):
    if max_length is None:
-       max_length = tokenizer.max_len
+       max_length = tokenizer.model_max_length

    if task is not None:
        processor = glue_processors[task]()
```

Message : fix deprecated ref to tokenizer.max_len

Figure 1: The figure above shows an example of commit message and git diff in Github. In the Git process, git diff uses unified format (unidiff [2]): A line marked in red or green means a modified line, and green highlights in '+' lines are the added code, whereas red highlights in '-' lines are the deleted code.

Similar to text summarization, many studies have been conducted by taking code modification $X = (x_1, ..., x_n)$ as encoder input and commit message $Y = (y_1, ..., y_m)$ as decoder input based on the NMT (Neural machine translation) model. (Jiang et al., 2017; Loyola et al., 2017; van Hal et al., 2019) However, taking the code modification without distinguishing between the added and

[1] https://conventionalcommits.org
[2] https://en.wikipedia.org/wiki/Diff

the deleted part as model input makes it difficult to understand the context of modification in the NMT model. In addition, previous studies tend to train from scratch when training a model, but this method does not show good performance because it creates a large gap in the contextual representation between programming language (PL) and natural language (NL). To overcome the problems in previous studies and train a better commit message generation model, our approach follows two stages:

(1) Collecting and processing data with the pair of the added and deleted parts of the code $X = ((add_1, del_1), ..., (add_n, del_n))$. To take this pair dataset into the Transformer-based NMT model (Vaswani et al., 2017), we use the BERT (Devlin et al., 2018) fine-tuning method about two sentence-pair consist of added and deleted parts. This shows a better BLEU-4 score (Papineni et al., 2002) than previous works using raw git diff. Similar to CodeSearchNet (Husain et al., 2019), our data is also collected for six languages (Python, PHP, Go, Java, JavaScript, and Ruby) from Github to show good performance in various languages. We finally released 345K code modification and commit message pair data.

(2) To solve a large gap about contextual representation between programming language (PL) and natural language (NL), we use CodeBERT (Feng et al., 2020), a language model well-trained in the code domain as the initial weight. Using CodeBERT as the initial weight shows that the BLEU-4 score for commit message generation is better than when using random initialization and RoBERTa (Liu et al., 2019). Additionally, when we pre-train the Code-to-NL task to document the source code in CodeSearchNet and use the initial weight of commit generation, the contextual representation between PL and NL is further reduced.

2 Related Work

Commit message generation has been studied in various ways. Jiang and McMillan (2017) collect 2M commits from the Mauczka et al. (2015) and top 1K Java projects in Github. Among the commit messages, only those that keep the format of "Verb + Object" are filtered, grouped into verb types with similar characteristics, and then the classification model is trained with the naive Bayes classifier.

Jiang et al. (2017) use the commit data collected by Jiang and McMillan (2017) to generate the commit message using an attention-based RNN encoder-decoder NMT model. They filter again in a "verb/direct-object pattern" from 2M data and finally used the 26K commit message data. Loyola et al. (2017) uses an NMT model similar to Jiang et al. (2017), but uses git diff and commit pairs collected from 1∼3 repositories of Python, Java, JavaScript, and C++ as training data. Liu et al. (2018) propose a retrieval model using Jiang et al. (2017)'s 26K commit as training data. Code modification is represented by bags of words vector, and the message with the highest cosine similarity is retrieved. Xu et al. (2019) collect only '.java' file format from Jiang and McMillan (2017) and use 509K dataset as training data for NMT. Also, to mitigate the problem of Out-of-Vocabulary (OOV) of code domain input, they use generation distribution or copying distribution similar to pointer-generator networks (See et al., 2017). van Hal et al. (2019) also argues that the Jiang and McMillan (2017) entire data is noise and proposes a pre-processing method that filters the better commit messages.

Liu et al. (2020) argue that it is challenging to represent the information required for source code input in the NMT model with a fixed-length. In order to alleviate this, it is suggested that only the added and deleted parts of the code modification be abbreviated as abstract syntax tree (AST) and applied to the Bi-LSTM model.

Nieb et al. presented a large gap between the contextual representation between the source code and the natural language when generating commit messages. Previous studies have used RNN or LSTM model, they use the transformer model, and similarly to other studies, they use Liu et al. (2018) as the training data. To reduce this gap, they try to reduce the two-loss that predict the next code line (Explicit Code Changes) and the randomly masked word in the binary file.

3 Background

3.1 Git Process

Git is a version management system that manages version history and helps collaboration efficiently. Git tracks all files in the project in the Working directory, Staging area, and Repository. The working directory shows the files in their current state. After modifying the file, developers move the files to the staging area using the `add` command to record the modified contents and write a commit message through the `commit` command. Therefore,

the commit message may contain two or more file changes.

3.2 Text Summarization based on Encoder-Decoder Model

With the advent of sequence to sequence learning (Seq2Seq) (Sutskever et al., 2014), various tasks between the source and the target domain are being solved. Text summarization is one of these tasks, showing good performance through the Seq2Seq model with a more advanced encoder and decoder. The encoder and decoder models are trained by maximizing the conditional log-likelihood below based on source input $X = (x_1, ..., x_n)$ and target input $Y = (y_1, ..., y_m)$.

$$p(Y|X;\theta) = log \sum_{t=0}^{T} p(y_t|y_{<t}, X;\theta)$$

where T is the length of the target input, y_0 is the start token, y_T is the end token and θ is the parameter of the model.

In the Transformer (Vaswani et al., 2017) model, the source input is vectorized into a hidden state through self-attention as the number of encoder layers. After that, the target input also learns the generation distribution through self-attention and attention to the hidden state of the encoder. It shows better summarization results than the existing RNN-based model (Nallapati et al., 2016).

To improve performance, most machine translations use beam search. It keeps the search area by K most likely tokens at each step and searches the next step to generate better text. Generation stops when the predicted y_t is an end token or reaches the maximum target length.

3.3 CodeSearchNet

CodeSearchNet (Husain et al., 2019) is a dataset to search code function snippets in natural language. It is a paired dataset of code function snippets for six programming languages (Python, PHP, Go, Java, JavaScript and Ruby) and a docstring summarizing these functions in natural language. A total of 6M pair datasets is collected from projects with a re-distribution license. Using the CodeSearchNet corpus, retrieval of the code corresponding to the query composed of natural language can be resolved. Also, it is possible to resolve the problem of documenting the code by summarizing it in natural language (Code-to-NL).

3.4 CodeBERT

Recent NLP studies have shown state-of-the-art in various tasks through transfer learning consisting of pre-training and fine-tuning (Peters et al., 2018). In particular, BERT (Devlin et al., 2018) is a pre-trained language model by predicting masked words from randomly masked sequence input and uses only encoder based on Transformer (Vaswani et al., 2017). It shows good perfomances in various datasets and is now extending out of the natural language domain to the voice, video, and code domains.

CodeBERT is a pre-trained language model in the code domain to learn the relationship between programming language (PL) and natural language (NL). In order to learn the representation between different domains, they refer to the learning method of ELECTRA (Clark et al., 2020) which is consists of Generator-Discriminator. NL and Code Generator predict words from code tokens and comment tokens masked at a specific rate. Finally, NL-Code Discriminator is CodeBERT after trained through binary classification that predicts whether it is replaced or original.

CodeBERT shows good results for all tasks in the code domain. Specially, it shows a higher score than other pre-trained models in the code to natural language(Code-to-NL) and code retrieval task from NL using CodeSearchNet Corpus. In addition, CodeBERT uses the Byte Pair Encoding (BPE) tokenizer (Sennrich et al., 2015) used in RoBERTa, and does not generate unk tokens in code domain input.

4 Dataset

We collect a 345K code modification dataset and commit message pairs from 52K repositories of six programming languages (Python, PHP, Go, Java, JavaScript, and Ruby) on Github. When using raw git diff as model input, it is difficult to distinguish between added and deleted parts, so unlike Jiang and McMillan (2017), our dataset focuses only on the added and deleted lines in git diff. The detailed data collection and pre-processing method are shown as a pseudo-code in Algorithm 1:

To collect only the code that is a re-distributable license, we have listed the Github repository name in the CodeSearchNet dataset. After that, all the repositories are cloned through multi-threading. Detailed descriptions of functions that collect the commit hashes in a repository and the code modifi-

Algorithm 1 Code modification parser from the list of repositories.

procedure REPOPARSER($Repos$)
  for $Repo$ in $Repos$ **do**
    $commits = get\_commits(Repo)$
    for $commit$ in $commits$ **do**
      $mods = get\_modifications(commit)$
      for mod in $mods$ **do**
        if $filtering(mod, commit)$ **then**
          break
        end if
        Save $(mod.add, mod.del)$ to dataset.
      end for
    end for
  end for
end procedure

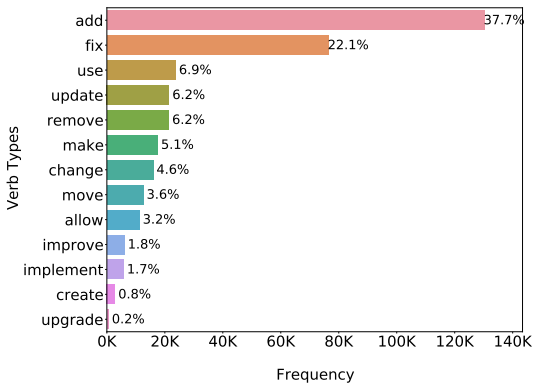


Figure 2: Commit message verb type and frequency statistics. Only 'upgrade' is not included in the high frequency, but is included in a similar way to 'update'. This refers to the verb group in Jiang and McMillan (2017).

cations in a commit hash are as follows:

- `get_commits` is a function that gets the commit history from the repository. At this time, the commits of the master branch are filtered, excluding merge commits. Commits with code modifications corresponding to 6 the program language(.py, .php, .js, .java, .go, .ruby) extensions are collected. To implement this, we use the open-source pydriller (Spadini et al., 2018).
- `get_modifications` is a function that gets the line modified in the commit. Through this function, it is possible to collect only the added or deleted parts, not all git diffs.

While collecting the pair dataset, we find that the relationship between some code modifications and the corresponding commit message is obscure and very abstract. Also, we check that some code modification or commit message is a meaningless dummy file. To filter these, we create the `filtering` function and the rules as follows.

1. To collect commit messages with various format distributions, we limit the collection of up to 50 commits in one repository.
2. We filter commits whose number of files changed is one or two per commit message.
3. Commit message with issue number is removed because detailed information is abbreviated.
4. Similar to Jiang and McMillan (2017), the non-English commit messages are removed.
5. Since some commit messages are very long, the first line is fetched.
6. If the token of code through tree-sitter[3], a parser generator tool, exceeds 32 characters, it is excluded. This removes unnecessary things like changes to binary files in code diff.
7. By referring to the Jiang and McMillan (2017) and Conventional Commits(§ 1) rules, the commit message that begins with a verb is collected. We use spaCy[4] for Pos tagging.
8. We filter commit messages with 13 verb types, which are the most frequent. Figure 2 shows the collected verb types and their ratio for the entire dataset.

As a result, we collect 345K code modification and commit message pair datasets from 52K Github repositories and split commit data into 80-10-10 train/validation/test sets. This results are shown in Table 1.

	Number of Pair Dataset			
	Train	Validation	Test	Number of Repositories
Python	81517	10318	10258	12361
PHP	64458	8079	8100	16143
JavaScript	50561	6296	6252	11294
Ruby	29842	3772	3680	4581
Java	28069	3549	3552	4123
Go	21945	2699	2812	3960
Total : 345759				52462

Table 1: Dataset Statistics for each language collected from 52K repositories of six programming languages.

[3] https://tree-sitter.github.io/tree-sitter
[4] https://spacy.io

5 CommitBERT

We propose the idea of generating a commit message through the CodeBERT model with the our dataset (§ 4). To this end, this section describes how to feed inputs code modification ($X = ((add_1, del_1), ..., (add_n, del_n))$) and commit message ($Y = (msg1, ..., msg_n)$) to CodeBERT and how to use pre-trained weights more efficiently to reduce the gap in contextual representation between programming language (PL) and natural language (NL).

5.1 CodeBERT for Commit Message Generation

We feed the code modification to the encoder and a commit message to the decoder input by following the NMT model. Especially for code modification in the encoder, similar inputs are concatenated, and different types of inputs are separated by a sentence separator (sep). Applying this to our CommitBERT in the same way, added tokens ($Add = (add_1, ..., add_n)$) and deleted tokens ($Del = (del_1, ..., del_n)$) of similar types are connected to each other, and sentence separators are inserted between them. Therefore, the conditional-likelihood is as follows:

$$p(M|C;\theta) = log \sum_{t=0}^{T} p(m_t|m_{<t}, C;\theta),$$
$$m_{<t} = (m_0, m_1, ..., m_{t-1})$$
$$C = concat([cls], Add, [sep], Del, [sep])$$

where M is commit message tokens, C is code modification tokens and $concat$ is list concatenation function. $[cls]$ and $[sep]$ are speical tokens, which are a start token and a sentence separator token respectively. Other notions are the same as Section 3.2.

Unlike previous works, all code modifications in git diff are not used as input and only changed lines in code modification are used. Since this removes unnecessary inputs, it shows a significant performance improvement in summarizing code modifications in natural language. Figure 3 shows how the code modification is actually taken as model input.

5.2 Initialize Pretrained Weights

To reduce the gap difference between two domains(PL, NL), We use the pretrained CodeBERT as the initial weight. Furthermore, we determine that removing deleted tokens from our dataset (§ 4) is similar to the Code-to-NL task in CodeSearchNet (Section 3.3). Using this feature, we use the initial weight after training the Code-to-NL task with CodeBERT as the initial weight. This method of training shows better results than only using CodeBERT weight in commit message generation.

6 Experiment

To verify the proposal in Section 5 in the commit message generation task, we do two experiments. (1) Compare the commit message generation results of using all code modifications as inputs and using only the added or deleted lines as inputs. (2) Ablation study several initial model weights to find the weight with the smallest gap in contextual representation between PL and NL.

6.1 Experiment Setup

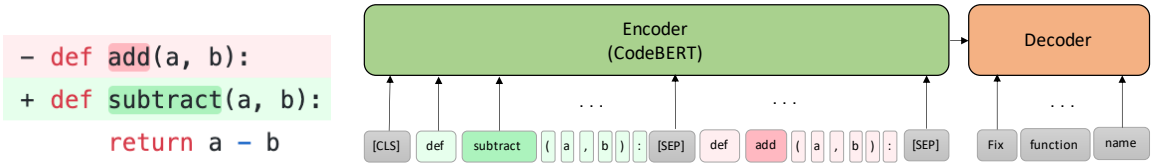
Our implementation uses CodeXGLUE's code-text pipeline library [5]. We use the same model architecture and experimental parameters for the two experiments below. As a model architecture, the encoder and decoder use 12 and 3 Transformer layers. We use 5e-5 as the learning rate and train on one V100 GPU with a 32 batch size. We also use 256 as the maximum source input length and 128 as the target input length, 10 training epochs, and 10 as the beam size k.

6.2 Compare Model Input Type

To experiment generating a commit message according to the input type, only 4135 data is collected from data with code modification in the '.java' files among 26K training data of Loyola et al. (2017). Then we transform these 4135 data into two types, respectively, and experiment with training data for RoBERTa and CodeBERT weights: (a) entire code modification in git diff and (b) only changed lines in code modification. Figure 3 shows these two differences in detail.

Table 3 shows the BLEU-4 values when inference with the test set after training about these two types. Both initial weights show worse results than (b), even though type (a) takes a more extended input to the model. This shows that lines other than changed lines as input data disturb training when generating the commit message.

[5] https://github.com/microsoft/CodeXGLUE



(a) Code modification in git diff

(b) CommitBERT input

Figure 3: Illustration of a code modification example in git diff (a) and method of taking it to the input of CommitBERT (b). (b) shows that all code modification lines in (a) are not used, and only changed lines are as input. So, in this example, code modification (a) includes `return a - b`, but not in the model input (b).

Metric	Initial Weight	Python	PHP	JavaScript	Java	Go	Ruby
BLEU-4 (Test)	(a) Random	7.95	7.01	8.41	7.60	10.38	7.17
	(b) RoBERTa	10.94	9.71	9.50	6.40	10.21	8.95
	(c) CodeBERT	12.05	13.06	10.47	8.91	11.19	10.33
	(d) CodeBERT + Code-to-NL	**12.93**	**14.30**	**11.49**	**9.81**	**12.76**	**10.56**
PPL (Dev)	(a) Random	144.60	138.39	195.98	275.84	257.29	207.67
	(b) RoBERTa	76.02	81.97	103.48	164.32	122.70	104.68
	(c) CodeBERT	68.18	63.90	94.62	116.50	109.43	91.50
	(d) CodeBERT + Code-to-NL	**49.29**	**47.89**	**75.53**	**77.80**	**64.43**	**82.82**

Table 2: Commit message generation result for 4 initial weights. In (c), CodeBERT is used as the initial weight. And (d) uses the weight trained on the Code-to-NL task in CodeSearchNet with CodeBERT as the initial weight. As a result, it shows BLEU-4 for the test set after training and the best PPL for the validation set in the during training.

Initial Weight	Input Type	BLEU-4
RoBERTa	(a) All code modification	10.91
	(b) Only changed lines (**Ours**)	**12.52**
CodeBERT	(a) All code modification	11.77
	(b) Only changed lines (**Ours**)	**13.32**

Table 3: The result of generating the commit message for the input type after collecting 4135 data with only source code change among the data of Loyola et al. (2017). (a) uses entire git diff(unidiff) as input, and (b) uses only the changed line according to Section 5.1 as input.

6.3 Ablation study on initial weight

We do an ablation study while changing the initial weight of the model for 345K datasets in six programming languages collected in Section 4. As mentioned in 5.2, when the model weight with high comprehension in the code domain is used as the initial weight, it is assumed that the large gap in contextual representation between PL and NL would be greatly reduced. To prove this, we train the commit message generation task for four weights as initial model weights: Random, RoBERTa[6], CodeBERT[7], and the weights trained on the Code-to-NL task(Section 3.3) with CodeBERT. Except for this initial weight, all training parameters are the same.

Table 2 shows BLEU-4 for the test set and PPL for the dev set for each of the four weights after training. As a result, using weights trained on the Code-to-NL task with CodeBERT as the initial weight shows the best results for test BLEU-4 and dev PPL. It also shows good performance regardless of programming language.

7 Conclusion and Future Work

Our work presented a model summarizing code modifications to solve the difficulty of humans manually writing commit messages. To this end, this paper proposed a method of collecting data, a method of taking it to a model, and a method of improving performance. As a result, it showed a successful result in generating a commit message using our proposed methods. Consequently, our work can help developers who have difficulty writing commit messages even with the application.

Although it is possible to generate a high-quality commit message with a pre-trained model, future studies to understand the code syntax structure remain in our work. As a solution to this, CommitBERT should be converted to AST (Abstract

[6] https://huggingface.co/roberta-base

[7] https://huggingface.co/microsoft/codebert-base

Language	Reference / Generated
Python	added figsize to plot methods
	added figure size to plot_weights
PHP	added default value to fieldtype
	Added default values of the fieldtype
JavaScript	Fix missing = in delete uri
	Fixed an issue with delete
Java	Fixed the parsing of orders without a cid
	Fix bug in exchange order
Go	Use ioutil . Discard for benchmark
	Use ioutil . Discard for logging
Ruby	fixing schema validation issues with CCR export
	fixing validation of ccr export

Table 4: The result of generating the commit message for six languages (Python, PHP, Go, Java, JavaScript, and Ruby) and the corresponding reference. We used the (d) model of Table 2.

Syntax Tree) before code modification is taken into the encoder like (Liu et al., 2020).

Acknowledgments

The author would like to thank Gyuwan Kim, Dongjun Lee, Mansu Kim and the anonymous reviewers for their thoughtful paper review.

References

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

SRP van Hal, Mathieu Post, and Kasper Wendel. 2019. Generating commit messages from git diffs. *arXiv preprint arXiv:1911.11690*.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*.

Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Automatically generating commit messages from diffs using neural machine translation. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 135–146. IEEE.

Siyuan Jiang and Collin McMillan. 2017. Towards automatic generation of short summaries of commits. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 320–323. IEEE.

Shangqing Liu, Cuiyun Gao, Sen Chen, Nie Lun Yiu, and Yang Liu. 2020. Atom: Commit message generation based on abstract syntax tree and hybrid ranking. *IEEE Transactions on Software Engineering*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Zhongxin Liu, Xin Xia, Ahmed E Hassan, David Lo, Zhenchang Xing, and Xinyu Wang. 2018. Neural-machine-translation-based commit message generation: how far are we? In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 373–384.

Pablo Loyola, Edison Marrese-Taylor, and Yutaka Matsuo. 2017. A neural architecture for generating natural language descriptions from source code changes. *arXiv preprint arXiv:1704.04856*.

Andreas Mauczka, Florian Brosch, Christian Schanes, and Thomas Grechenig. 2015. Dataset of developer-labeled commit messages. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 490–493. IEEE.

Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*.

Lun Yiu Nieb, Cuiyun Gaoa, Zhicong Zhongc, Wai Lamb, Yang Liud, and Zenglin Xua. Coregen: Contextualized code representation learning for commit message generation.

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 of the Association for Computational Linguistics*, pages 311–318.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Davide Spadini, Maurício Aniche, and Alberto Bacchelli. 2018. Pydriller: Python framework for mining software repositories. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 908–911.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Shengbin Xu, Yuan Yao, Feng Xu, Tianxiao Gu, Hanghang Tong, and Jian Lu. 2019. Commit message generation for source code changes. In *IJCAI*.