VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANASANGAMA, BELAGAVI - 590018

Project Phase 1 Report

On

AUTOMATIC GENERATION OF GIT COMMIT MESSAGES

Submitted in partial fulfillment for the award of degree of

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by
ANCHAL GUPTA 1BG18CS009
ANTARA BHAVSAR 1BG18CS014
CHITHRA SMITHA H M 1BG18CS024
M S NAGAJYOTHI 1BG18CS066
Internal Guide
Prof. Aashitha L Shamma
Dept.of CSE
BNMIT,Bengaluru

Vidyayāmruthamashnuthe

B.N.M Institute of Technology

Approved by AICTE, Affiliated to VTU, Accredited as grade A Institution by NAAC.
All UG branches – CSE, ECE, EEE, ISE &Mech.E accredited by NBA for academic years 2018-19 to 2020-21 & valid upto 30.06.2021
Post box no. 7087, 27th cross, 12th Main, Banashankari 2nd Stage, Bengaluru- 560070, INDIA
Ph: 91-80- 26711780/81/82 Email: principal@bnmit.in, www. bnmit.org

Department of Computer Science and Engineering

2021-2022

B.N.M Institute of Technology

Approved by AICTE, Affiliated to VTU, Accredited as grade A Institution by NAAC.
All UG branches – CSE, ECE, EEE, ISE &Mech.E accredited by NBA for academic years 2018-19 to 2020-21 & valid upto 30.06.2021
Post box no. 7087, 27th cross, 12th Main, Banashankari 2nd Stage, Bengaluru- 560070, INDIA
Ph: 91-80- 26711780/81/82 Email: principal@bnmit.in, www. bnmit.org

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vidyayāmruthamashnuthe

CERTIFICATE

Certified that the Mini Project entitled **AUTOMATIC GENERATION OF GIT COMMIT MESSAGES** carried out by Ms.**Anchal Gupta** USN 1BG18CS009**, Ms.AntaraBhavsar** USN 1BG18CS014**, Ms.Chithra Smitha HM** USN 1BG18CS024**,Ms.MS Nagajyothi** USN 1BG18CS066 are bonafide students of VII Semester B.E., B.N.M Instituteof Technology in partial fulfilment for the Bachelor of Engineering in COMPUTER SCIENCE AND ENGINEERING of the Visvesvaraya Technological University, Belagavi during the year 2020-21. It is certified that all corrections/ suggestions indicated for internal Assessment have been incorporated in the report deposited in the departmental library. The projectreport has been approved as it satisfies the academic requirements in respect of Project workprescribed for the said degree.

Prof.Aashitha L Shamma	**Dr. Sahana D. Gowda**	**Dr.Krishnamurty GN**
Assistant Professor	**Professor and HOD**	**Principal**
Department of CSE	**Department of CSE**	**BNMIT,Bengaluru**
BNMIT,Bengaluru	**BNMIT,Bengaluru**	

Name & Signature

Examiner 1:

Examiner 2:

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crowned my effort with success.

We would like to thank **Shri. Narayan Rao R Maanay**, Secretary, BNMEI, Bengaluru for providing the excellent environment and infrastructure in the college.

We would like to sincerely thank **Prof. T J Rama Murthy**, Director, BNMIT, Bengaluru for having extended his constant support and encouragement during the course of this project.

We would like to sincerely thank **Dr. S Y Kulkarni** ,Additional Director, BNMIT, Bengaluru for having extended his constant support and encouragement during the course of this project.

We would like to express my gratitude to **Prof. Eishwar N Maanay**, Dean, BNMIT, Bengaluru for his relentless support and encouragement.

We would like to thank **Dr. Krishnamurthy G N**, Principal, BNMIT, Bengaluru for his constant encouragement.

We would like to thank, **Dr. Sahana D. Gowda**, Professor & Head of the Department of Computer Science and Engineering for the encouragement and motivation she provides.

We would also like to thank **Mrs.Aashitha L Shamma,** Assistant Professor, Department of Computer Science and Engineering for providing us with her valuable insight and guidance wherever required throughout the course of the project and its successful completion.

Anchal Gupta 1BG18CS009

Antara Bhavsar 1BG18CS014

Chithra Smitha HM 1BG18CS024

M S Nagajyothi 1BG18CS066

ABSTRACT

Commit messages are a valuable resource in com-prehension of software evolution, since they provide a record of changes such as feature additions and bug repairs. Unfortunately, programmersoften neglect to write good commit messages. Different techniques have been proposed to help programmers by automatically writing these messages. These techniques are effective at describing what changed, but are often verbose and lack context for understanding the rationale behind a change. In contrast, humans write messages that are short and summarize the high-level rationale. In this paper, we adapt Neural Machine Translation (NMT) to automatically “translate” diffs into commit messages. We will train an NMT algorithm using a corpus of diffs and human-written commit messages from the top 1k GitHub projects. We will also design a filter to help ensure that we only train the algorithm on higher-quality commit messages. Our evaluationuncovers a pattern in which the messages we generate tend to be either very high or very low quality. Therefore, we create a quality-assurance filter to detect cases in which we are unable to produce good messages, and return a warning instead.

Table of Contents

CHAPTER 1

INTRODUCTION

Commit messages are a valuable resource in comprehension of software evolution, since they provide a record of changes such as feature additions and bug repairs. Unfortunately, programmers often neglect to write good commit messages.

Different techniques have been proposed to help programmers by automatically writing these messages.

These techniques are effective at describing what changed, but are often verbose and lack context for understanding the rationale behind a change. In contrast, humans write messages that are short and summarize the high level rationale.

In this project we will adapt Neural Machine Translation (NMT) to automatically "translate" diffs into commit messages. We trained an NMT algorithm using a corpus of diffs and human-written commit messages from the top 1k Github projects.

We designed a filter to help ensure that we only trained the algorithm on higher-quality commit messages. Our evaluation uncovered a pattern in which the messages we generate tend to be either very high or very low quality.

Therefore, we created a quality-assurance filter to detect cases in which we are unable to produce good messages, and return a warning instead.

1.1 MOTIVATION

Commit messages are useful in the long term for program comprehension and maintainability, but cost significant time and effort in the short term. These short term pressures lead programmers to neglect writing commit messages, studies and researches point out that programmers use commit messages for two reasons:

1) to summarize what changed

2) to briefly explain why the change was necessary.

Automated commit message generation is still very much an open problem. Different code change embeddings, for example by embedding the before and after state of the code separately, or focusing on specific types of commits may improve the quality of generated commit messages.

Thus, from this model we would aim at obtaining substantial reduction in the manual effort of generating an abstract commit message to increase understandability of code changes in repositories.

1.2 PROBLEM STATEMENT

“The aim of this project is to building a tool that summarizes changes between the control flow of program versions and generates an efficient commit message”. In this project we target the problem of automatically generating the git commit messages by implementing the machines learning and neural network over the code difference to obtain an abstract and efficient message.

1.3 OBJECTIVES

- The **git commit message automation** model is being build to reduce the developer effort in understand the ever increasing codebase. Commit messages are natural language descriptions of changes in source code.
- When a programmer updates code, a typical procedure is to upload the change to a version control system with a short commit message to describe the purpose of the change.
- E.g., “adds support for 9 inch tablet screen size.” The repository stores the message alongside a diff that represents the difference between the current and previous version of the affected files.

The **main objective** of this project are :

- Build a learning algorithm to generate commit messages by processing it through history of commit messages and standardizing the format of git commit messages
- Grade the generated commit messages based on the standard metrics

1.4 SUMMARY

- Git is a version control system which helps to keep the team updated with the changes that is made in the code. We use git commit messages to keep track of the changes that is made in the code by the team.

- Thus, commit messages are a valuable resource in comprehension of software evolution, since they provide a record of changes such as feature additions and bug repairs.
- Due to increased work pressure the developer neglect the formation of abstract and meaningful commit message.
- Thus, a git commit message automation model can be implemented to reduce the developers effort of writing an efficient message and thereby improving the understanding for the everchanging codebase**.**

CHAPTER 2

INTRODUCTION

A detailed literature survey was done for seven papers, by which we understood the different methodologies used for developing an algorithm to automatically generate commit messages. The literature survey helps us to understand the data collection, pre-processing of the data and algorithms applied for the generation of the commit message pattern.

The data collected from GitHub undergoes processing, cleaning, tokenization and filtration process. After pre-processing of the data, an algorithm is applied so as to generate a commit message. Different methods that can be applied for feature extraction are NMT algorithm using RNN encoder and decoder, NNGen algorithm. After the generation of the commit message, the evaluation of the messages is carried out either by human evaluation or by automatically evaluating the generated messages.

2.1 LITERATURE SURVEY

The approach takes as input a new diff and a training set, and outputs a one-sentence commit message for the new diff. It first extracts diffs from the training set. Next, the training diffs and the new diff are represented as vectors in the form of “bags of words”. In a bag-of-words model, the grammar and the word order of a diff are ignored, only term frequencies are kept. We refer to this kind of vector as a diff vector. Then, NNGen calculates the cosine similarity between the new diff vector and each training diff vector, and selects the top k training diffs with highest similarity scores. After that, the BLEU-4 score between the new diff and each of the top-k training diffs are computed. The training diff with the highest BLEU-4 score is regarded as the nearest neighbor of the new diff. Finally, it simply outputs the reference message of the nearest neighbor as the final result. In summary, given a new diff, the algorithm will first find its nearest neighbor in the training set, then reuse the reference message of the nearest neighbor as the generated message for the new diff. First, Jiang et al.’s experimental results were analysd. It was found that there are noisy commit messages in their dataset, and that the good performance of NMT benefits from those noisy commit messages. Then, a simple, nearest-neighbor-based approach, named NNGen, to generate short commit messages from diffs was proposed. The experimental results show that NN Gen is much faster and performs better than NMT on Jiang et al.’s dataset and the cleaned dataset. Finally, further analysis of commit message generation was conducted, and some challenges were discussed in the road ahead for this task to inspire other researchers [1].

A dataset was created for the Java and the C# programming language. To gather these, the GitHub API was used to retrieve the top 1000 most-starred repositories for both languages. The collected data needs to be divided into three distinct sets that have no overlap. These sets will be used as training, validation or testing data respectively. The approach to generate commit message from a git diff file will be the same as Jiang et al., namely with the neural machine translation approach that does sequence to sequence learning. The model that is to be used is from Bahdanau et al., a encoder-decoder model that uses attention to attend to the important parts of the git diff sequence x during the generation of the commit message sequence y. The model was trained with Stochastic Gradient Descent (SGD) with an initial learning rate of 0.1. The learning rate was reduced with factor 0.1 if no improvements were made on the validation loss for 10 epochs. Early stopping of the training process was done if no validation loss improvement was seen for 20 epochs. All models were trained on a Nvidia GeForce GTX 1660 GPU with 6GB of memory. After each epoch, the intermediate model was saved and the model with the least validation loss was used for evaluation.

During training, the model is evaluated on the validation data by computing the cross entropy loss. The model with the lowest loss will be considered the best model. During testing, for all models the BLEU scores are computed according to and the ROUGE F1 scores are computed according to, which is a combination ROUGE precision and recall. The ROUGE-1 and ROUGE-2 scores are based on the overlap of unigrams and bigrams respectively. ROUGE-L is based on the longest common subsequence (LCS) and ROUGE-W builds further on this by using weighted LCSes which favours consecutive subsequence. Experiments showed that a reproduction of the attentional RNN encoder-decoder model from Jiang et al. achieves slightly better results on the same dataset. This confirms the reproducibility of under similar circumstances.

However, the model was unable to generate commit messages of high quality for any input dataset that was processed with the novel technique. The BLEU score dropped by at least 78% for any dataset. This exposed the underlying problem of the original model, which seems to score high by remembering (long) path names and frequently occurring messages from the training set [2].

The commit data set provided by Jiang and McMillan, which contains 2M commits was used. The data set includes commits from top 1k Java projects (ordered by the number of stars) in GitHub**.** The strategy was, in a nutshell, to 1) collect a large repository of commits from large projects, 2) filter the commits to ensure relatively high-quality commit messages, and 3) train a Neural Machine Translation algorithm to "translate" from diffs to commit messages using the filtered repository. It was then evaluated the generated commit messages in two ways. First, an automated evaluation using accepted metrics and procedures from the relevant NMT literature was conducted. Second, as a verification and for deeper analysis, an experiment with human evaluators was also conducted. NMT algorithm succeeded in identifying cases where the commit had a similar rationale to others in the repository. The human evaluators rated a large number of the generated messages as very closely matching the reference messages. However, the algorithm also generated substantial noise in the form of low-quality messages. A likely explanation is that these include the cases that involve new insights which the NMT algorithm is unable to provide. While creating these new insights from the data is currently beyond the power of existing neural network-based machine learning. at a minimum we would like to return a warning message to the programmer to indicate that we are unable to generate a message, rather than return a low-quality message. Therefore, we created a Quality Assurance filter in Section VII. This filter helped reduce the number of low-quality predictions [3].

The idea was to generating a commit message through the CodeBERT model with the dataset. They feed inputs code modification and commit message to CodeBERT and use pre-trained weights more efficiently to reduce the gap in contextual representation between programming language (PL) and natural language (NL). Collected 345K code modification and commit message pair datasets from 52K GitHub repositories.

With the advent of sequence-to-sequence learning, 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 based on source input X and target input Y. 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 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.

The actual method is to 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 the CommitBERT in the same way, added tokens and deleted tokens of similar types are connected to each other, and sentence separators are inserted between them. To reduce the gap difference between two domains(
PL, NL), use the pretrained CodeBERT as the initial weight. Furthermore, determine
that removing deleted tokens from our dataset is similar to the Code-to-NL task in CodeSearchNet. Using this feature, 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 [4].

In this paper, the proposed idea is a novel approach CODISUM for the commit message generation problem, addressing the limitations. In particular, firstly extract both the
code structure and the code semantics from the code changes. For the code structure, identify all the class/method/variable names and replace them with the corresponding placeholders. For the code semantics, segment each class/method/variable name into several single words. Here, the word segmentation is based on the well-known naming conventions widely adopted by developers.

In summary, the main contributions of this paper include:

- Propose a commit message generation approach CODISUM that jointly learns the representations of both code structure and code semantics. The model can also
mitigate the OOV problem.

- Conduct experimental evaluations on real data, demonstrating that the proposed approach significantly outperforms the state-of-the-art in terms of the accuracy of generating commit messages.

The proposed CODISUM model contains an encoder part and a decoder part. The input of the model is a piece of source code change. Then, in the encoder part, we first extract the code structure from the input by recognizing the identifiers (i.e., the names of classes, methods, and variables). Then replace all the identifiers with the corresponding placeholders and then learn the representations of the resulting code structure. For each identifier, also learn its semantic representation, and combine the learned semantics with the corresponding placeholder as the overall representation of this identifier. The overall representation of each input word is then fed into an attention layer to obtain the final representation of the input code change. For the decoder part, we use a multi-layer unidirectional GRU to generate a sequence of words as the initial commit message. Meanwhile, incorporate the copying mechanism to allow directly copying the OOV words from the code changes. Such copying applies on the code structure, which directly copies the placeholders instead of the original input words. Finally, substitute the placeholders with the corresponding input words as the final generated commit message.

The following approaches are compared in the experiments:

- **NMT :** NMT uses attention-based RNNs to translate diffs into commit messages. It treats diffs and commit messages as two different languages.
- **NNGen :** NNGen is an information retrieval approach. It represents diffs as "bags-ofwords" vectors, and re-uses the commit message from the most similar diff calculated with cosine similarity.
- **CopyNet :** CopyNet is a text summarization method, which incorporates copying mechanism into the decoder to allow copying words from source to target.
- **CODISUM :** CODISUM is the proposed approach built upon the NMT model [5].

The paper presents an approach for generating automatic commit messages based on the code changes included in a change set. ChangeScribe extracts and analyzes the differences between two versions of the source code, and also, performs a commit characterization based on the stereotypes of methods modified, added and removed. The outcome is a commit message that provides an overview of the changes and classifies and describes in detail each of the changes made by a developer in the source code. Furthermore, we conducted a survey in which 23 developers performing 107 evaluations of 50 commit messages from six open source systems and equivalent number of commit messages generated by ChangeScribe. According to the case study results, 84% of the generated commit messages do not miss essential information required to understand the changes, 25% of them are concise, and in 39% of the cases the generated message is easy to read and understand. The results also demonstrate that while the original commit messages miss some very important information that can hinder understanding of the changes what and why in 40% of the cases, ChangeScribe's commit messages have been rated to have this deficiency in just 16% of the cases. Finally, in 51% of the cases the study participants preferred ChangeScribe's commit messages to the ones written by the original developers. When considering only large commits, ChangeScribe's commit messages are preferred in 62% of the cases; and when considering only small commits ChangeScribe's commit messages are preferred in 54% of the cases. All in all, the evaluation indicates that ChangeScribe can be useful as an online assistant to aid developers in writing commit messages or to automatically generate commit messages when they do not exist or their quality is low. Moreover, the length of the message is an important attribute that should be controlled by the developer to avoid unnecessary information without truncating the message. ChangeScribe provides developers with a filter based on the changes impact, that reduce the size of the message without truncating the descriptions [6].

Writing a good commit message is helpful for software development but challenging for developers. Although a number of automated commit message generation (CMG) approaches have been proposed, a large proportion of messages generated by them are semantically irrelevant to the ground truths, which can mislead developers and hinder their usage in practice. Instead of proposing yet another CMG approach, tackling this problem in another promising way: proposing an automated Quality Assurance framework for COMmit message generation (QAcom). QAcom can automatically predict and filter out the generated messages that are semantically irrelevant to the ground truths in an unsupervised way. In particular, QAcom combines a Collaborative-Filtering-based component and a Retrieval-based component to estimate the semantic relevance of generated messages to references. We evaluate the effectiveness of QAcom with three state-of-theart CMG approaches on three public datasets. Automatic and human evaluation demonstrate that QAcom can effectively filter out semantically-irrelevant generated messages and successfully preserve semantically-relevant ones. QAcom is also shown to outperform a supervised quality assurance method named QA filter [7].

2.2 PROPOSED METHODOLOGY

- In this approach, code 'diffs' are translated into commit messages automatically using Neural Machine Translation[NMT] followed by generating commit messages automatically by taking code difference as input.

- The machine will traine an NMT algorithm using a corpus of diffs and human-written commit messages from the Github projects.
- A Verb/Direct-Object [V-DO] filter ensures that algorithm generates only higher-quality commit messages.
- It automatically filtering out the semantically-irrelevant messages and preserves the semantically-relevant ones.
- The Quality Assurance Framework has embedded the Plan, Implement, Monitor/Review, and Improve quality assurance and continuous improvement model. This framework enables to achieve its strategic promises, ensure quality outcomes and meet its statutory and regulatory obligations.

2.3 SUMMARY

From the literature survey, we have observed that there are various ways to implement the algorithm to generate the commit messages. The NMT algorithm using RNN encoder and decoder will be incorporated in our project as it gives the best results for generating the commit message. A quality assurance filter will also be implemented in order to filter only high quality commit messages and discard the low quality messages.

REFERENCES

[1] Zhongxin Liu, Zhejiang University, China ,Xin Xia , Monash University,Australia Ahmed E. Hassan,Queen's University,Canada (2018) "**Neural-Machine-Translation-Based Commit Message Generation: How Far Are We?"**

[2] Sven van Hal, Delft University of Technology, S.R.P.vanHal@student.tudelft.nl
Mathieu Post, Delft University of Technology, M.Post@student.tudelft.nl
Kasper Wendel, Delft University of Technology, K.Wendel@student.tudelft.nl
(2019) "**Generating Commit Messages from Git Diffs"**
arXiv:1911.11690v1 [cs.SE] 26 Nov 2019

[3] Siyuan Jiang, Ameer Armaly, and Collin McMillan, Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA (2017) **"Automatically Generating Commit Messages from Diffs using Neural Machine Translation"** arXiv:1708.09492v1 [cs.SE] 30 Aug 2017

[4] Tae-Hwan Jung Kyung Hee University (2021) "**CommitBERT: Commit Message Generation Using Pre-Trained Programming Language Model"**
arXiv:2105.14242v1 [cs.CL] 29 May 2021

[5] Shengbin Xu1 , Yuan Yao1 , Feng Xuy, Alibaba Group, USA (2019) **"Commit Message Generation for Source Code Changes"**

[6] Luis Fernando Cortés-Coy, Mario Linares-Vásquez, Jairo Aponte, Denys Poshyvanyk, Universidad Nacional de Colombia, Bogotá, Colombia, The College of William and Mary, Williamsburg, VA, USA (2014) **"On Automatically Generating Commit Messages via Summarization of Source Code Changes"**

[7] Wei Wang, Meng Yan, Zhongxin, Ling Xu, Xin Xia, Xiaohong Zhang2, Dan Yang Key Laboratory of Dependable Service Computing in Cyber Physic (2021) **"Quality Assurance for Automated Commit Message Generation**"
2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)