Leveraging Prompt Engineering for Enhanced Code Summarization

Burgers Wout Hu Weicheng Ibanez Elena

Macsim Violeta
vmacsim@student.tudelft.nl
5498031

Nujhat Nawmi van Driel Jort

van Vliet Vincent

Abstract

A clear and well-documented LaTeX document is presented as an article formatted for publication by ACM in a conference proceedings or journal publication. Based on the "acmart" document class, this article presents and explains many of the common variations, as well as many of the formatting elements an author may use in the preparation of the documentation of their work.

Keywords

Do, Not, Us, This, Code, Put, the, Correct, Terms, for, Your, Paper

ACM Reference Format:
Burgers Wout, Hu Weicheng, Ibanez Elena, Macsim Violeta, Nujhat Nawmi, van Driel Jort, and van Vliet Vincent. 2018. Leveraging Prompt Engineering for Enhanced Code Summarization. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Collaboration is an important aspect of software development and has been greatly aided by version control frameworks, a type of timestamped system that mirrors and controls changes made to files. Since 2005, *git* has been the most used version control system by developers [10], contributing to the creation of many successful projects we now use through one simple command: `git commit`.

This command enables people to easily filter through different code versions by providing a `commit message`, a summary selected by the author of a commit. In general, it provides concise information about the most significant changes from the previous state of code. However, finding the right words to condense a large amount of work into meaningful information without becoming overly detailed can be difficult for people who have not been trained to master word manipulation. Fortunately, smart developers have already devised a solution to this problem, which they may not be aware of: using a Large Language Model (LLM) to complete this task.

LLMs have rapidly gained popularity in software engineering [5], [8], with some becoming increasingly specialized in particular tasks. *DeepCode* is designed for code improvement and bug detection, *CodeX* [2] for code creation, *CodeT5* [12] for code summarization, and there are many other categories in this list generally named. And, while these contribute significantly to the development of our world, almost none attempt to fill the earlier mentioned gap, which can be easily streamlined. In a literature survey by Hou et al. [7], out of nearly 230 papers, only 10 models were used for code summary, but all were tailored to create better and more extensive documentation, not for concise formats such as commit messages.

Once an LLM is capable of analyzing and summarizing the `git diff` logs, the task of implementing an IDE extension is nearly finished. Together with using the terminal to perform `git` commands, anyone can create their own IDE extension to automatically generate commit messages.

This study aims to see if current code summarization language models can be specifically improved for this task. Experimenting with prompting techniques on three large language models, not necessarily specialized in code summary, will aid in providing the most fitting recipe to generate commit messages, potentially inspiring more work to streamline the usage of *git*. The result of this endeavor will be a Visual Studio Code extension that any developer can use to simplify their workflow.

2 Previous Work

Commit messages are crucial in software development and are being widely used to enhance code readability. However, writing a summarized code message that captures the entire purpose can be time-consuming. So far, the existing approaches for commit message generation can be classified into three different categories, template-based, retrieval-based, and hybrid approaches. To improve the performance of these methods, researchers can either fine-tune the models or experiment with different prompt templates to elicit the LLM to provide the exact response that the human expects. The latter technique is commonly referred to as prompt engineering.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/XXXXXXX.XXXXXXX

2.1 Template-based approaches

This method uses predefined templates to create placeholders to summarize code changes for generating commit changes [3].

Such tools have been developed with the purpose of making commit messages clearer and more informative by automatically structuring code change information. Below are some notable tools:

- **ChangeScribe**: This tool analyzes code differences between two versions and uses predefined templates to generate commit messages, focusing on summarizing key changes [3].
- **DeltaDoc**: This tool creates concise, human-readable documentation by capturing logical paths and predicates, fitting changes into predefined document templates [3].
- **ARENA**: This tool identifies code changes, analyzes them in the context of the associated project, and generates a commit message based on this information [3].

Tools like this can be very helpful. However, while template-based methods for commit message generation exist, He et al. highlight their limited inability to handle diverse or complex code changes [6]. These methods struggle with intricate edits and are effective only when the changes fit well within pre-defined rules [4]. As such, Dong et al. have begun addressing these limitations by employing graph-based fine-grained representations by explicitly capturing edit operations and token-level changes, enabling more precise commit message generation even for complex code modifications [4].

2.2 Retrieval-Based Approaches

This approach revolves around finding and reusing existing commit messages that closely match new code changes.

Many retrieval-based approaches rely on information retrieval techniques to find similar code diffs and their associated commit messages, such as ChangeDoc and NNGen. These approaches focus on retrieving semantically or syntactically similar commits and directly reusing their messages as templates for new ones [11]. For instance, NNGen utilizes the Nearest Neighbor algorithm to match similar diffs from the training data, demonstrating notable speed and efficiency, with performance improvements of up to 21% in BLEU-4 scores over neural models [3]. Similarly, ChangeDoc reuses commit messages by analyzing the syntactic and semantic information of the code before and after changes, leveraging large commit repositories for accurate recommendations [3].

While these methods excel at producing readable messages, they struggle with unique or highly specific code changes, often resulting in inaccurate outputs. This limitation has led to the emergence of hybrid approaches, which combine retrieval-based methods with neural models. These models aim to balance adaptability and accuracy, making it possible to handle a wider variety of code changes while preserving the clarity and relevance of retrieval-based outputs

2.3 Hybrid approaches

To address the limitations of retrieval-based approach, the RACE framework combines retrieval-based and neural-based methods, using the retrieved commit message as an exemplar to guide the neural model. RACE introduces an "exemplar guider" to evaluate and adapt the similarity between retrieved messages and current code changes, thereby minimizing errors from irrelevant examples [11]. This approach makes the most of what retrieval methods do well while addressing their shortcomings, leading to commit messages that are clearer, more concise, and easier to understand.

Moreover, COME (Commit Message Generation with Modification Embedding) introduces significant advancements in commit message generation by addressing these shortcomings [6]. Unlike previous methods that treat code changes as pre-filled text sequences [3] or abstract syntax trees [4], COME employs modification embeddings derived from the edit distance algorithm. These embeddings enable a fine-grained representation of code changes while reducing redundancy. Additionally, COME employs a self-supervised generative task to enhance contextual understanding and a decision algorithm to effectively combine retrieval and translation methods. By combining retrieval-based and translation-based approaches through a decision algorithm, COME achieves a 9.2% improvement over state-of-the-art methods on automated metrics and an 8.0% enhancement in human evaluations.

2.4 Prompt engineering

While prompt quality and specificity have an impact on LLM outcomes [1], [15], few studies assess the potential of models using prompt engineering. There are many different prompting strategies, such as giving the LLM examples (few-shot prompting or retrieval-augmented generation [14]), providing no information at all (zero-shot prompting), or using chain-of-thought prompting for white-box modeling. When compared to in-context prompting alone, RAG can increase ChatGPT's capacity to write commit messages by as much as 66%, according to a recent study [13]. However, Kojima et al. [9] have shown that simply adding the phrase "Let's think step by step" to the prompt can significantly improve LLMs.

3 Methodology

The auto-commit message generation pipeline is broken into 3 main phases:

3.1 Dataset selection

We select a suitable dataset, called **CommitBench**. The dataset is chosen because... To use the dataset more efficiently, we employ some data pre-processing techniques. This involves extracting meaningful data from contexts. In our cases, the `git diff` logs are inputs and the commit messages are output. Then, we tokenize the git diff logs into manageable segments and normalize the commit messages. Normalizing means keeping the same formatting and abbreviations to focus only on the content.

3.2 Model selection

We select a suitable LLM. We start with **DeepInfra** for this task, because DeepInfra is a pre-trained model which makes the experiments easier to do. After choosing the LLM, we fine-tune it by prompt engineering on our dataset to adapt the model for better message generation. This means adjusting the prompts to yield the best possible outcome, as LLMs are sensitive to prompts.

3.3 Evaluation metrics selection

We mainly use **F1** as the evaluation metric, essentially comparing the generated commit message with human-written ones. However,

it is slightly biased because engineers might write the commits poorly. Therefore, another metric called **BLEU** is employed.

References

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems* 33 (2020), 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/file/fd502f6611e62d6b9e9cfdc245e0f8cc-Paper.pdf Retrieved May 31, 2023.

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Neel Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Maxim Pavlov, Adam Power, Lukas Kaiser, Matthew Bavarian, Clemens Winter, Philippe Tillet, Felipe Such, Daniel Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H Guss, Carl Nicholas, Armand Joulin, Peter Welinder, Vladimir Karpukhin, Sanjay Nerurkar, Vedant Mishra, Jack Clark, Sam Coombs, Alec Radford, Ilya Sutskever, and Dario Amodei. 2021. Evaluating Large Language Models Trained on Code. *arXiv* (2021). https://doi.org/10.48550/arXiv.2107.03374

[3] Xiangping Chen, Yangzi Li, Zhicao Tang, Yuan Huang, Haojie Zhou, Mingdong Tang, and Zibin Zheng. 2024. ESGen: Commit Message Generation Based on Edit Sequence of Code Change. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension* (Lisbon, Portugal) *(ICPC '24)*. Association for Computing Machinery, New York, NY, USA, 112–124. https://doi.org/10.1145/3643916.3644414

[4] Jinhao Dong, Yiling Lou, Qihao Zhu, Zeyu Sun, Zhilin Li, Wenjie Zhang, and Dan Hao. 2022. FIRA: Fine-Grained Graph-Based Code Change Representation for Automated Commit Message Generation. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. 970–981. https://doi.org/10.1145/3510003.3510069

[5] Md Asraful Haque. 2024. LLMs: A Game-Changer for Software Engineers? *arXiv* (2024). https://doi.org/10.48550/arXiv.2411.00932

[6] Yichen He, Liran Wang, Kaiyi Wang, Yupeng Zhang, Hang Zhang, and Zhoujun Li. 2023. COME: Commit Message Generation with Modification Embedding. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Seattle, WA, USA) *(ISSTA 2023)*. Association for Computing Machinery, New York, NY, USA, 792–803. https://doi.org/10.1145/3597926.3598096

[7] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *arXiv* (2024).

[8] Haolin Jin et al. 2024. From LLMs to LLM-based Agents for Software Engineering: A Survey of Current, Challenges and Future. *arXiv* (2024). https://doi.org/10.48550/arXiv.2408.02479

[9] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large Language Models are Zero-Shot Reasoners. arXiv:2205.11916 [cs.CL] https://arxiv.org/abs/2205.11916

[10] Stack Overflow. 2022. Stack Overflow Developer Survey 2022. https://survey.stackoverflow.co/2022/#section-version-control-version-control-systems Accessed: 2024-11-27.

[11] Ensheng Shi, Yanlin Wang, Wei Tao, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2022. RACE: Retrieval-Augmented Commit Message Generation. arXiv:2203.02700 [cs.SE] https://arxiv.org/abs/2203.02700

[12] Yue Wang, Shafiq Joty Cheng, Chejiang Zhang, Yue Wan, Iuliia Safonova, Zheng Sui, Yingwei Li, and Tatsunori Hashimoto. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.

[13] Yifan Wu, Ying Li, and Siyu Yu. 2024. Commit Message Generation via ChatGPT: How Far Are We?. In *Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering* (Lisbon, Portugal) *(FORGE '24)*. Association for Computing Machinery, New York, NY, USA, 124–129. https://doi.org/10.1145/3650105.3652300

[14] Linghao Zhang, Hongyi Zhang, Chong Wang, and Peng Liang. 2024. RAG-Enhanced Commit Message Generation. *arXiv preprint arXiv:2406.05514* (2024).

[15] Yilun Zhou, Andreea I. Muresanu, Ziniu Han, Kevin Paster, Silviu Pitis, Hingjin Chan, and Jimmy Ba. 2023. Large language models are human-level prompt engineers. *arXiv* 2211.01910 (2023). https://arxiv.org/abs/2211.01910 arXiv preprint.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009