AI-Powered Commit Explorer (APCE)

Yousab Grees*, Polina Iaremchuk*, Ramtin Ehsani†, Esteban Parra*, Preetha Chatterjee†, Sonia Haiduc‡
*Department of Mathematics, Computer Science, and Data Science, Belmont University, Nashville, TN, USA
{yousab.grees@bruins.belmont.edu, polina.iaremchuk@bruins.belmont.edu, esteban.parrarodriguez@belmont.edu}
†Department of Computer Science, Drexel University, Philadelphia, PA, USA
{ramtin.ehsani@drexel.edu, preetha.chatterjee@drexel.edu}
‡Department of Computer Science, Florida State University, Tallahassee, FL, USA
shaiduc@fsu.edu

***Abstract*—Commit messages in a version control system provide valuable information for developers regarding code changes in software systems. Commit messages can be the only source of information left for future developers describing what was changed and why. However, writing high-quality commit messages is often neglected in practice. Large Language Model (LLM) generated commit messages have emerged as a way to mitigate this issue. We introduce the AI-Powered Commit Explorer (APCE), a tool to support developers and researchers in the use and study of LLM-generated commit messages. APCE gives researchers the option to store different prompts for LLMs and provides an additional evaluation prompt that can further enhance the commit message provided by LLMs. APCE also provides researchers with a straightforward mechanism for automated and human evaluation of LLM-generated messages. Demo link https://youtu.be/zYrJ9s6sZvo**

***Index Terms*—Large Language Models, GitHub, Automated Commit Messages, AI4SE, Code Summarization**

I. Introduction

As part of the routine software maintenance and evolution process of large software systems, developers often need to reference previous changes to fix bugs or implement new features. Changes to software artifacts are tracked through Version Control Systems (e.g., Git). Git allows developers to describe the changes to the software artifacts via a commit message containing a textual description of the changes within the commit as well as the rationale behind those changes [22].

Commit messages play a critical role in developers' understanding and communicating code changes and software maintenance. In particular, commit messages can be the only source of information left for future developers in long-lived projects [6]. Therefore, high-quality commit messages are crucial for the long-term maintenance and evolution of any software project. However, writing high-quality commit messages is often neglected in practice, leading to commit messages that are incomplete, ambiguous, non-informative, difficult to understand, or empty [4], [6], [16].

Automatic commit message generation aims to leverage computational methods to develop approaches to support developers in their software maintenance process by automatically generating meaningful commit messages that provide them information regarding *what?* and *why?* a set of code changes where made [2], [6]. Automated commit generation approaches can be classified into four main categories: rule-based, retrieval-based, learning-based, and hybrid [3].

The rise of Large Language Models (LLMs) and multi-agent frameworks composed of multiple LLM-based agents that interact and collaborate to solve complex problems or achieve goals beyond the capability of any single agent, has led to a growing body of research on using LLMs in various software engineering tasks [5], [11], [13], [14], [18]. Among them are LLM-based approaches for automatically generating commit messages [15], [20]–[22]. As these newer approaches continue to evolve and demonstrate promising performance [22], it is necessary to evaluate the newer approaches quickly and seamlessly, ensuring timely validation and comparison.

In this paper, we introduce the AI-Powered Commit Explorer (APCE), a tool aimed at assisting researchers in the generation and evaluation of commit messages using LLM-based approaches. APCE serves two primary purposes. First, APCE provides seamless integration into GitHub repositories, enabling the automatic generation of high-quality commit messages that describe both the what and the why of code changes. By selecting any commit within a GitHub-hosted repository, developers can use APCE to obtain a synthesized, contextual message generated by one or multiple LLM-based approaches, aiding them in their code understanding and long-term maintainability of the system. Second, APCE facilitates and streamlines empirical analysis and evaluation of commit messages generated by new LLM-based approaches with a built-in evaluation module. The evaluation module compares automatically generated commit messages to human-written counterparts and computes standard evaluation metrics (*e.g.*, BLEU, ROUGE-L, and METEOR) [12], [15]. Furthermore, the evaluation module supports the empirical evaluation of new automated commit message generation techniques by collecting users' feedback on the quality of the automated commit messages (*e.g.*, completeness, consistency, and informativeness). The full implementation of APCE is available on GitHub[1] and our replication package[2].

The rest of this paper is structured as follows. Section II will explain the works that inspired this tool. Section III presents the architecture and design of our tool, and Section IV outlines the tool's availability. Finally, section V will conclude the paper with future work and limitations.

[1]https://github.com/yousabg/AI-Powered-Commit-Explorer
[2]https://figshare.com/s/0f1f15af0ecb5aeedee2

II. Related Work

In this section, we briefly discuss existing work in automated commit message generation. Approaches for automatic generation of commit messages can be broadly classified into generation-based and retrieval-based methods [2], [7], [8], [19]. Retrieval-based approaches use Natural Language Processing (NLP) and Information Retrieval (IR) techniques to pick the most similar words or messages from a large dataset, whereas generation-based approaches use machine learning and large datasets to predict the words that the commit message should contain [22].

One of the first retrieval-based tools proposed to address this problem was ChangeScribe [2]. ChangeScribe utilized NLP to compare the code differences of a commit. Then, it formulates a short sentence based on that difference. Since then, several methods have been built to improve upon this work. For example, Liu et al. [9] proposed NNGen, which uses the nearest neighbors algorithm to retrieve the most similar code diff from a training set and reuse its commit message. Additionally, Wang et al. [17] introduced CoRec that combines retrieval and machine learning approaches. CoRec uses a decay sampling strategy during training to shift from ground truth to model-generated inputs and leverages the most similar commit to refine its output.

More recently, generation-based methods using neural architectures and pre-trained models have emerged [3], [10], [21]. FIRA, an approach that utilizes fine-grained graph representations of code changes and a graph-neural network encoder with a transformer-based decoder improved by a dual copy mechanism allowing flexible access to sub-token and integral token information [3]. The CCRep, a newer technique introduced by Liu *et al.* [10], leverages a pre-trained model to encode code changes and generate a commit message. Experiments using models such as $CCRep^{token}$, $CCRep^{line}$, and $CCRep^{hybrid}$ demonstrated improvement in the quality of generated messages. Zhang *et al.* [21] conducted a study evaluating commit message generation using popular open-source and closed-source LLMs such as ChatGPT and Llama. The two-phase evaluation consisted of a zero-shot setup, where prompts from a dataset of commit messages were sent through ChatGPT and Llama2, and then evaluated by BLEU and Rouge-L. Then, the messages were shuffled and evaluated by the two researchers. Surprisingly, LLM-generated messages were preferred over human-written ones in most cases—human messages were favored only 13.1% of the time.

APCE contributes to the study of automated commit generation by providing an easy-to-set-up framework to streamline the evaluation of LLM-based commit generation approaches.

III. Architecture

APCE is a web-based tool that supports the generation and evaluation of commit messages. APCE uses a web application architecture in which the front-end client is built using *Next.js*[3], a JavaScript web development framework, the back-end is hosted on a Flask[4] server, a lightweight Python web framework for building APIs and handling HTTP requests, and using a MySQL[5] database to store the data.

APCE has two modules: (i) a commit generation module and (ii) an evaluation module, as shown in Figure 1.

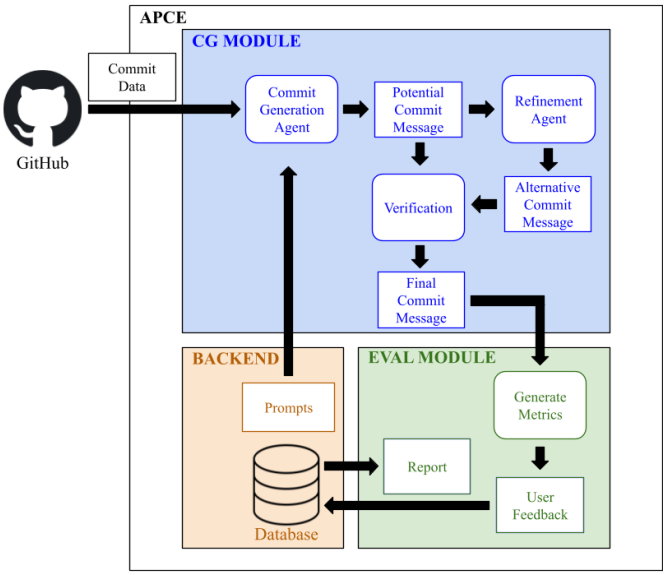


Fig. 1. APCE Architecture

The commit generation module (CG MODULE) uses a Multi-agent framework [5] to generate a commit message. In particular, APCE leverages two agents (i.e., a commit Generation Agent and a Refinement Agent) that engage in a multi-turn self-collaboration process.

The evaluation module (EVAL MODULE) supports the following functionality: approach management, consent form, data collection, user interaction, evaluation metric computation, and reporting, which allows researchers to work purely on analyzing results rather than building an infrastructure.

The APCE provides a consent form when a participant first loads the tool. Currently, the sample consent form is tailored to Belmont University, since the author of the tool conducted research at that university. The consent form used can be replaced to meet the researcher's specific institutional needs. The GitHub credentials require a user to make a choice by clicking presented 'Accept' button to proceed with a tool.

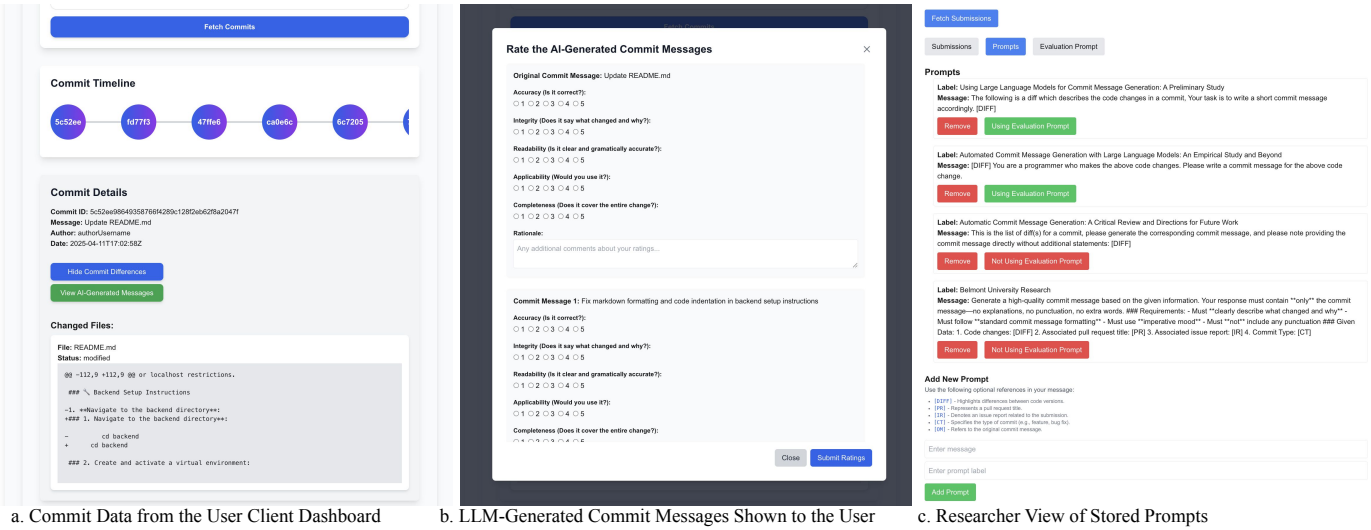
Assuming consent is accepted, the user is then prompted to share their GitHub token and username. The GitHub token is a digital authentication key that is used to access the OctoKit API[6], a collection of client libraries, that simplifies interaction with the GitHub API, and pulls the user's repositories and commits. For the sake of secure and correct final commit message generation, as emphasized in the consent form, personal data that can link back to the user is not collected, besides the data specific to the actual commit.

[3]https://nextjs.org/

[4]https://flask.palletsprojects.com/en/stable/

[5]https://www.mysql.com/

[6]https://github.com/octokit



a. Commit Data from the User Client Dashboard b. LLM-Generated Commit Messages Shown to the User c. Researcher View of Stored Prompts

Fig. 2. APCE GUI

A. Commit Generation Module

The commit generation module allows the user to select a repository. After selecting a repository, the user is presented with a list of all commits associated with that repository in a node timeline as shown in Figure 2a. When selecting a commit node, APCE populates with the details about the commit and enables a button to 'View AI Generate Messages'. When the user clicks on the button, the commit details will be replaced with a loading log while APCE generates a commit message for the selected commit using each of the configured commit generation approaches.

APCE uses two agents in the commit generation process. The commit generation agent uses a prompt to generate a commit message, while the refinement agent assesses the commit message generated by the commit generation agent. The refinement agent can be enabled or disabled for any approach. The default prompt used by the refinement agent is shown in Figure 3. The two-agent framework allows APCE to be easily modified to support any existing and future LLM-based commit generation approaches by adding or removing a prompt to the backend (See Section III-B for details on how to set up approaches in APCE).

For each of the approaches configured in APCE: 1) the commit generation agent will first generate a potential commit message using the corresponding prompt, 2) the refinement agent will asses the potential commit message generated by the commit generation agent and generate an alternative commit message if the potential commit message does not follow the format of a commit message, 3) APCE will compare the potential and the alternative commit messages and select one of the two.

If an alternative prompt is returned by the refinement agent, APCE will compare both the potential and the alternative commit messages and select one of the two. First, if either response is not a valid commit message based on the criteria, then the other response is chosen. If both are not valid, an error is generated for this message. If both are valid, then it checks if any of the generated commit messages are greater than 72 characters. If one of them is, then the client will choose the other commit message, since we prefer a commit message that is less than 72 characters for readability between tools and terminals [1]. Lastly, if both commit messages are less than 72 characters, then the verification will choose the longer commit message. If the user configures an approach to skip the refinement agent, APCE would only check the validity of the initial commit message by ensuring that the response is not an error and that the response is less than 200 characters. APCE uses the 200-character limit to prioritize returning a commit message over an error and avoiding overly verbose messages.

APCE uses the unified API service OpenRouter[7] to support access to various LLM models. In the default implementa-

> Evaluate the commit message below.
> If it fully meets all criteria, reply only with the exact same commit message.
> If it does not fully meet all criteria, reply only with a corrected commit message.
> Your response must contain nothing else—no explanations, no punctuation, no extra words.
> Criteria:
> - Must be less than 72 characters
> - Must use imperative mood (e.g., "Fix bug" instead of "Fixed bug")
> - Must clearly describe the change
> - Must not include explanations or reasoning
> - May describe multiple changes
>
> Commit message to evaluate: "[MESSAGE]"

Fig. 3. Refinement Agent prompt

[7]https://openrouter.ai/docs/quickstart

tion, APCE utilizes the DeepSeek[8] model on OpenRouter, which has consistently delivered the most accurate responses among free-tier models. To use OpenRouter, the developer or researcher must set up a free account and generate an API key. This key should be added to the code-base via a .env.local file in the following format:

```
NEXT_PUBLIC_OPENROUTER_API_KEY=<
    your_api_key_here>
```

If any errors occurred during an API call to the LLM to generate a commit message for a given approach, APCE will re-attempt after 5000ms. After 3 unsuccessful tries, APCE will return an error for that message.

Although the default LLM in APCE is deepSeek-R1 via OpenRouter, the tool can be customized to fit the needs of other researchers by changing the LLM model and provider by modifying the *api.js* file. In particular, if a researcher wants to change the LLM model used, then only the body of the API needs to be changed in the *getDeepSeekResponse()* method. On the other hand, if a researcher wants to add a different API service, they will have to write a new method to replace the existing *getDeepSeekResponse* method (for example, using a cron job or a simple js compiler).

B. Configuring Commit Generation Approaches in APCE

Using the research view (see Figure 2c), the user can configure one or more LLM-based commit generation approaches to be used by the commit generation module. The following optional references can be added to the approach to include the corresponding information from the commit itself:

- `[DIFF]` – Differences between code versions.
- `[PR]` – Title of the pull request.
- `[IR]` – The issue report related to the submission.
- `[CT]` – The type of commit (e.g., feature, bug fix).
- `[OM]` – The original commit message.

The generation prompt used by the default commit generation approach configured in APCE can be seen in Figure 4. Furthermore, as seen in Figure 2c, APCE is already configured to use the approaches by Xue et al. [20] and Zhang et al. [21].

C. Evaluation Module

The evaluation module is created to support researchers in performing studies to evaluate new automated commit generation approaches.

APCE will evaluate the similarity between the original commit message and the LLM-generated message(s) by computing the BLEU, METEOR, and ROUGE-L [20] evaluation metrics for each of the LLM-generated messages, well-known summarization metrics used to assess the quality of computer-generated commit messages [21]. However, there are cases where the original message may hold truth, but have minor overlap with the LLM-generated message, even if the meaning remains the same [22]. Therefore, APCE supports the collection of human evaluation feedback.

[8]https://chat.deepseek.com

Prompt: *Generate a high-quality commit message based on the given information. Your response must contain only the commit message—no explanations, no punctuation, no extra words.*
Requirements: - Must clearly describe what changed and why - Must follow standard commit message formatting - Must use imperative mood - Must not include any punctuation
Given Data:
1. Code changes: [DIFF]
2. Associated pull request title: [PR]
3. Associated issue report: [IR]
4. Commit Type: [CT]
Approach Name: *Default*
Refinement: `true`

Fig. 4. Commit Generation Agent Prompt Example

When the user clicks the ”View AI-Generated Messages” button, a modal pop-up window appears for the user to rate and provide feedback for each of the commit messages generated by the different approaches. The modal shows the user the original message, followed by AI-generated commit messages displayed in a shuffled order with arbitrary indexes, as shown in Figure 2b. In particular, the user is asked to rate each LLM-generated message on a 5-point Likert scale using five quality criteria derived from existing work [2], [20]. The criteria are: *accuracy* (Is the commit message correct?), *integrity* (Does it explain what changed and why?), *readability* (Is the commit message clear and free of grammatical errors?), *applicability* (Would other developers use the same commit message?), and *completeness* (Does the commit message cover all the changes?).

Lastly, for each LLM-generated message, the user is asked to include a rationale for their ratings. Once a user successfully submits the ratings, the submission is stored in the database.

For each submission, APCE stores the following attributes:

- A unique submission ID
- The issue report of the GitHub commit
- The Commit ID
- The Commit Type
- The original message associated with the GitHub commit
- The title of the pull request of the GitHub commit
- The timestamp of the GitHub commit
- The files associated with the GitHub commit:
  - Filename
  - File status (e.g., added, modified, deleted)
  - Number of additions
  - Number of changes
  - Number of deletions
- The ratings associated with the submission:
  - The generated commit message
  - The Name of the approach used

- Whether the prompt generated a successful commit message
- Whether the refinement prompt was used
- User's ratings
- Rationale for the ratings
- Evaluation metric scores (BLEU, METEOR, ROUGE-L)

During the evaluation, the researcher can access stored submissions using the password-protected research view, shown in Figure 2c. The research view shows the submissions, prompts, approaches, and refinement prompt. In the *Prompts* section, the user can add or remove approaches, or adjust the refinement prompt settings of the chosen approach.

IV. Availability

More about APCE can be found in the tool's GitHub repository, which contains (i) setup and configuration instructions, (ii) the source code, and (iii) the architecture description.

V. Limitations and Future Work

One limitation of APCE is that a large number of API calls can significantly slow down the tool. As noted earlier, performance drops as the commit diff gets larger, which worsens the commit message quality. Furthermore, its usability can be hindered for projects with longer commit histories due to GitHub's hourly rate limit for API usage. In the future, we will work on implementing asynchronous processing, caching, and queuing to help reduce the delay and increase performance when multiple users are using the system concurrently.

Future work includes implementing a feature to import commit message datasets, allowing researchers to perform a bulk analysis of data to will speed up the non-human evaluation of approaches and incorporating additional evaluation metrics and reports.

Currently, researchers are the primary intended users and beneficiaries of APCE. However, developers can use the commit generation module and skip the evaluation. Future iterations of APCE aim to separate the commit generation and evaluation modules, making the tool more accessible and useful for developers.

Acknowledgment

Esteban Parra and Polina Iaremchuk were supported in part by Belmont University through the Summer Undergraduate Research Fellowships in the Sciences (SURFS) program.

References

[1] C. Beams, "How to write a git commit message," 2014, accessed: 2025-04-11. [Online]. Available: https://chris.beams.io/posts/git-commit/

[2] L. F. Cortés-Coy, M. Linares-Vásquez, J. Aponte, and D. Poshyvanyk, "On automatically generating commit messages via summarization of source code changes," in *Proceedings of the 14th IEEE International Working Conference on Source Code Analysis and Manipulation(SCAM'14)*, 2014, pp. 275–284.

[3] J. Dong, Y. Lou, Q. Zhu, Z. Sun, Z. Li, W. Zhang, and D. Hao, "Fira: fine-grained graph-based code change representation for automated commit message generation," in *Proceedings of the 44th IEEE/ACM International Conference on Software Engineering (ICSE'22)*, 2022, pp. 970–981.

[4] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen, "Boa: A language and infrastructure for analyzing ultra-large-scale software repositories," in *Proccedings of the 35th IEEE International Conference on Software Engineering (ICSE'13)*. IEEE, 2013, pp. 422–431.

[5] J. He, C. Treude, and D. Lo, "Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 5, May 2025.

[6] J. Li and I. Ahmed, "Commit message matters: Investigating impact and evolution of commit message quality," in *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE'23)*. IEEE, 2023, pp. 806–817.

[7] Q. Liu, Z. Liu, H. Zhu, H. Fan, B. Du, and Y. Qian, "Generating commit messages from diffs using pointer-generator network," in *Proceedings of the 16th IEEE/ACM International Conference on Mining Software Repositories (MSR'19)*. IEEE, 2019, pp. 299–309.

[8] S. Liu, C. Gao, S. Chen, L. Y. Nie, and Y. Liu, "Atom: Commit message generation based on abstract syntax tree and hybrid ranking," *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1800–1817, 2020.

[9] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang, "Neural machine translation-based commit message generation: How far are we?" in *Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE'18)*. ACM, 2018, pp. 373–384.

[10] Z. Liu, Z. Tang, X. Xia, and X. Yang, "CCRep: Learning code change representations via pre-trained code model and query back," in *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE'23)*. IEEE, 2023, pp. 17–29.

[11] S. Mohamed, A. Parvin, and E. Parra, "Chatting with ai: Deciphering developer conversations with chatgpt," in *Proceedings of the 21st IEEE/ACM International Conference on Mining Software Repositories (MSR'24)*. Lisbon, Portugal: ACM, 2024, p. 187–191.

[12] D. Roy, S. Fakhoury, and V. Arnaoudova, "Reassessing automatic evaluation metrics for code summarization tasks," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC-FSE'21)*, 2021, pp. 1105–1116.

[13] J. Sallou, T. Durieux, and A. Panichella, "Breaking the silence: the threats of using llms in software engineering," in *Proceedings of the 44th ACM/IEEE International Conference on Software Engineering: New Ideas and Emerging Results (ICSE'24)*, 2024, pp. 102–106.

[14] M. Shehata, B. Lepore, H. Cummings, and E. Parra, "Creating uml class diagrams with general-purpose llms," in *Proceedings of the 20th IEEE Working Conference on Software Visualization (VISSOFT'24)*, Flgstaff, Arizona, USA, 2024, pp. 157–158.

[15] W. Tao, Y. Wang, E. Shi, L. Du, S. Han, H. Zhang, D. Zhang, and W. Zhang, "A large-scale empirical study of commit message generation: models, datasets and evaluation," *Empirical Software Engineering*, vol. 27, no. 7, p. 198, 2022.

[16] Y. Tian, Y. Zhang, K.-J. Stol, L. Jiang, and H. Liu, "What makes a good commit message?" in *Proceedings of the IEEE/ACM 44th International Conference on Software Engineering (ICSE'22)*. New York, NY, USA: Association for Computing Machinery, 2022, p. 2389–2401.

[17] H. Wang, X. Xia, D. Lo, Q. He, X. Wang, and J. Grundy, "Context-aware retrieval-based deep commit message generation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 4, pp. 1–30, 2021.

[18] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A prompt pattern catalog to enhance prompt engineering with chatgpt," *arXiv preprint arXiv:2302.11382*, 2023.

[19] S. Xu, Y. Yao, F. Xu, T. Gu, H. Tong, and J. Lu, "Commit message generation for source code changes," in *Proceedings of the 18th International Joint Conferences on Artificial Intelligence (IJCAI'19)*, 2019, pp. 3975–3981.

[20] P. Xue, L. Wu, Z. Yu, Z. Jin, Z. Yang, X. Li, Z. Yang, and Y. Tan, "Automated commit message generation with large language models: An empirical study and beyond," *IEEE Transactions on Software Engineering*, vol. 50, no. 12, pp. 3208–3224, 2024.

[21] L. Zhang, J. Zhao, C. Wang, and P. Liang, "Using large language models for commit message generation: A preliminary study," in *Proceedings of the 31st IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER'24)*, 2024, pp. 126–130.

[22] Y. Zhang, Z. Qiu, K.-J. Stol, W. Zhu, J. Zhu, Y. Tian, and H. Liu, "Automatic commit message generation: A critical review and directions for future work," *IEEE Transactions on Software Engineering*, vol. 50, no. 4, pp. 816–835, 2024.