Advances in Artificial Intelligence and Machine Learning; Research 4 (4) 2950-2968 Received 28-09-2024; Accepted 2-12-2024; Published 9-12-2024

Enhancing Commit Message Categorization in Open-Source Repositories Using Structured Taxonomy and Large Language Models

Muna Al-razgan malrazgan@ksu.edu.sa
Department of Software Engineering,
College of Computer and Information Sciences
King Saud University, Riyadh, 11345, Saudi Arabia

Manal Alaqil Malageel@ksu.edu.sa
Department of Software Engineering,
College of Computer and Information Sciences
King Saud University, Riyadh, 11345, Saudi Arabia

Ruba Almuwayshir ruba.almuwayshir@gmail.com
Department of Software Engineering,
College of Computer and Information Sciences
King Saud University, Riyadh, 11345, Saudi Arabia

Zamzam Alhijji zalhijji@seu.edu.sa
Department of Software Engineering,
College of Computer and Information Sciences
King Saud University, Riyadh, 11345, Saudi Arabia

Corresponding Author: Muna Al-razgan

Copyright © 2024 Muna Al-razgan, et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract

Version Control Systems (VCS) manage source code changes by storing modifications in a database. A key feature of VCS is the commit function, which saves the project's current state and summarizes changes through Commit Message (CM). These messages are vital for collaboration, particularly in open-source artificial intelligence (AI) projects on platforms, where contributors work on rapidly evolving codebases. This paper presents an empirical analysis of CM within open-source AI repositories on GitHub, focusing on their content, the effectiveness of categorization by Large Language Models (LLMs), and the impact of message quality on categorization accuracy. A sample of 384 CMs from 34 repositories was manually categorized to establish a taxonomy. Python was then used for automated keyword extraction, refined with regex patterns. Also, an experiment involved assessing the performance of ChatGPT-4 in categorizing CMs, first without guidance and later using our developed taxonomy. Our findings indicate that the quality of CMs varies greatly, which has a clear impact on how efficiently they can be categorized. This study contributes to the field by providing a structured taxonomy of CMs and exploring how tools like ChatGPT-4 can be used to analyze them. The insights from this research are intended to benefit both academic studies and real-world software development, particularly by helping teams better understand and automate the handling of CM in AI projects.

Citation: Muna Al-razgan, et al. Enhancing Commit Message Categorization in Open-Source Repositories Using Structured Taxonomy and Large Language Models. Advances in Artificial Intelligence and Machine Learning. 2024;4(4):171.

Keywords: Categorization, Commit's Quality, Experiment, LLMs, Maintenance.

1. INTRODUCTION

Version Control System (VCS) manages source code changes by recording them in a database. VCS provides benefits such as improved collaboration, easy backups, and efficient change tracking, helping maintain code integrity and support project management.

Commit Messages (CMs) are crucial in VCSs, providing detailed explanations for code changes to enhance understanding and collaboration. A typical CM includes a concise title or summary (up to 50 characters) and a more detailed body that explains the rationale behind the changes, the decision-making process, and any alternatives considered.

In complex software development, especially with GitHub, clear and keyword-rich CMs (e.g., "fix," "add," "update") are essential for maintaining project continuity, enhancing understanding and categorization, and streamlining development and management processes. These keywords in CMs provide clarity about the purpose of changes, helping contributors and maintainers quickly identify whether a commit fixes bugs, adds features, or modifies functionalities. This level of detail helps keep projects organized, supports tools that automate tasks like generating release notes, and ensures that future developers can easily understand past changes, especially in long-running projects.

The rise of open-source Artificial Intelligence (AI) repositories on GitHub has made it clear that improved version control practices are necessary. With many contributors and rapid code changes, clear CMs are more important than ever for tracking development. Furthermore, the advent of Large Language Models (LLMs) makes it easier to automate the analysis and categorization of CMs, helping developers streamline reviews, improve accuracy in tracking changes, and enhance project traceability and documentation.

Our study aims to empirically analyze CMs within open-source AI repositories to uncover standard practices and potential areas for improvement. By focusing on these repositories, we can gain insights into how the unique characteristics of AI software development influence version control strategies and how LLMs could be deployed to support these activities. This analysis contributes to academic understanding and provides practical recommendations for AI developers on GitHub, promoting better documentation practices and more effective project management.

Three principal questions guide our research, each addressing a critical aspect of understanding and enhancing the use of CMs in software development, especially within the context of open-source AI projects. The first research question (RQ1) explores the types of content most commonly described in the CMs across these repositories. This involves analyzing the frequency and patterns of specific terms and phrases, thereby identifying prevalent themes and the level of detail typically provided. The second question (RQ2) assesses the effectiveness of ChatGPT in accurately categorizing these CMs. Here, we aim to understand the capabilities and limitations of LLMs when applied to sorting and classifying complex software documentation under varied conditions across different repositories. Lastly, our third question (RQ3) examines how the quality of CMs influences the accuracy and efficiency of these categorization processes. Poorly written or vague CMs may hinder automated tools' ability to categorize them correctly, impacting overall project manageability and traceability.

By addressing these questions, our study aims to provide a comprehensive overview of the current state of CM's quality and its implications for software development practices, particularly in projects involving AI.

RQ1: *What types of content are most commonly described in the CMs of open-source AI repositories?*

RQ2: *How effectively do LLMs categorize CMs across open-source AI repositories?*

RQ3: *How does the quality of CMs impact the accuracy and efficiency of categorization processes in software development projects?*

To answer these questions, we conducted a multi-method study involving multiple steps: First, we randomly selected 384 CMs to analyze, using statistical sampling to ensure representation. These messages were manually examined for keywords and categorized using an open card sorting method to establish initial categories. Next, we used Python to automate the extraction and categorization of keywords, refining our approach with regex patterns to account for variations in terminology. Finally, we conducted a thorough manual verification of the automated categorization results, refining our process based on the discrepancies found, thus ensuring the accuracy and reliability of our categorization.

We will conduct an experiment in two phases to assess ChatGPT's effectiveness in categorizing CMs. Initially, we will prompt ChatGPT to categorize CMs immediately after uploading the dataset without specific guidelines. In the second phase, we will provide ChatGPT with a predefined taxonomy and set of categories. We will then ask it to categorize the CMs again, using our proposed taxonomy as a framework.

The contributions of this paper are fourfold, centered around the enhancement of understanding and processing CMs within open-source AI repositories. First, we have compiled a comprehensive dataset of categorized CMs from 34 open-source AI repositories, which provides a diverse basis for analysis. Second, we have developed and proposed a taxonomy of CM categories, which serves as a structured framework for evaluating and categorizing CMs more effectively. Third, we assess the performance of ChatGPT-4, an advanced language model, in the categorization process based on our taxonomy, offering insights into the feasibility and accuracy of using automated systems for this purpose. Lastly, this research provides valuable benchmarks for future studies and practical applications in automating the classification and analysis of CMs, thereby contributing to academic research and valuable improvements in software development practices.

In the remainder of this paper, we review related work in Section (2), outline our multi-method research approach in Section (3), and present the results of our study in Section (4). We present threats to the validity of our reported findings in Section (5) and conclude the paper in Section (6).

2. RELATED WORK

2.1 GitHub Commits

Commits are key records in VCS that track modifications in software repositories, providing insights into project evolution [1]. Researchers can analyze commit histories to understand software changes and the motivations behind them. Several studies have focused on automating the classification of software maintenance activities within commits, using data such as CMs, code changes, and metadata [2]. By applying supervised learning models trained on annotated data, these approaches aim to improve the efficiency of identifying and classifying maintenance tasks, thereby enhancing software maintenance processes.

The study [3], aimed to classify CMs related to maintenance activities using aggregated semantic word embeddings. The authors collected and preprocessed 1,132 CMs, removing stop words, punctuation, and special characters. They applied the Word2Vec model to generate word embeddings and aggregated them to represent each CM. Using the DBSCAN clustering algorithm, they grouped the messages into maintenance activities. When tested on a set of 200 unseen CMs, the approach achieved an accuracy of 85.5%, demonstrating its effectiveness in classifying CMs into maintenance tasks.

Another study investigated the use of source code density to improve the accuracy of automatic commit classification in maintenance activities [4]. The approach involved measuring the number of lines of code per unit of code to categorize CMs. The process included data collection, preprocessing (removal of stop words, punctuation, and special characters), and classification using machine learning such as a support vector machine (SVM) algorithm. The study compared this method to a bag-of-words approach and found that the source code density-based method achieved higher accuracy (96.4%), indicating its effectiveness in capturing the semantic meaning of CMs and improving classification accuracy.

In another study [5], the authors combined feature extraction and supervised learning techniques to classify significant code changes into maintenance categories. They used features such as commit author, date, time, and file details from commit metadata to train a SVM classifier on 1,132 CMs. The model was tested on 200 messages, achieving an accuracy of 88.5%. In comparison, the k-nearest neighbors algorithm resulted in lower accuracy, highlighting the effectiveness of the SVM classifier for this task.

2.2 Machine Learning Classification Techniques

Manual labeling requires a lot of effort [6], so researchers have turned to machine learning techniques like classification and clustering to help automate the process of categorizing data. While most studies have focused on GitHub issues, fewer have investigated classifying commits. This research highlights recent work on labeling GitHub commits to support software maintenance engineers.

Two studies explored classifying security-relevant commits. One used co-training to categorize commits as ’positive’ (security-related) or ’negative’ (non-security) based on message content and

code changes [7]. The other simplified commit classification by converting it into a document classification task using natural language terms, reducing the need for extensive training data [8].

A study [6], introduced a framework for commit classification using contrastive learning to generate meaningful representations by contrasting positive and negative data pairs. By employing sentence transformers and self-supervised methods to compare embedding vectors, the model's effectiveness was evaluated through experiments on two public datasets.

This study [9], introduced a T5-based generative framework for commit classification, which simplifies the model and enables prompt tuning for few-shot scenarios, showing its effectiveness through extensive experiments.

This research [10], introduced CommitBART, a pre-trained encoder-decoder transformer model specifically designed for GitHub commits. It utilizes pre-training across six tasks, including denoising, cross-modal generation, and contrastive learning, to capture commit fragment representations. Extensive experiments demonstrated that CommitBART outperformed previous models, leveraging BART's structure and PLBART parameters for faster training, and was fine-tuned to both understand and generate tasks.

Also, the authors of [11], fine-tuned DistilBERT to classify CMs, addressing the challenge of distinct programming language keywords, and validated their approach using a commit labeling dataset on industrial GitHub projects, including PyTorch. The authors of [12], enhanced the state-of-the-art commit classification model by incorporating features based on code changes, replacing the BGM algorithm with XGBoost, and evaluating the model to identify the most effective features for supporting maintenance activities and project management.

3. STUDY DESIGN

Following a previous study [13], this study adopts a systematic methodology to analyze the CMs within open-source software (OSS) repositories specializing in AI hosted on GitHub. The scope of this research is defined by the significant role GitHub plays as a central platform for AI development, leveraging its widespread use and the accessibility of its API for data extraction. Our methodology encompasses several key phases: data collection, taxonomy development, CM analysis, and validation of the findings. Additionally, an experiment was conducted to assess the performance of LLMs in the categorization process.

The initial phase of our methodology involves collecting CMs from open-source AI repositories on GitHub. This process builds upon established methods from prior studies that examined issues in OSS AI projects, refining techniques to capture a comprehensive dataset of CMs that reflect the day-to-day activities and development patterns within these projects.

Following data collection, we develop a taxonomy of commit types. This taxonomy is essential for classifying the CMs. This taxonomy's development is informed by existing literature and iterative analysis of the collected data, ensuring that it accurately reflects the nuances of AI software development.

In this study, we also investigated the efficacy of LLMs in categorizing CMs from open-source AI repositories on GitHub. Specifically, we selected ChatGPT-4 for our experiments due to its ability to process uploaded files. ChatGPT-4's advanced capabilities make it an ideal candidate for complex text analysis tasks, which are essential for our study's objectives. The categorization experiments were conducted in two phases: Categorization Without Taxonomy: In this first experiment, we provided ChatGPT-4 with the dataset and a generic prompt to categorize the CMs based on its understanding. Categorization With Taxonomy: In the second experiment, we provided the same dataset along with our detailed taxonomy and categories. Then, we asked ChatGPT-4 to categorize the CMs according to these categories. These experiments were designed not only to gauge the performance of ChatGPT-4 in a controlled environment but also to explore the potential of LLMs to enhance the management and categorization of software development documentation in AI-driven projects.

Finally, we evaluated ChatGPT's performance in categorizing CMs in the two phases without taxonomy and with taxonomy. We also used Cohen's Kappa score to measure the agreement between the results of ChatGPT and manual categorization.

3.1 Scope

We use the same set of 576 open-source AI repositories identified in the original study [13], as the starting point for our data collection. Due to the vast number of repositories and limited time, we had to refine our selection criteria. We chose specific columns from the original repository dataset to filter our repository collection effectively. Here is a list of these columns:

- **Framework:** The main framework used in the repository (e.g., PyTorch, TensorFlow).
- **Year_Paper: The year the paper was published.**
- **Conference_Paper: The conference where the paper was presented.**
- **Owners: The type of owner (e.g., academia, industry).**
- **Contributors: Number of contributors to the repository.**
- **Num_Star:** The number of stars the repository has on GitHub.

Our study includes the five prestigious AI conferences: NeurIPS, ICML, CVPR, ICCV, and ECCV. All the selected conferences are categorized as A* in the CORE Ranking [14]. The dates of these conferences are between 1/1/2020 and 31/12/2022. The repositories used in the experiment are labeled as official on the PapersWithCode platform [15], which inherently signifies a recognized standard of credibility and is owned by academia. Specifically, we have applied a filter to select only those repositories that garnered over 100 stars, a metric indicating a threshold level of community endorsement. We have chosen repositories with multiple contributors to explore different CM styles and collaborative practices. Repositories with active discussions, pull requests, and issue tracking may offer deeper insights into the development process and how CMs reflect this. Our experimental framework leans heavily on PyTorch [16], which has gained significant traction within the research community, especially for its user-friendly interface and flexibility in designing complex models.

This popularity means that it is more likely that the repositories we are examining from prestigious AI conferences utilize PyTorch, ensuring our study's relevance and applicability to current research practices. After applying our selection criteria, our dataset comprises 34 repositories.

3.2 Commit Message Collection and Preprocessing

Since CMs are not directly included in the original dataset [13], we employ the GitHub REST API and use the *URL_Repo* (the URL of the GitHub repository) column from our repositories dataset to access each repository and fetch CMs and commit information for the collected repositories. For each commit, we extracted the following information:

Repository: The name of the repository to which the CM belongs.

Commit Hash: A unique SHA-1 hash used to identify each commit.

Author Email: The email of the individual who made the commit.

Author Name: The name of the individual who made the commit.

Date: The date and time when the commit was made.

Commit Message: The message associated with the commit.

URL: The URL to view the commit on the GitHub website.

We used Windows PowerShell to make requests to the GitHub API to fetch information about CMs after obtaining a personal access token from GitHub. The GitHub API has rate limits, and we are accessing many repositories with many commits, so we needed to handle pagination to fetch more commits. After collecting all the CMs from the 34 repositories, we cleaned and preprocessed them to ensure they were ready for analysis. In the data collection, we only considered commits in which the messages are written in English. Our final dataset consists of 34 pairs of repositories and their corresponding papers. Additionally, we find that 3823 CMs belong to these repositories.

3.3 Development of Taxonomy for Commit Messages

We obtained 3823 CMs from the 34 repositories using the GitHub REST API. The original study [13], used the card sorting method to create mental models and derive taxonomies from data to categorize software engineering-related discussions. As the CMs in open-source AI repositories can be unique, we adopt open card sorting, which uses no predefined groups to create the taxonomy for CMs in AI repositories. The taxonomy is intended to categorize the primary content and types of CMs. In our study, we followed four steps to create a taxonomy and categorize the CMs in the open-source AI repositories. These steps are described as follows:

1) **Preparation:** We randomly picked 384 CMs from all the collected messages as a sample using a formula considering a 95% confidence level and 5% margin of error. Then, we import these

CMs into an Excel sheet and manually create a card for each possible keyword by examining our dataset's sample of CMs. This gives us an initial sense of the language and terminology commonly used.

2) **Execution:** We discussed categorizing the cards into meaningful groups in meetings. We adopted the open card sorting method, which lets the groups emerge and evolve during the sorting process. In this step, we generated the initial categories and their related keywords and categorized the sample set.

3) **Coding:** For more accuracy, we write Python code to extract keyword counts to verify if our initial set of keywords was correct. After that, we iterated and expanded as we categorized more CMs; we discovered additional keywords and realized that some keywords were too broad or irrelevant, so we adjusted our list of keywords accordingly. We also used Python to automate the categorization of the entire dataset of CMs based on our identified keywords and categories. In the categorization code, we refined the keywords with Regular Expressions (Regex Patterns) to capture variations of our keywords (e.g., "fix," "fixes," "fixed") to ensure we aren't missing relevant messages due to slight differences in wording. Finally, we reviewed the categorization results to validate the effectiveness of our keywords and categories and made further refinements.

4) **Manual Verification:** We manually verified the entire dataset of the categorized messages by each author independently to ensure accuracy and refine the patterns as necessary.

3.4 Using LLMs For Commit Message Categorization

LLMs have garnered significant attention and adoption in both academic and industrial domains, including software engineering, due to their exceptional performance across a wide range of applications [17]. ChatGPT's [18], ability to answer questions conversationally allows it to categorize CMs by answering related prompts. ChatGPT is a general-purpose LLM from OpenAI. It has many network parameters and has shown excellent performance in tasks such as solving programming bugs [19], and acting as a translator [20]. In our experiment, we selected ChatGPT- 4 since it allows uploading files, offers an open API interface, and has several advantages that make it a powerful tool for many applications.

Our experimental design consisted of two distinct phases to assess the model's categorization skills under different conditions:

Categorization Without Taxonomy: Initially, we conducted an experiment where ChatGPT-4 was provided with the dataset and a generic prompt to categorize CMs based solely on its inherent understanding and algorithms. This phase evaluated the model's baseline ability to interpret and classify text without external guidance.

Categorization With Taxonomy: In the second phase of the experiment, we supplied ChatGPT -4 with the same dataset, this time accompanied by a detailed taxonomy and predefined categories names and keywords developed during the initial dataset analysis. The objective was to direct the model's categorization process using specific criteria, allowing us to assess how well ChatGPT-4 could adapt its responses to fit a structured classification framework.

For the first experiment, we uploaded our entire dataset. Following the instructions of White et al. [21], we designed the following prompt: *"This is a dataset of commit messages from open-source AI repositories extracted from GitHub. Your task is to analyze each commit message and categorize it into relevant groups or topics based on its content. Please provide the most appropriate category for each commit message, ensuring accuracy and consistency in classification. Additionally, any terminology or patterns common in AI development that may influence categorization should be considered. Thank you!"*. ChatGPT first analyzed and categorized each CM based on its content to identify the predominant themes or topics. It initiated this process by cleaning the text of each CM, removing any non-alphabetical characters, and converting the text to lowercase to standardize the data for analysis. Using a 'CountVectorizer, ' ChatGPT extracted the most frequent keywords from the cleaned CMs. This analysis revealed common keywords such as "update," "fix," "add," "pull," and "bug," which are typical in software development contexts. Based on these keywords, it defined five potential categories for the CMs: Documentation Updates, Feature Additions, Bug Fixes, Code Reviews and Revisions, and Integration Activities. TABLE 1 shows a list of some of the keywords. ChatGPT then developed a function to categorize each CM into one of these five categories based on the presence and frequency of the identified keywords. Messages that did not clearly fit into these categories were labeled "Other."

Table 1: ChatGPT List of Keywords

Keyword	***Count***	***Keyword***	***Count***	***Keyword***	***Count***
Update	1651	Request	535	reviewed	771
Revision	963	Ppwwyyxx	460	fix	607
differential	960	Resolved	384	add	596
fbshipitsourceid	953	Pulled	361	pull	543
readmemd	908	Merge	264		
Summary	816	Rbgirshick	259		

For the second experiment, we uploaded the entire dataset of the CMs and a Word document with our taxonomy explaining each category with its related keywords. Following the instructions of White et al. [21], we designed the following prompt: *"Using the attached taxonomy and category definitions, please categorize the dataset of the commit messages uploaded. Each message should be assigned to the most appropriate category based on content. For each commit message, provide the category that best fits the description of the change made. Your analysis should consider the context and details provided in each commit message to accurately align it with the defined categories. Please give me the result as a CSV file."* After analyzing the files, ChatGPT provided us with a downloadable CSV file.

3.5 Evaluation Metrics

Several tools and libraries are available across different programming environments for automated calculation of evaluation metrics, especially in the context of machine learning and data analysis. Scikit-learn is one of the most popular machine-learning libraries in Python. It provides a straightforward way to calculate different evaluation metrics.

We calculated accuracy, recall, and the F1 score based on two columns from different CSV datasets in Python since we have one dataset containing true labels and another containing predicted labels, and we want to evaluate different aspects of ChatGPT4 's performance of predictions. These metrics can provide a more detailed understanding of a model's performance, especially in scenarios where we have imbalanced classes. First, we installed Pandas for data manipulation and Scikit-learn for calculating accuracy, and then we imported the necessary libraries into the Python script. Then, the script will load the two CSV files, which should have the same number of rows and be aligned correctly. Finally, we extracted the relevant columns to compute the evaluation metrics.

Additionally, in our study, we utilized Cohen's Kappa score to measure the agreement between our manual categorization of CMs and the categorizations produced by ChatGPT-4 in two different phases: one where a taxonomy was employed and one without the use of taxonomy. Cohen's Kappa score is a statistical measure widely used to assess the degree of agreement between two raters beyond what would be expected by chance alone. It is particularly valuable when decisions are subjective, as it accounts for the possibility that agreements might occur randomly.

4. RESULTS

This section presents our analysis of the CMs in OSS AI repositories. To answer our research questions, we conducted an empirical study to systematically investigate the CMs in open-source AI repositories hosted on GitHub.

4.1 A Taxonomy of Commit Messages

This section will answer RQ1. *What types of content are most commonly described in the CMs of open-source AI repositories?*

In open-source AI repositories, CMs commonly describe several key types of content that are essential for maintaining the clarity and context of code changes. CMs tend to focus on various categories, each highlighting different aspects of code development and maintenance. Analyzing the types of content commonly described in the CMs of open-source AI repositories is essential for improving the quality and efficiency of software development. By categorizing these CMs, we can enhance communication among developers, streamline project management, and support the creation of automated tools for code review and integration. Our analysis revealed nine distinct categories, each critical to the development process. TABLE 2 below summarizes our results about CM categories and their associated keywords.

Documentation Updates: This category, making up 31.57% of CMs, involves changes related to project documentation. These updates ensure that user guides, README files, and other documentation are accurate and comprehensive, which is crucial for both users and developers to understand the project's functionality and usage.

Bug Fixes: This category comprises 20.72% of CMs and addresses the correction of errors and issues within the code.

Table 2: The distribution of CMs of different categories

Category	***Keywords***	***No***	***Rate***
Documentation Updates	“documentation”, “readme”, “wiki”.	1207	31.57%
Bug Fixes	“fix”, “bug”, “error”, “issue”, “correct”, “resolve”, “fixes”, “debug”, “address”, “remediation”,” typo”.	792	20.72%
Feature Improvements	“update+feature”, “edit”, “feature”, “adjust”, modify”, “improve”, “allow”,” declared”,” merge”.	551	14.41%
Other		535	13.99%
Feature Additions	“add,” “add+feature,” “implement,” “introduce,” “new,” “create,”” add +script.”	464	12.14%
Build System	“build”, “CI”, “pipeline”, “automation”, “setup”, “recompile”, “configs”, “configuration”.	123	3.22%
Testing	“test”, “unit test”, “integration test”.	79	2.07%
Dependency Updates	“upgrade”, “dependency”, “library”, “package”, “version”, “modify” “API”, “links”,” link”,” delete+ code file”, “rename+ code file”,” create + code file”.	41	1.07%
Code Refactoring	“refactor”, “restructure”, “clean up”, “optimize”, “recursion”	31	0.81%

Feature Improvements: Representing 14.41% of CMs, feature improvements refer to the enhancement of existing functionalities. These commits aim to optimize the software, improving performance, usability, and user experience.

Feature Additions: This category comprises 12.14% of CMs and introduces new features or functionalities. These additions expand the software’s capabilities, allowing it to meet or enhance new requirements.

Build System: This category includes changes related to the build system and continuous integration setup, accounting for 3.22% of CMs. These updates are crucial for maintaining an efficient development pipeline, ensuring the software can be built and tested automatically and consistently.

Testing: This category represents 2.07% of CMs and pertains to changes related to testing the software. This includes adding or updating unit tests, integration tests, and other testing mechanisms.

Dependency Updates: This category comprises 1.07% of CMs and involves updating external libraries or dependencies. These updates ensure that the software remains compatible with the latest versions of libraries and frameworks and benefits from new features, performance improvements, and security patches.

Code Refactoring: This category makes up 0.81% of CMs and includes changes to improve the code structure without altering its functionality. Refactoring helps maintain a clean, efficient, and manageable codebase that is easier to understand and extend in the future.

Other: This category accounts for 13.99% of CMs and includes various changes that do not fit into the other specified categories.

This diverse distribution of CMs underscores the multifaceted nature of software development in AI repositories, balancing between maintenance, improvement, and expansion. Understanding these categories allows developers to adopt better CM practices, leading to improved documentation, easier tracking of changes, and enhanced collaboration within the development team. This structured approach ultimately contributes to software development's overall efficiency and effectiveness in AI projects.

In addition, to generalize our results and allow them to be matched and evaluated by previous works, we can link our categories with the umbrella of commit classifications for maintenance tasks as follows:

- **Adaptive:** This includes Documentation updates, Feature additions, and Build System commits.
- **Corrective:** This encompasses Bug Fixes and Code Refactoring.
- **Perfective:** This category includes Feature Improvements, Testing, and Dependency Updates.

In our current taxonomy, each CM is assigned to a single category to simplify classification. We recognize, however, that CMs may fit multiple categories. In future work, we plan to enhance the taxonomy by allowing overlapping categories, increasing flexibility in classifying CMs for maintenance tasks.

We align our findings with established frameworks by categorizing CMs under these broader maintenance task classifications, facilitating comparison and evaluation with prior research.

4.2 Performance Of Chatgpt-4 For Commit Message Categorization

This section will answer RQ2: *How effectively do LLMs categorize CMs across open-source AI repositories?* LLMs, such as OpenAI's GPT-4, are AI systems trained on vast datasets to understand and generate human-like text. They leverage deep learning techniques, specifically transformer architectures, to process and produce natural language. These models are designed to perform various tasks, including text completion, translation, summarization, and question-answering. LLMs' capabilities stem from their ability to capture complex patterns and contextual nuances in language, making them powerful tools for applications requiring sophisticated text processing and generation. LLMs have diverse applications, including text classification and categorization, content generation, and powering conversational agents. They categorize and classify text data by understanding context and identifying patterns, which is useful in software development, customer feedback analysis, and content moderation.

In our study, we experimented with the efficiency of ChatGPT-4 in categorizing and classifying CMs by conducting two separate experiments. In the first experiment, we used ChatGPT-4 without providing our proposed taxonomy, allowing the model to classify the CMs based on its inherent training. In the second experiment, we provided ChatGPT-4 with our detailed taxonomy to guide its classification process. We aimed to evaluate the performance of ChatGPT-4 in both scenarios and then compare the results. To do this, we measured the precision, recall, and F1 score of ChatGPT-4's classifications in both experiments and compared these metrics with the results from our manual

categorization. This comparison allowed us to assess the impact of providing a structured taxonomy on the model's classification accuracy and efficiency.

The following TABLE 3, and TABLE 4, illustrate the distribution of CMs into various categories, with and without using a predefined taxonomy. The categorization of CMs in TABLE 3, is divided into several distinct types based on the activities they represent. "Documentation Updates" involves commits related to modifying project documentation, with keywords like "update" or references to specific files such as "README.md". "Feature Additions" pertains to commits adding new functionalities or enhancements, indicated by keywords like "add" or "support". "Bug Fixes" includes commits that resolve software issues, using keywords such as "fix" or "bug". "Code Reviews and Revisions" covers commits stemming from the peer review process, highlighted by terms like "reviewed", "revision", or "resolved", which reflect efforts to enhance code quality. "Integration Activities" relates to commits involving the integration of different software parts or updates from various branches, with keywords such as "merge", "pull", or "request". The "Other" category encapsulates all commits that do not fit into the aforementioned categories, including a variety of activities outside the typical development scope. Both the taxonomy-based and non-taxonomy-based categorizations highlight the importance of Documentation Updates and Bug Fixes, with Documentation Updates being the largest category in both, though at different rates (25.45% with taxonomy vs. 37.1% without). Feature Additions also appear prominently in both, indicating ongoing development activities. However, the taxonomy-based categorization includes specific categories like Feature Improvements, Build System, Testing, Dependency Updates, and Code Refactoring, which are not explicitly identified in the non-taxonomy-based results. Instead, the non-taxonomy-based approach groups broader activities under categories like Code Reviews and Revisions (18.9%) and Integration Activities (11.2%). Both approaches have a miscellaneous category "Other" capturing diverse commits, with 23.78% in the taxonomy-based and 12.4% in the non-taxonomy-based results.

Table 3: ChatGPT Categories (Without Taxonomy)

Category	Keywords	No	Rate
Documentation Updates	(e.g., keywords like "update", "readmemd")	1418	37.1%
Feature Additions	(e.g., keywords like "add", "support")	447	11.7%
Bug Fixes	(e.g., keywords like "fix", "bug")	331	8.7%
Code Reviews and Revisions	(e.g., keywords like "reviewed", "revision", "resolved")	724	18.9%
Integration Activities	(e.g., keywords like "merge", "pull", "request")	429	11.2%
Other	*Messages that did not fit into the other categories	474	12.4%

The two phases of the experiment in TABLE 5, one without taxonomy and one with taxonomy, help understand how external guidance in the form of a predefined taxonomy enhances the model's categorization capabilities.

In Phase 1, the model operated without specific guidance beyond its pre-trained capabilities and the input data. The accuracy (50.95%) is relatively low, hovering just above 50%. This suggests that,

Table 4: ChatGPT Categories (With Taxonomy)

Category	***No***	***Rate***	***Category***	***No***	***Rate***
Documentation Updates	973	25.5%	Build System	63	1.56%
Bug Fixes	942	24.6%	Testing	42	1.10%
Feature Improvements	231	6.04%	Dependency Updates	28	0.73%
Other	909	23.8%	Code Refactoring	21	0.55%
Feature Additions	614	16.06%			

Table 5: Performance Of ChatGPT in Two Phases

Metrics	Phase 1: ChatGPT without Taxonomy	Phase 1: ChatGPT with Taxonomy
Accuracy	50.95%	66.83%
Precision	60.05%	76.08%
Recall	50.95%	66.83%
F1 Score	52.95%	69.04%

while the model can categorize commit messages to some degree, its performance is only slightly better than random guessing in this context. Nevertheless, since we have imbalanced datasets, accuracy might not be a good metric as it can be skewed by the majority class. In such cases, precision, recall, and the F1 score are more informative for evaluating the effectiveness of ChatGPT-4 in categorizing CMs in open-source AI repositories. The precision (60.05%) is somewhat higher, indicating that when the model does predict a category, it does so with moderate accuracy. However, there is still a significant amount of false positives. However, the recall (50.95%) indicates that the model only correctly identifies about 51% of relevant instances per category. This implies that nearly half of the actual categorizable commits are being missed. Also, the F1 score (52.95%), which balances precision and recall, is relatively low, reflecting the model's challenges in effectively categorizing CMs without a guiding taxonomy. This score highlights the difficulty in achieving both good precision and recall, particularly when the model operates without specific category guidelines.

In Phase 2, with the introduction of a detailed taxonomy and predefined categories, all metrics improved significantly. The model not only categorizes more accurately (66.83%) but also does so with greater confidence, as evidenced by the higher precision (76.08%). The increase in recall (66.83%) indicates that more true positives are correctly identified against the total actual positives, showing that the model effectively utilizes the provided taxonomy to make more informed categorization decisions. The F1 score (69.04%), which is substantially higher than in Phase1, reflects a more balanced and effective categorization process.

Comparing the two phases clearly shows the impact of providing a taxonomy on the categorization performance of ChatGPT-4. The significant improvements in accuracy, recall, and the F1 score in Phase 2 highlight the importance of structured, domain-specific input in enhancing the model's effectiveness, particularly in scenarios involving complex or specialized data, such as commit messages in open-source AI repositories. In addition, TABLE 6 compares our results with previous studies, showing that ChatGPT's accuracy significantly improved from 50.95% without taxonomy to 66.83% with it. However, both phases still fall short of traditional machine learning methods,

with SVMs achieving accuracies of 96.4% and 88.5%, and the DBSCAN method reaching 85.5%. These findings underscore the need for further advancements in LLM methodologies, particularly through prompt engineering, to enhance accuracy in CM classification.

Table 6: Comparative result of CMs classification models

Method/Study	Accuracy
ChatGPT (phase1: without taxonomy)	50.95%
ChatGPT (phase2: with taxonomy)	66.83%
SVM (code density) [4]	96.40%
SVM (Feature extraction) [5]	88.50%
Word Embeddings (DBSCAN) [3]	85.50%

In our application, Cohen's Kappa provided a robust framework for quantitatively evaluating the consistency of CM categorizations between manually labeled data and those produced by AI-assisted methods. The results indicated a substantial agreement when taxonomy was used (Kappa ≈ 0.605), which underscores the effectiveness of structured categorization aids. Conversely, the agreement was moderate (Kappa ≈ 0.412) without the taxonomy, highlighting the challenges and potential inconsistencies when categorization lacks systematic guidelines. These findings emphasize the importance of employing systematic methods like taxonomies in AI-assisted text categorization to enhance reliability and agreement with human judgment. The data reveals that while ChatGPT-4 has a baseline capability to categorize text, its performance substantially benefits from the addition of clear, contextual guidelines. This insight is particularly valuable for LLM applications in specialized fields where precision and reliable categorization are critical.

4.3 Impact Of the Quality of The Commit Messages on The Categorization Process

This section will answer RQ3. *How does the quality of CMs impact the accuracy and efficiency of categorization processes in software development projects?*

CMs are essential for software maintenance and evolution, providing crucial documentation that helps developers understand code changes and their rationale. High-quality CMs improve communication, streamline code reviews, aid in documentation, enhance project management, and facilitate debugging and maintenance, especially in collaborative projects. Poorly written messages, on the other hand, can lead to inefficiencies and inaccuracies in categorizing changes, negatively impacting project outcomes.

Our dataset analysis revealed that 13.99% of CMs, particularly those categorized as "Other," are low-quality. These messages suffer from ambiguity and lack clarity, making them difficult to categorize accurately. Due to their generic nature or insufficient detail, they might not fit neatly into predefined categories.

According to Yingchen Tian et al. [22], a good CM should summarize the changes (noted as 'What') and describe the reasons for the changes (noted as 'Why'). This dual information is crucial for clarity and understanding. We adopted the same classification framework as the study [22], to classify CMs in our dataset. CMs in the "Other" category have been classified manually based on the presence of

"Why" and "What" information. In this study, we highlighted the 'Other' category only to show how low-quality commits impact categorization accuracy. While initial manual analysis was necessary, examining all categories will improve long-term efficiency and offer broader insights into the effects of low-quality CMs

The results in TABLE 7, showed a high prevalence of low-quality messages in the "Other" category, which is consistent with the paper's observations. These messages often lack sufficient detail, falling into the "Neither Why nor What" classification with a rate of 97.39 %, meaning they fail to provide the reason for the change and a summary of what was changed. In our analysis of CMs, we also aligned our categorization of low-quality messages with those identified in the study [22], by Yingchen Tian et al. The study outlined several categories of low-quality CMs, and we found similar patterns in our dataset that contain the "Other "category with the "neither why nor what" classification.

Table 7: Classification Of "Other" Category

Classification	***Count***	***Rate***
Neither Why nor What	521	97.39%
No What	4	0.75%
No Why	10	1.87%
Why and What	0	0

Low-quality CMs include *Single-word, Submit-centered, Scope-centered, Redundant, and Irrelevant messages.* Upon applying these criteria, we found that some of the CMs from our dataset fit into these subcategories. The results are shown in TABLE 8. These categories highlight the issues in CMs, such as lack of detail, scope, and relevance, which can hinder effective categorization and understanding of the changes made in the codebase. The "Other" category in our dataset predominantly contains CMs that lack both the "Why" and "What" elements. This prevalence highlights the challenges of categorization due to insufficient detail. This aligns with the paper's observation that such messages are problematic for automated tools and manual reviewers, as they do not offer the necessary information to understand the commit. This also highlights the need to improve the quality of these CMs by including both key elements, "Why" and "What," thereby enhancing their clarity and usefulness for understanding the changes made.

Table 8: Categories of Low-Quality CMs

Category	***Count***	***Rate***
Single-word messages	45	8.64%
Submit-centered messages	39	7.49%
Scope-centered messages	67	12.86%

To improve the quality of CMs and thereby enhance the accuracy and efficiency of categorization, we recommend adopting a combination of strategies. First, developers should follow standardized guidelines for writing CMs, such as those outlined in Chris Beams' CM quality guideline [23]. Second, the use of automated linting tools, such as Commitlint [24], can enforce CM standards and reject messages that do not meet specified criteria. Third, integrating automated quality checks can

further improve CM clarity and detail. A full-fledged CM quality checker based on machine learning can rigorously assess CM's quality according to established guidelines like those of Chris Beams [25]. Finally, implementing CM templates can prompt developers to include specific details, such as the type of change, references to issues or tasks, and a summary of the changes. By adopting these strategies, teams can significantly enhance the quality of their CMs, leading to better categorization, improved code maintainability, and more efficient software development processes.

5. THREATS TO VALIDITY

Threats to Internal Validity: The study may face selection bias due to focusing on specific popular repositories, possibly excluding others with different commit patterns. The taxonomy for classifying CMs is subjective, and variability in prompts for ChatGPT-4 could affect the accuracy of categorization.

Threats to External Validity: The findings may not be generalizable to the broader AI development landscape due to the selection of high-profile repositories and reliance on PyTorch. The study's temporal focus on projects active between 2020 and 2022 might not capture recent advancements.

Threats to Construct Validity: The assumption of consistent terminology across projects may lead to misclassification, and the limited dataset size could impact the robustness of conclusions. Additional validation methods could further strengthen the reliability of the results.

6. CONCLUSION AND RECOMMENDATIONS

This study investigates the role of CMs in open-source AI repositories on GitHub, highlighting their importance in maintaining code integrity and collaboration. It categorizes CMs using a systematic methodology, finding that documentation updates are the most common, followed by bug fixes and feature improvements. The study also assesses the effectiveness of ChatGPT-4 in categorizing CMs when provided with a taxonomy. Recommendations include broadening repository selection, involving multiple experts in taxonomy development, and standardizing prompts. Future work may involve validating findings through surveys and exploring automated classification using machine learning.

References

[1] Heričko T, Šumak B. Commit Classification Into Software Maintenance Activities: A Systematic Literature Review. In 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC). Torino Italy: IEEE PUBLICATIONS. 2023:1646-1651

[2] T. Heričko. Automatic Data-Driven Software Change Identification via Code Representation Learning. Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering. ACM. 2023:319–323

[3] Heričko T, Brdnik S, Šumak B. Commit Classification Into Maintenance Activities Using Aggregated Semantic Word Embeddings of Software Change Messages. SQAMIA 2022: Workshop on Software Quality, Analysis, Monitoring, Improvement, and Application. 2022. Available at: https://ceur-ws.org/Vol-3237/paper-her.pdf

[4] Hönel S, Ericsson M, Löwe W, Wingkvist A. Using Source Code Density to Improve the Accuracy of Automatic Commit Classification Into Maintenance Activities. J Syst Softw. 2020;168:110673.

[5] Hindle A, German DM, Godfrey MW, Holt RC. Automatic Classication of Large Changes Into Maintenance Categories. In the 17th International Conference on Program Comprehension. Vancouver BC Canada: IEEE Publications. 2009:30-39.

[6] Tong J, Wang Z, Rui X. Boosting Commit Classification With Contrastive Learning. 2024. ArXiv preprint: https://arxiv.org/pdf/2308.08263.

[7] Lee JY, Chieu HL. Co-training for Commit Classification. In: Proceedings of the Seventh Workshop on Noisy User-Generated Text W-NUT Online. Association for Computational Linguistics. 2021:389-395.

[8] Sarwar MU, Zafar S, Mkaouer MW, Walia GS, Malik MZ. Multi-Label Classification of Commit Messages Using Transfer Learning. IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). Coimbra Portugal. IEEE PUBLICATIONS. 2020:37-42.

[9] Tong J, Rui X. Incorporating Prompt Tuning for Commit Classification With Prior Knowledge. 2023. ArXiv preprint: https://arxiv.org/pdf/2308.10576

[10] Liu S, Li Y, Xie X, Liu Y. CommitBart: A Large Pretrained Model for Github Commits. 2023. ArXiv preprint: https://arxiv.org/pdf/2208.08100

[11] Sarwar MU, Zafar S, Mkaouer MW, Walia GS, Malik MZ. Multi-Label Classification of Commit Messages Using Transfer Learning. IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). Coimbra Portugal. IEEE PUBLICATIONS. 2020:37-42.

[12] Mariano R, Santos G, Brandão W. Improve Classification of Commits Maintenance Activities With Quantitative Changes in Source Code. In: Proceedings of the 23rd international conference on enterprise information systems. SCITEPRESS— Science and Technology Publications. 2021;2:19-29

[13] Yang Z, Wang C, Shi J, Hoang T, Kochhar P, et al. What Do Users Ask In Open-Source AI Repositories? An Empirical Study of Github Issues. 2023. ArXiv Preprint: https://arxiv.org/pdf/2303.09795

[14] Yang Z, Wang C, Shi J, Hoang T, Kochhar P, et al. What Do Users Ask In Open-Source AI Repositories? An Empirical Study of Github Issues. 2023. ArXiv Preprint: https://arxiv.org/pdf/2303.09795

[15] https://paperswithcode.com/

[16] https://paperswithcode.com/

[17] Ozkaya I. Application of Large Language Models to Software Engineering Tasks: Opportunities Risks and Implications. IEEE Softw. 2023;40:4-8.

[18] https://chatgpt.com

[19] Surameery NM, Shakor MY. Use Chat GPT to Solve Programming Bugs. Int J Inf Technol Comput Eng. 2023;31:17-22

[20] Jiao W, Wang W, Huang J, Wang X, Shi S, et. al. Is ChatGPT a Good Translator? Yes With GPT-4 as the Engine.2023. ArXiv preprint: https://arxiv.org/pdf/2301.08745

[21] White J, Fu Q, Hays S, Sandborn M, Olea C, et al. A Prompt Pattern Catalog to Enhance Prompt Engineering With ChatGPT. 2023. ArXiv Preprint: https://arxiv.org/pdf/2302.11382

[22] Tian Y, Zhang Y, Stol KJ, Jiang L, Liu H. What Makes a Good Commit Message?. In Proceedings of the 44th International Conference on Software Engineering, Pittsburgh Pennsylvania. ACM. 2022:2389-2401.

[23] https://cbea.ms/git-commit/

[24] https://commitlint.js.org/

[25] Faragó D, Färber M, Petrov C. A Full-Fledged Commit Message Quality Checker Based on Machine Learning. 2023 IEEE 47th Annual Computers Software and Applications Conference (COMPSAC). Torino Italy. IEEE PUBLICATIONS. 2023:788-799.