

Large Language Models Meet Commit Message Generation: An Empirical Study

Yufan Tang

Delft University of Technology

Large Language Models Meet Commit Message Generation: An Empirical Study

by

Yufan Tang

Student Name	Student Number
Yufan Tang	5701503

Thesis advisor: Ujwal Gadiraju
Daily Supervisor: Rihan Hai
Project Duration: February, 2024 - September, 2024
Faculty: Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

Preface

This report details the work conducted for my master's thesis project. It outlines the research questions I formulated, the experiments I carried out to address these questions, and the literature I reviewed throughout the process. Additionally, it includes a background chapter designed to equip the reader with the technical knowledge necessary to comprehend my work, assuming a foundational understanding of deep learning and basic linear algebra.

This project has been one of the most challenging endeavours I've undertaken, not only due to the complexity of the technical subject matter but also because of the demanding nature of working with my mentors. My daily co-mentor, Wenbo Sun, daily mentor, Professor Rihan Hai, and chair mentor, Professor Ujwal Gadiraju, were all exceptionally high standards, making collaboration sometimes challenging. Nevertheless, I am deeply grateful to each of them for their unwavering support and for remaining engaged throughout the project, even when my own motivation wavered. They were always prompt in answering my questions and showed genuine interest in my work during our meetings.

I also want to extend my thanks to Maliheh Izadi, another member of my thesis committee, for her involvement in evaluating the project.

Furthermore, I appreciate the participation of my computer science master classmates and predecessors Zihan Wang, Dongxu Lu, and Xinqi Li in the human evaluation of my project.

Finally, I am profoundly thankful to my family and closest friends for their constant love and support, which have been the foundation of my achievements, including this one.

*Yufan Tang
Delft, September 2024*

Abstract

In the realm of software development, commit messages are vital for understanding code changes, enhancing maintainability, and improving collaboration. Despite their importance, generating high-quality commit messages remains a challenging task, with existing methods often facing issues such as limited flexibility and high training costs. This paper addresses the research gaps in automated commit message generation (CMG) by exploring the capabilities of large language models (LLMs) in this domain. We specifically investigate the potential of the prompt engineering method to enhance LLM performance compared to state-of-the-art (SOTA) techniques such as RACE.

Our research begins with a comprehensive literature review of CMG methodologies, focusing on the effectiveness of various message formats and the limitations of existing approaches. To fill the gap, we introduce a unified commit message formats dataset and evaluate the previous LLM-based method on the dataset, utilizing the GPT model as a representative example of LLMs. By comparing the LLM zero-shot method with previous retrieval-based and hybrid methods, we provide a detailed analysis of the strengths and weaknesses of LLM-based approaches.

We further explore the impact of different retrieval augmented generation (RAG) configurations on CMG performance and investigate what constitutes a good demonstration of the LLM RAG method for the CMG task. That is followed by proposing a new prompt engineering method called Adaptive Retrieval Augmented Generation With Commit Type Classification And Partitioned Retrieval (ARC-PR), which incorporates a classification module and a database partitioning module to the LLM RAG system. Validating through comprehensive testing on the unified message format dataset, our experiments demonstrate that the proposed method shows significant improvements in message effectiveness compared to the previous LLM-based methods and in the aspects of informativeness, message format consistency and the balance between precision and recall, our method surpasses the state-of-the-art methods in the field. Further generalizability study illustrates the robustness of our proposed method. With the introduction of human evaluation, we further confirmed the superiority of our proposed method over the state-of-the-art methods in terms of informativeness and expressiveness.

In summary, this study makes several key contributions: it provides a thorough comparison of previous LLM-based methods with existing techniques, proposes an enhanced LLM prompt engineering approach specifically tailored for commit message generation (CMG) tasks that address the issue of low informativeness and expressiveness seen in past state-of-the-art methods and demonstrates performance that surpasses other LLM-based methods.

Contents

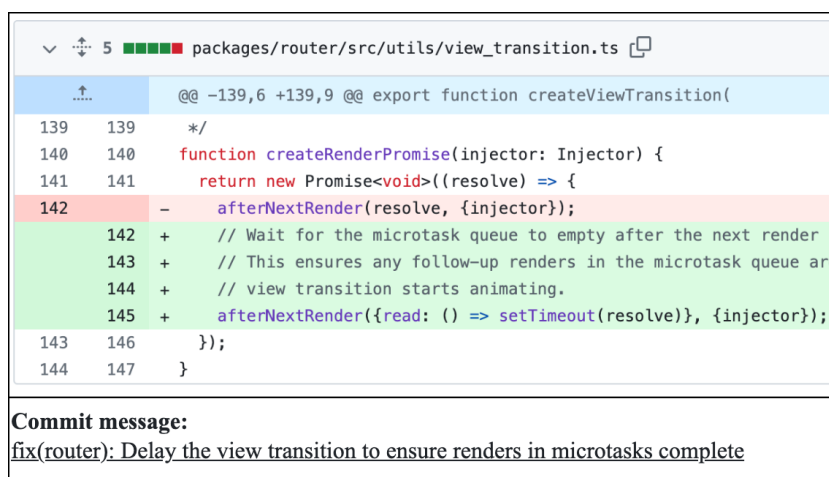
Preface	i
Abstract	ii
1 Introduction	1
1.1 Research Question	4
2 Background	6
2.1 What Is Commit	6
2.2 Why We Need Commit Message Generation	6
2.2.1 Quality Of Commit Message	7
2.2.2 Effectiveness Of Commit Message	7
2.3 Commit Message Format	7
2.3.1 Verb Direct Object Format	7
2.3.2 Angular Format	8
2.4 Large Language Models	8
2.4.1 Code Summarization	8
2.4.2 Code Embedding	9
2.4.3 Zero-shot Prompting	9
2.4.4 Retrieval-Augmented Generation	9
3 Related Work	12
3.1 Commit Message Generation Approach	12
3.1.1 Rule-based Method	12
3.1.2 Retrieval-based Method	13
3.1.3 Generation-based Method	13
3.1.4 Hybrid Method	14
3.1.5 LLM-based Method	14
3.2 Commit Message Generation Evaluation Approach	15
3.2.1 BLEU	15
3.2.2 BLEU-Norm	16
3.2.3 ROUGE-L	16
3.2.4 Meteor	17
3.2.5 Log-MNEXT	17
3.2.6 Human Evaluation	18
3.3 Discussion	19
3.3.1 The Challenge Of Commit Message Generation Approaches	19
3.3.2 Lack of detailed evaluation of message quality	20
3.3.3 Limited Human Evaluation	20
4 Methodology	21
4.1 Research Questions	21
4.2 Hypotheses	21
4.3 Method	23
4.3.1 Engineering Senario	24
4.3.2 Dataset Collection	24
4.3.3 Comparison Settings of LLM-based methods With Existing Methods	26
4.3.4 LLM - RAG System Design	27
4.3.5 Impact Of RAG Module Components On CMG Task Performance	29
4.3.6 What constitutes a good demonstration of the LLM RAG method	30

4.3.7	Adaptive Retrieval Augmented Generation With Commit Type Classification And Partitioned Retrieval (ARC-PR)	31
4.3.8	Metric Evaluation And Analysis	33
4.3.9	Human Evaluation	34
5	Result	37
5.1	Experimental Results For RQ1	37
5.2	Experimental Results For RQ2	39
5.2.1	The Impact Of Chunking Method And Embedding Method To LLM RAG	40
5.2.2	Evaluating The Hypothesis: Enhancing Model Generation Quality By Good Demonstration	42
5.2.3	Evaluate The Matching Accuracy Of The LLM RAG Method And ARC-PR Method	44
5.2.4	Comparative Analysis of ARC-PR With LLM RAG And LLM Zero-Shot Approaches	45
5.2.5	Comparative Analysis Of ARC-PR With SOTA Method RACE	47
5.2.6	Generalizability Evaluation: VDO Format Dataset	48
5.3	Experimental Results For RQ3	52
6	Discussions	55
6.1	Key Findings	55
6.2	Limitations	56
6.3	Future Work	57
7	Conclusion	58
	References	60
A	Human evaluation example	65
B	Human evaluation Participants Background	67

1

Introduction

In software development, a commit refers to the action of saving changes to a version control system (VCS), and a commit message is a brief explanation of the changes made in that commit. As illustrated in **Figure 1.1**, in this thesis, a commit consists of a code change and its corresponding commit message. Commit messages are crucial in VCS as they provide context and rationale for changes, helping developers understand the evolution of the codebase. Well-crafted commit messages enhance code maintainability, facilitate debugging, and improve collaboration among team members. According to [59, 78], effective commit messages are vital for software project management as they serve as a log of changes and aid in tracking the project's history.



The screenshot shows a code editor window for the file `packages/router/src/utils/view_transition.ts`. The code is displayed with line numbers and a diff view. The original code (line 142) is shown in red, and the new code (lines 142-145) is shown in green. The commit message is displayed below the code.

```
@@ -139,6 +139,9 @@ export function createViewTransition(  
139 139 */  
140 140 function createRenderPromise(injector: Injector) {  
141 141   return new Promise<void>((resolve) => {  
142 -   afterNextRender(resolve, {injector});  
142 +   // Wait for the microtask queue to empty after the next render h  
143 +   // This ensures any follow-up renders in the microtask queue are  
144 +   // view transition starts animating.  
145 +   afterNextRender({read: () => setTimeout(resolve)}, {injector});  
143 146   });  
144 147 }
```

Commit message:
`fix(router): Delay the view transition to ensure renders in microtasks complete`

Figure 1.1: Example of a Commit: Code Change and Corresponding Commit Message

Despite their importance, writing commit messages is often time-consuming and challenging for developers [19]. Research has shown that the quality of commit messages can vary significantly, often due to a lack of motivation or time [19, 48, 50, 55]. Developers may be uncertain about what information to include to craft a good commit message [79]. A study analyzing commit messages from five open-source projects found that approximately 44% of them required improvement, indicating considerable room for enhancement in manual annotation [79]. Consequently, there has been substantial interest in automating commit message generation to address issues of motivation and time constraints.

Early approaches to automating commit messages were predominantly rule-based, utilizing predefined templates and heuristics [5, 8, 46, 73]. However, these methods often failed to handle all scenarios and lacked a deep understanding of the reasons behind code changes. Retrieval-based methods then emerged, which generate commit messages by finding and adapting similar past messages [50, 21,

30, 33]. While these methods leverage past examples, they can struggle with retrieving appropriate messages for novel inputs.

With the advent of neural networks, researchers have increasingly turned to network models for training and generating commit messages [35, 52, 62, 37, 47, 53]. Hybrid methods combining retrieval and learning techniques have also been explored to improve performance [74, 14, 84, 48]. The current state-of-the-art (SOTA) in this domain is the RACE method [74], which incorporates these advancements [20].

However, high-performing methods in commit message generation (CMG) typically require extensive training data, making them resource-intensive. The rise of large language models (LLMs) has garnered attention due to their robust text-generation capabilities and efficiency in learning [98]. LLMs are advanced language models with billions of parameters, trained on vast datasets containing billions of tokens. Most modern LLMs leverage the Transformer decoder architecture [80]. A notable feature of LLMs is their in-context learning ability, which allows them to perform tasks in a few-shot setting (using just a few examples) or even a zero-shot setting (using a natural language description of the task without examples) [4].

In the software engineering domain, OpenAI's Codex [6], designed specifically for code, exemplifies this capability with strong performance in code generation [6, 42, 39, 91] and natural language generation tasks. Recently, OpenAI has released two general-purpose LLMs, GPT-3.5-turbo (ChatGPT) and GPT-4, whose capabilities are detailed in the GPT-4 technical report [1] and a survey by [49].

LLMs, trained on extensive corpora, excel in generating coherent and contextually appropriate text. They can adapt their output style based on minimal prompt modifications, a feature that previous methods lack. Despite the promise shown by LLMs, research in this area remains limited [93, 20, 51, 76, 90]. For example, [93] compared zero-shot LLMs with previous SOTA methods, revealing a preference for LLMs among human developers. To explore LLMs' in-context learning ability, [90] introduced retrieval-based demonstrations as a prompting method. However, previous research on LLMs often lacks rigorous human evaluation [93, 90] or fails to delve deeply into prompt engineering [90]. Consequently, these studies do not convincingly demonstrate the superiority of LLM-based methods over traditional CMG approaches.

In this thesis, we address existing research gaps by defining a specific application scenario: a company or project adopting a unified commit message format as the standard and guideline for writing commit messages. Although past research has recognized the impact of message format on message quality [36, 76] and explored various formats such as Verb Direct Object (VDO) and Angular in datasets for evaluating different CMG tools [35, 76], there is a noticeable lack of comprehensive studies focused on assessing the generation capabilities of Large Language Models (LLMs) across these formats. This gap underscores the need for targeted research that examines how LLMs perform in generating commit messages within diverse formatting frameworks, thereby providing a deeper understanding of their effectiveness and limitations in real-world applications.

To address this gap, we collect two unified message format datasets: one for primary testing and the other for robustness and generalizability analysis. We use the GPT-3.5 model as a representative example of LLMs for our experiments and analysis. Throughout the thesis, references to LLMs specifically pertain to the GPT model used in our research. For comparative benchmarks, we select RACE, a state-of-the-art (SOTA) method in the CMG field, and NNGen, a SOTA method in retrieval-based approaches.

In our experiments, we initially considered the zero-shot prompt method for LLM generation. We evaluate and analyze these methods using various metrics, providing a detailed comparison of the LLM zero-shot approach with previous methods in terms of effectiveness. This analysis lays the groundwork for understanding the generation capabilities of different methods.

Past research [32, 27] has demonstrated that Retrieval-Augmented Generation (RAG) can enhance model performance by retrieving relevant external data in response to queries, thus ensuring more accurate and up-to-date outputs. Further studies [85] have highlighted RAG's promising potential in software engineering applications. Additionally, [90] showed that well-crafted examples significantly improve LLM performance in commit message generation.

Despite these advancements, previous research on LLM-based methods has only partially explored prompting techniques [90]. Specifically, the impact of different module configurations within RAG systems on LLM generation capabilities has not been thoroughly investigated. Moreover, while prior work has indicated that demonstrations improve in-context learning for LLMs, it has not explored what constitutes an effective demonstration for LLM RAG in the commit message generation (CMG) task [90].

To address these gaps, we propose a novel LLM-based method: Adaptive Retrieval-Augmented Generation with Commit Type Classification and Partitioned Retrieval (ARC-PR). Our method involves several key components:

1. **Component Impact Analysis:** We first assess how different component settings affect the generation capabilities of the LLM RAG system.
2. **Effective Demonstration Identification:** Based on experimental results, we identify the elements that make a good demonstration for LLM RAG. We design validation experiments to test our hypotheses and confirm their validity.
3. **Method Enhancement:** We introduce a fine-tuned code model as a classifier and partition the original RAG database into separate sub-databases. This approach aims to increase the proportion of good demonstrations, thereby improving LLM generation capabilities.

We validate the efficacy of ARC-PR through extensive experimental comparisons with LLM zero-shot and LLM RAG methods across various effectiveness metrics. To further assess its robustness and generalizability, we use a VDO format dataset. We comprehensively evaluate our method and compare it with the state-of-the-art (SOTA) method RACE using numerical metrics and human evaluations. This thorough assessment highlights the strengths of our proposed approach.

In summary, current research in Large Language Models (LLMs) within the Commit Message Generation (CMG) domain reveals several gaps:

1. **Balance high generated message quality and low cost:** Despite the variety of commit message generation (CMG) approaches—rule-based, generation-based, retrieval-based, and hybrid—achieving a balance between low training costs and high quality remains challenging. Recent advances in large language models (LLMs) show that they can deliver high-quality results at low costs through prompt engineering, especially in natural language generation tasks. While LLMs may not always excel in traditional metrics, they often produce more informative and expressive commit messages that human evaluators prefer.
2. **Performance in Specific Scenarios:** There is a need to thoroughly investigate how LLMs perform in various engineering scenarios, particularly in generating commit messages within diverse formatting frameworks.
3. **Prompt Engineering:** The potential of prompt engineering to enhance LLM generative capabilities is not fully explored. Existing methods have made some advancements, but their performance remains below that of the current state-of-the-art (SOTA) method, RACE. This shortfall is partly due to insufficient comprehensive evaluation metrics and underutilization of LLM capabilities.
4. **Human Evaluation:** Qualitative differences in the generated texts, assessed through human evaluation, remain largely unexplored.

To address these gaps, our research involves the following key contributions:

- We provide a comprehensive assessment of LLM-based commit message generation approaches compared to existing CMG methods, highlighting their respective strengths and limitations.
- We explored the impact of various component configurations on the performance of the LLM RAG method.
- We investigated what constitutes a good demonstration of the LLM RAG method for the CMG task.
- We propose a new approach called Adaptive Retrieval Augmented Generation with Commit Type Classification and Partitioned Retrieval (ARC-PR). This method increases access to high-quality

demonstrations related to code changes compared to existing CMG techniques, addressing the issues of low informativeness and expressiveness observed in previous state-of-the-art methods.

- Experimental results show that ARC-PR overall achieves state-of-the-art performance in commit message generation and each component in ARC-PR is effective.
- We introduced a comprehensive human evaluation, assessing our method and the state-of-the-art method RACE from multiple perspectives, including informativeness and expressiveness.

This thesis not only reveals the promising potential of prompt engineering in LLMs for generating commit messages but also highlights the shortcomings and limitations of some existing methods.

1.1. Research Question

The main goal of this thesis is to answer the following question:

Can the Adaptive Retrieval Augmented Generation with Commit Type Classification and Partitioned Retrieval (ARC-PR) method further improve the performance of large language models (LLMs) on commit message generation (CMG) tasks, and what are the advantages and disadvantages of the ARC-PR method in message generation compared to previous state-of-the-art (SOTA) method?

For clarity, we split this question up into three subquestions.

Research Question 1: *What is the effectiveness of LLM zero-shot on commit message generation in a unified message style dataset compared to the existing methods?*

We initially aim to gain a clear understanding of the LLM zero-shot method through this research question, evaluating its strengths and weaknesses in commit message generation compared to other methods. This will help to better frame subsequent research questions.

To answer this question, we conducted an in-depth literature review focusing on what is a good commit message, the useful commit message format and previous CMG methodologies. In the literature review section, we summarize in detail the angular format and the VDO format, and the common approaches to automatically generate the commit message. After the literature review section, we collected the unified message style dataset by extracting specific subsets from the existing dataset and we picked two methods from previous research.

To comprehensively evaluate LLM on the unified message style dataset, we employed an empirical research methodology for testing. Specifically, we chose two previous state-of-the-art methods as the baseline, and we evaluated these three methods on five commonly used CMG metrics. Based on the five assessment criteria, we delved into the aspects of effectiveness they encompass and conducted a comparative analysis. This allowed us to comprehensively evaluate the strengths and weaknesses of LLM zero-shot methods.

Research Question 2: *What is the impact of different component settings on the generation capabilities of the LLM RAG method? What constitutes a good demonstration of the LLM RAG method? Can the effectiveness of the LLM method in generating commit messages be improved by ARC-PR? And what is the effectiveness of the ARC-PR method compared to the state-of-the-art method? Additionally, what is the robustness and generalizability of the ARC-PR method?*

We aim to explore our proposed method through this research question by analyzing and comparing its strengths and weaknesses with previous LLM methods and the SOTA method using various metrics. This will provide a more comprehensive understanding and evaluation of our method.

To answer this question, we conducted an in-depth literature review focusing on the improvement way of LLM and the structure of retrieval-augmented generation.

We designed several sets of parameter contrast experiments for RAG, studying the impact of different parameter configurations on RAG generation performance. Additionally, we discussed what constitutes a good demonstration of the LLM RAG method by designing and performing an ideal validation experiment. With the conclusions of the previous experiments, we propose a

new method called ARC-PR, which incorporates a classifier and a database partitioning module. We compared our proposed method with previous LLM approaches and the SOTA method on the experimental dataset to assess the advantages and disadvantages of the proposed method. After conducting a further generalizability study, we have gained a deeper understanding of the robustness and generalizability of ARC-PR.

Research Question 3: *How does the effectiveness of the ARC-PR method compare to state-of-the-art approaches in generating commit messages in human application evaluation?*

We aim to use this research question to compare the message generation quality of our method with the SOTA method at an evaluative level. This will supplement the aspects that numerical metrics in Research Question 2 cannot assess.

To answer this question, we conducted an in-depth literature review focusing on the human evaluation in the CMG domain. In the literature review section, we summarize in detail how previous research conducted their human evaluation. After that, we proposed our evaluation workflow and criteria.

In the experimental part, we invited several participants to conduct a human evaluation of the commit messages generated by our proposed method and the RACE method. The commit messages were assessed from two perspectives: expressiveness and informativeness. Finally, statistical methods were employed to sample and analyze the participants' evaluations, and to verify the significance of the differences between the two methods in terms of expressiveness and informativeness.

2

Background

This chapter contains preliminaries to the thesis. The following topics are discussed: What is a commit (Section 2.1), why do we need the commit message and automatic commit message generation tools (Section 2.2), common commit message format (Section 2.3), what are Large Language Models and the basic concepts about prompting method (Section 2.4).

2.1. What Is Commit

A large volume of code is created and updated daily, typically managed by version control systems such as Git. This historical record serves as a valuable resource for understanding the evolutionary process of software development [38].

A commit is a fundamental concept in version control systems (VCS), such as Git, representing a snapshot of changes made to the codebase at a specific point in time. It captures the state of the entire project or repository at the moment the commit is created, including all changes to files, such as additions, modifications, and deletions [97].

Each commit contains at least one file's code modification. Along with the changes, a commit also includes metadata such as:

- **Author:** The person who made the changes.
- **Committer:** Sometimes different from the author, this is the person who actually added the commit to the repository.
- **Timestamp:** The date and time when the commit was made.
- **Commit Message:** A brief description of the changes, which helps others understand the purpose of the commit.

In a software development team, it's crucial for developers to understand the reasons behind code changes, especially when multiple people are working on the same codebase. Understanding why code was added, deleted, or modified helps prevent conflicts and facilitates collaboration. Raw diffs often fall short in this regard, as they only display textual differences between file versions, which can be lengthy and confusing, and may not address high-level questions developers might have [46]. Commit messages and bug reports provide essential context and explanations [71]. However, due to the volume and diverse nature of daily tasks performed by software developers [56, 10, 60], commit messages frequently lack useful information, often containing generic statements like "initial commit" or being almost empty [46]. To address this issue, researchers have proposed techniques for automatically generating commit messages, which have shown some promise [77].

2.2. Why We Need Commit Message Generation

Commit message generation aims to automatically create meaningful and descriptive messages that explain the changes made in a particular code commit. These messages help developers understand

the history and context of changes, making it easier to manage the codebase and collaborate with others [5].

Here are several reasons why we need a high-quality commit message:

- **Clarity:** Assists developers in understanding and modifying code [5].
- **Traceability:** Makes it easier to track and understand the evolution of the codebase [5].

2.2.1. Quality Of Commit Message

Software development is increasingly conducted in distributed environments, involving developers from diverse cultural backgrounds. Over the past 20 years, the surge in commercial involvement in open-source software (OSS) projects has further diversified the developer workforce. This diversity can lead to variations in the quality of commit messages due to differing development cultures and practices [79].

A study [79] analyzing commit messages from five open-source projects found that, on average, approximately 44% of commit messages were deemed in need of improvement. This variation in quality can make it challenging to understand the rationale behind code changes and can hinder effective collaboration among developers.

To improve the quality of commit messages and ensure they effectively convey essential information, it is important to include specific elements in the message. Research findings [79] suggest that a well-written commit message should address two key aspects: "What" and "Why." The "What" component summarizes the changes made in the commit, providing a clear overview of the modifications introduced to the codebase. The "Why" component explains the motivation behind the changes, detailing the reasons and context that led to the code modifications.

Incorporating both the "What" and "Why" elements in commit messages allows developers to create messages that not only document the changes made but also provide valuable insights into the reasoning behind those changes. This comprehensive approach to crafting commit messages can significantly enhance understanding, collaboration, and maintenance of software projects within a development team.

2.2.2. Effectiveness Of Commit Message

In this thesis, we evaluate the effectiveness of the commit message generation (CMG) tool using several metrics: BLEU [65], BLEU-Norm [77], ROUGE-L [44], METEOR [3], and Log-MNEXT [13]. The assessment focuses on several key aspects:

- **N-gram Matching Accuracy:** The accuracy of matching coherent phrases between generated and reference messages.
- **Total effective information:** The total amount of useful information contained in the message.
- **Effective Information Density:** The amount of useful information per unit length of the message.
- **Message Format Consistency:** The similarity in format between the generated message and the reference message.
- **Balance Between Precision and Recall:** The extent to which the length of the generated message approximates the length of the reference message.

By considering these aspects, we provide a comprehensive evaluation of the CMG tool's effectiveness.

2.3. Commit Message Format

Commit messages, which adhere to different formats, represent various standards and project requirements. These formats aim to standardize how committers document their changes and encourage them to include all necessary information in their messages.

2.3.1. Verb Direct Object Format

Some previous research has utilized datasets featuring the VDO (Verb Direct Object) format for commit messages. The VDO format refers to messages structured as "verb + object." Researchers such as

[35, 50, 47, 84, 36] first identified this format, noting that 47% of commit messages start with a verb followed by its direct object. They categorized 15 types of verbs, listed in **Table 2.1**, which represent 70% of all VDO data and are considered the most common and illustrative.

Verb Types	Verb Types	Verb Types
add, create, make, implement	use	handle
fix	move, change	rename
remove	prepare	allow
update, upgrade	improve	set
ignore	revert	replace

Table 2.1: Verb Groups

2.3.2. Angular Format

To create informative and easy-to-understand commit messages, some projects, like AngularJS ¹, have established and adhere to specific formatting guidelines. Research [76] has shown that following the AngularJS commit message rules significantly improves the quality of commit messages.

AngularJS has defined a commit message format as `<type>(<scope>): <subject>` ². In this format:

- "type": This must be one of the types listed in **Table 2.2**.
- "scope": This part is optional and specifies the area of the codebase affected by the commit.
- "subject": This provides a succinct description of the change.

This structured approach helps ensure clarity and consistency in commit messages.

Types	Types	Types
build	ci	docs
feat	fix	perf
refactor	style	test
chore		

Table 2.2: Angular Type Groups

2.4. Large Language Models

Recently, the success of large language models (LLMs) in natural language processing (NLP) [17, 70] has led an increasing number of software engineering (SE) researchers to explore their application across various SE tasks. These tasks encompass code generation [18, 83], program repair [94, 95], and code summarization [2, 82, 75, 28, 25]. These studies highlight the strong learning capabilities and potential of LLMs in code-related domains.

In this section, we first discuss the application of LLMs in code summarization and code embedding, both of which are closely related to the commit message generation (CMG) field explored in this thesis. Additionally, we introduce two prompting methods—zero-shot and retrieval-augmented generation—as essential background knowledge for our subsequent research.

2.4.1. Code Summarization

The task of commit message generation is closely related to code summarization, as both require a nuanced understanding of the code to produce high-quality results. Success in these tasks hinges on the model's ability to grasp the intricacies of the codebase. Research in code summarization has extensively explored the capabilities of large language models (LLMs) [2]. Models like Codex [6] have been

¹Angular Commit Guidelines: <https://github.com/angular/angular.js/blob/master/DEVELOPERS.md#commit-message-format>

²<https://github.com/angular/angular.js/blob/master/DEVELOPERS.md#commit-message-format>

employed for code summarization, also known as docstring generation. For instance, [24] introduced the InCoder model and performed zero-shot training using the CodeXGLUE [54] Python dataset.

Previous studies [43, 96] have demonstrated that information retrieval (IR) methods and the use of similar code snippets can significantly enhance an LLM's ability to generate effective text content in code summarization. This evidence supports the hypothesis that the retrieval-augmented generation (RAG) method could also be effective in the commit message generation (CMG) domain.

2.4.2. Code Embedding

Code embedding can be viewed as a specialized form of text embedding. In natural language processing (NLP), embedding refers to the process of converting text or other types of data into dense, continuous vector representations. These vectors capture the semantic meaning of the data in a format that machine learning models can efficiently process. Essentially, an embedding is a numerical representation of data, often expressed as a high-dimensional vector.

The field of text representation in NLP has seen significant evolution over recent decades. From basic representations to more sophisticated embeddings, these advancements have greatly improved the capability of machines to process and understand human language with greater precision [69].

For the word embedding (Representing individual words as vectors), there are several types of models³:

- Static word embedding:
 - No machine learning: TF-IDF (Term Frequency-Inverse Document Frequency)
 - With machine learning: Word2Vec [58], GloVe [66]
- Contextualized word embedding:
 - RNN-based: ELMO [67]
 - Transformer-based: BERT [12], ALBERT [41]

Large Language Models (LLMs) significantly advance beyond earlier models by providing unparalleled depth and breadth of knowledge through their word and sentence-level embeddings. Trained on vast datasets, these models capture a broad spectrum of human language variations, producing embeddings that reflect a deep understanding of context and concepts. The shift from TF-IDF to advanced LLM embeddings represents a major leap forward in achieving more contextually aware text representations in natural language processing (NLP). This progress continues to propel the field, expanding the possibilities for applications such as text clustering, sentiment analysis, and more [69]. Given the exceptional performance of LLMs in text representation tasks, we will leverage LLMs to enhance the representation of code change snippets in our research.

2.4.3. Zero-shot Prompting

Zero-shot prompting involves using a natural language prompt to elicit a desired response from a language model without providing specific examples or additional training for the task. This approach leverages the model's pre-existing knowledge and its capacity to generalize from the prompt alone to perform the task effectively.

2.4.4. Retrieval-Augmented Generation

Although large language models have been shown to have a strong ability to capture large amounts of in-depth knowledge from data [68], such models have their shortcomings: they cannot easily expand or revise their memory, they are unable to provide a reliable factual basis for the generation of some results and even produce "hallucination" [57].

As an alternative to embedding extensive real-world knowledge within large language models (LLMs), retrieval augmentation allows models to acquire sufficient knowledge for a task on demand. The core concept of retrieval augmentation involves incorporating an information retrieval step before making predictions. During this step, relevant texts related to the task are retrieved from a large corpus. The

³<https://medium.com/@eordaxd/mastering-the-art-of-embeddings-choosing-the-right-model-for-your-rag-architecture-38e15a9adcbc>

model then makes predictions based on both the input context and the retrieved texts. This approach effectively transforms a closed-book task into an "open-book" scenario. Consequently, fine-tuned models can perform well even with much smaller sizes, as the necessary knowledge is obtained through retrieval rather than being stored within the model itself.

Figure 2.1 illustrates the workflow of a Retrieval-Augmented Generation (RAG) system. Here's a step-by-step description of the process:

1. **Load Documents:** The system begins by loading a set of documents.
2. **Generate Document Chunks:** These documents are then divided into smaller, manageable pieces known as chunks.
3. **Vectorize Document Chunks:** Each document chunk is converted into a dense vector representation (embedding) using an embedding model.
4. **Store Embeddings:** The embeddings, along with their corresponding document chunk IDs, are stored in a vector database (Vector DB).
5. **Vectorize Question:** When a query is received, it is also converted into an embedding using the same or a compatible embedding model.
6. **Retrieve Relevant Document Chunk IDs:** The query embedding is used to search the vector database to find the most relevant document chunk IDs.
7. **Retrieve Document Chunks:** Using these IDs, the system retrieves the corresponding document chunks from storage.
8. **Use LLM to Generate Answer:** The retrieved document chunks, along with the original question, are fed into a large language model (LLM). The LLM uses this information to generate a coherent and contextually appropriate answer.
9. **Generated Answer:** The final output is the generated answer, which is returned to the user.

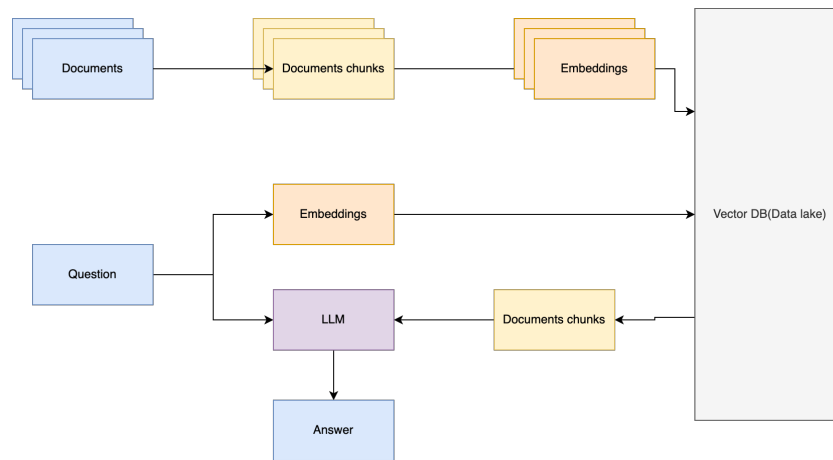


Figure 2.1: Workflow of the LLM RAG Method

In a Retrieval-Augmented Generation (RAG) system, various component choices significantly impact the final generation results. Among the critical components are the chunking module and the embedding module. The chunking module involves decisions on chunking size and chunking method, while the embedding module primarily focuses on the choice of embedding model. These selections are crucial as they determine the quality and relevance of the retrieved information, which in turn influences the final output of the RAG system.

Chunking: Chunking is the process of breaking down a large piece of text or data into smaller, manageable, and meaningful units, often referred to as "chunks." This technique is used in various natural language processing (NLP) tasks to improve the efficiency and accuracy of data processing and retrieval.

Chunking size: This refers to the length or size of each chunk. The size can vary depending on the application and the nature of the text. For example, chunks could be sentences, paragraphs, or fixed-length segments of text.

Chunking method: This refers to the strategy used to divide the text into chunks. Common methods include:

- **Sentence-Based Chunking:** Dividing text into chunks based on sentence boundaries.
- **Fixed-Length Chunking:** Dividing text into chunks of a predetermined number of words or characters.
- **Semantic Chunking:** Dividing text into chunks based on semantic units or logical sections, such as paragraphs or topic boundaries.

Embedding: In the LLM RAG method, the embedding module is primarily used to vectorize the input document, which has already been chunked. It converts the original text into high-dimensional numerical vectors for further processing ⁴. More detailed embedding techniques are discussed in section 2.4.2.

In conclusion, Retrieval-Augmented Generation (RAG) presents a robust framework with broad applicability across various problems and tasks. In this thesis, we investigate the use of RAG specifically for commit message generation, exploring its potential to enhance this critical aspect of software development.

⁴<https://www.rungalileo.io/blog/mastering-rag-how-to-select-an-embedding-model>

3

Related Work

This chapter consists of three parts. First, discuss related work on the commit message generation (CMG) approach (Section 3.1), and explore the pros and cons of these approaches. After that, related work on CMG evaluation methodologies (Section 3.2). Lastly, we identify some challenges in the current-day CMG landscape and why we focus our research on LLM and prompting method (Section 3.3).

3.1. Commit Message Generation Approach

In this section, we examine the strengths and weaknesses of previous research methods, highlight areas needing improvement, and analyze the advantages of utilizing LLMs as a generative approach.

We examine prominent commit message generation approaches introduced in recent years, which can be categorized into five types [20, 90] (In previous research, commit message generation approaches were generally categorized into four types. This thesis introduces the concept of LLM-based methods, thereby expanding the classification to five categories.): rule-based methods, generation-based methods, retrieval-based models, hybrid models and LLM-based methods. We summarize the strengths and weaknesses of these five methods in the table below.

	Weakness	Strength
Rule-based method	Limited flexibility	Low construction cost
Retrieval-based method	Poor contextual relevance	High generation speed
Generation-based method	High cost for training	Better contextual relevance
Hybrid method		Better flexibility
LLM-based method	Untapped potential	Low cost of extra training More flexibility

Table 3.1: Comparison of LLM-Based Methods with Previous Approaches in the CMG Domain

Based on the strengths and weaknesses of various methods summarized in **Table 3.1**, we believe that LLM-based methods hold significant research potential. Their advantages, including the low cost of extra training and greater flexibility, further motivate us to pursue related research.

The subsequent sections are organized as follows: Sections 3.1.1, 3.1.2, 3.1.3, and 3.1.4 will specifically detail the advantages and disadvantages of previous methods. Section 3.1.5 will compare our work with past LLM-based methods, highlighting the advancements and innovations of our approach.

3.1.1. Rule-based Method

Early research translates code changes into natural language descriptions using predefined rules and templates [5, 8, 73, 46].

Pros:

- Consistency: These rule-based methods ensure a standardized format for commit messages, making it easier for team members to understand the nature of changes quickly [46].

Cons:

- Limited Flexibility and Adaptability: The predefined templates may not cover all possible scenarios, leading to potentially inadequate descriptions for complex changes, and they do not learn from new data, meaning they cannot improve their performance over time or adapt to new patterns in code changes [97, 90, 62].

3.1.2. Retrieval-based Method

Some studies [34, 50, 33] adopt information retrieval (IR) approaches to reuse commit messages of similar code changes.

NNGen [50]: leverages the Nearest Neighbor(NN) algorithm to generate commit messages from diffs. For a new diff request, NNGen calculates the cosine similarity between the target code diff and each code diff in the training set. Subsequently, the algorithm selects the top-k code differences from the training set to calculate the BLEU scores between each of these variances and the target code difference. The variance with the highest BLEU score is identified as the most akin code difference, and its associated commit message is designated as the target commit message.

Pros:

- No Need for Large Training Datasets: Unlike machine learning-based methods, retrieval-based approaches do not require extensive labelled datasets, making them easier and faster to implement [20].

Cons:

- Less Contextual Understanding: The reused commit messages might not correctly describe the content/intent of the current code change. These methods might not capture the deeper context or rationale behind a code change as effectively as methods that utilize semantic analysis or contextual information [90].
- Limited Flexibility: Retrieval-based methods might struggle to adapt to new or unseen types of changes since they rely heavily on history commits [20].

3.1.3. Generation-based Method

Recently, numerous deep learning (DL)-based approaches for generating commit messages have been proposed [92, 35, 52, 47, 37]. Generation-based methods typically employ an encoder-decoder architectural framework and are trained on datasets gathered from open-source repositories.

Pros:

- Advanced models like CodeBERT [23] leverage pre-trained weights to better understand the relationship between code changes and natural language, potentially leading to more relevant and accurate commit messages [90].

Cons:

- Limited Flexibility: Learning-based techniques tend to generate commit messages that are relatively short (averaging 4.5 tokens), suggesting that they still exhibit limited performance in producing more flexible or complex commit messages [15].
- Data Dependence: The effectiveness of the model heavily relies on the quality and diversity of the training dataset. If the dataset is biased or lacks variety, the model may produce suboptimal results [97].
- High cost for training: These approaches either involve training models from scratch or fine-tuning a pre-trained language model using labelled data, which may be impractical due to limited computing resources and the scarcity of labelled data [90].

3.1.4. Hybrid Method

Hybrid methods are mostly based on retrieval-based methods and learning-based methods. They combine the advantages of past methods to achieve better performance than some past methods [48, 74, 84].

For instance, **RACE** [74] introduces a novel approach to generating commit messages in software development by leveraging retrieval-augmented techniques. The RACE method aims to improve the quality and accuracy of commit messages by incorporating retrieved similar commits as exemplars to guide the neural network model in generating more informative, concise, and expressive messages. The method of RACE involves the following key steps:

1. **Retrieval of Similar Commits:** Similar commits are retrieved based on the code diff of the current commit being processed. These retrieved commits serve as exemplars for guiding the generation of the current commit message.
2. **Exemplar Guider:** An exemplar guider is used to align the retrieved commit message with the current code diff, ensuring semantic similarity and relevance. This guide helps the neural network model in capturing the essential information from the exemplar commit to enhance the quality of the generated commit message.
3. **Neural Network Model:** A neural network model is employed to generate the final commit message based on the input code diff and the guidance provided by the exemplar commit through the exemplar guider. This model is trained on a large parallel code and natural language corpus, fine-tuned on commit message generation datasets, and evaluated on a diverse set of programming languages.

RACE is currently regarded as state of the art in commit message generation domain [20].

Pros:

- **Improved Accuracy:** By leveraging retrieved similar commits as exemplars, the hybrid approach can produce more contextually relevant and accurate commit messages. The IR (Information Retrieval) component ensures that the generated messages are grounded in real examples, which can enhance semantic relevance [97].

Cons:

- **Training Challenges:** Training a hybrid model can be more challenging, as it requires careful tuning of both the retrieval and generation components to work effectively together [97].

3.1.5. LLM-based Method

The emergence of large language models (LLMs) has addressed several limitations of traditional methods—rule-based, retrieval-based, learning-based, and hybrid methods to a significant extent.

- LLMs have a strong ability to solve general problems [22], which enables them to perform well on specific tasks even without explicit training.
- Compared with traditional deep learning (DL) approaches, LLMs exhibit robust contextual learning capabilities. Simple examples can help LLMs quickly understand and adapt to specific tasks [16, 4].

In recent years, numerous LLMs have been introduced, showcasing significant potential across a wide range of software engineering tasks [22, 31, 64]. The performance of LLMs in commit message generation was first explored by Eliseeva et al. [20], who provided foundational insights into the capabilities and limitations of LLMs for this specific task.

Building on this foundation, Zhang et al. [93] conducted a detailed experimental study comparing ChatGPT and Llama2 LLMs with prior state-of-the-art (SOTA) methods. Their findings revealed that, although LLMs did not outperform existing methods based on BLEU and Rouge-L metrics, a significant proportion of the LLM-generated results were preferred by human evaluators. However, the study lacked agreement validation among participants, potentially leading to significant errors. Additionally, the impact of different prompting methods on LLMs' generation of commit messages was not explored.

In a related study, Tao et al. [76] designed various prompts for ChatGPT using the MCMD dataset [77]. Despite not surpassing their proposed method in terms of BLEU, METEOR, and Rouge metrics, their experimental comparison highlighted the potential of LLMs for commit message generation.

To further investigate the in-context learning abilities of LLMs, [90] introduced retrieval-based demonstrations as a prompting method to enhance LLM performance. Their method did not delve deeper into the architecture of retrieval but instead focused on studying the impact of the number of demonstration examples on the performance of LLMs. The approach did not yield significant advantages compared to some previous SOTA methods. While they recognized the potential of the approach, they did not conduct any human evaluation to compare the generation quality of LLMs using retrieval-based demonstrations versus the SOTA methods. This omission somewhat detracts from the completeness of their study.

Based on previous LLM-based research, this thesis explicitly outlines the practical application scenarios addressed by our method and introduces a new LLM-based approach aimed at improving existing methods. We compare this new approach with state-of-the-art methods from various effectiveness perspectives and further validate its superiority through human evaluation. We compare our method with past LLM-based approaches as shown in the table below:

	Engineering Scenario	In-depth metric analysis	Human evaluation
LLM zero-shot [93]	No	No	Limited number of people
LLM with retrieval demonstration [90]	No	No	No
Our approach	Yes	Yes	Statistically reliable human assessment

Table 3.2: Comparison of Our Approach with Previous LLM-Based Methods

In summary, past research on LLM-based methods has demonstrated the significant potential of LLMs in the CMG field and showcased their in-context learning capabilities through prompting methods. However, these studies often focus on overly general engineering scenarios, lacking in-depth exploration of LLMs in specific contexts. Additionally, they typically only perform simple score comparisons at the metrics level without delving into the message quality attributes represented by these metrics. Furthermore, some studies lack rigorous human evaluation experiments. This thesis aims to address these gaps, providing a foundation for further exploration into optimizing these technologies to enhance collaboration and efficiency in software development.

3.2. Commit Message Generation Evaluation Approach

Various metrics frequently employed in NLP tasks like machine translation and text summarization can be applied to assess commit message generation. These metrics encompass BLEU [65], Meteor [3], Rouge-L [44], Cider [81], etc. In this research, our emphasis is on BLEU [65], BLEU-Norm [52], Rouge-L [44], Meteor [3], and Log-MNEXT [13], as these metrics encompass a wealth of information related to effectiveness.

3.2.1. BLEU

The BLEU [65] (Bilingual Evaluation Understudy) score evaluates the quality of generated translations by combining n-gram (contiguous sequences of n words) matching precision and text brevity. BLEU is commonly used in previous commit message generation tasks [35, 50]. The following is the formula to calculate the BLEU Score:

$$BLEU = BP * exp\left(\sum_{n=1}^N w_n \log p_n\right) \quad (3.1)$$

where w_n is the weight of n-gram precision p_n , which can be obtained as Equation 3.3. $N = 4$ and uniform weights $w_n = 1/4$ if not explicitly specified.

BP is the brevity penalty which is computed as:

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \quad (3.2)$$

where c is the length of the candidate generation and r is the length of the reference.

The n-gram precision p_n can be obtained as:

$$p_n = m_n / l_n \quad (3.3)$$

where m_n is the Number of matched n-grams, and l_n is the total number of n-grams in the candidate generation.

3.2.2. BLEU-Norm

BLEU-Norm [77] is a variant of the BLEU metric [65].

Compared to BLEU, BLEU-Norm is case insensitive as it converts all characters both in the reference and the generation to lowercase before calculating scores. The smoothing method [45] is adopted in BLEU-Norm to smooth the calculation of n-gram precision scores. It adds a constant number (one) to both the numerator and denominator of p_n for $n > 1$:

$$p_n = \begin{cases} m_n / l_n & \text{if } x = 1 \\ (m_n + 1) / (l_n + 1) & \text{if } x > 1 \end{cases} \quad (3.4)$$

3.2.3. ROUGE-L

ROUGE-L is a metric within the ROUGE (Recall-Oriented Understudy for Gisting Evaluation) [44] family that evaluates the quality of a summary by measuring the Longest Common Subsequence (LCS) between the system-generated summary and the reference summary. To calculate the ROUGE-L score, we first need to identify the LCS, which is the longest sequence of words that appears in both summaries in the same order, though not necessarily consecutively.

ROUGE-L precision (P) is the ratio of the length of the LCS to the length of the system-generated summary:

$$Precision = \frac{LCS(S, R)}{|S|} \quad (3.5)$$

where $LCS(S, R)$ is the length of the LCS of the system-generated summary S and the reference summary R , and $|S|$ is the length of the system-generated summary.

ROUGE-L recall (R) is the ratio of the length of the LCS to the length of the reference summary:

$$Recall = \frac{LCS(S, R)}{|R|} \quad (3.6)$$

where $|R|$ is the length of the reference summary.

ROUGE-L F-measure (F1) is the harmonic mean of Precision and Recall:

$$F1 = \frac{(1 + \beta^2) \cdot P \cdot R}{R + \beta^2 \cdot P} \quad (3.7)$$

Typically, β is set to 1, which gives equal weight to Precision and Recall:

$$F1 = \frac{2 \cdot P \cdot R}{R + P} = \frac{2 \cdot LCS(S, R)}{|R| + |S|} \quad (3.8)$$

ROUGE-L score usually refers to ROUGE-L F1 score.

3.2.4. Meteor

METEOR (Metric for Evaluation of Translation with Explicit ORdering) [3] is a metric used to evaluate the quality of machine translation outputs. Compared to BLEU, METEOR takes into account more factors, such as synonym matching, stemming, and word order, and is therefore often considered a more comprehensive evaluation metric.

It can be calculated using the following formula:

$$Meteor = (1 - FragPenalty) * \left(\frac{PR}{\alpha P + (1 - \alpha)R} \right) \quad (3.9)$$

$$P = \frac{m}{L_{pred}} \quad (3.10)$$

$$R = \frac{m}{L_{ref}} \quad (3.11)$$

$$FragPenalty = \gamma * \left[\frac{\#chunks}{\#unigrams\ matched} \right]^\beta \quad (3.12)$$

Where m stands for the number of matching unigrams, P stands for precision and R stands for recall. L_{pred} is the length of predicted text and L_{ref} is the length of reference text.

3.2.5. Log-MNEXT

Log-MNEXT [13] is a modified version of the METEOR-NEXT metric [11] proposed in the context of evaluating commit messages. Log-MNEXT incorporates various factors such as exact word matchings, stemmed and paraphrased matchings, string lower-casing, and punctuation removal as preprocessing steps before comparing predicted and reference texts.

One key improvement of Log-MNEXT over METEOR-NEXT is in handling cases where the reference and predicted messages are identical. In such cases, Log-MNEXT assigns no penalty score, unlike the fragmentation penalty factor in METEOR-NEXT, which penalizes such cases based on the number of unigram matches.

The Log-MNEXT score is calculated using the F-score, which considers precision and recall with weighted unigram matches, shown in 3.13, where P stands for Precision, shown in 3.14, R stands for Recall, shown in 3.15, and FragPenalty formula shown in 3.16. where for a matcher type $i \in \text{exact match, stemmed match, synonym match}$, w_i is the weight and m_i is the number of matched unigrams. L_{pred} is the length of predicted text and L_{ref} is the length of reference text. For the optimum values of α , β and γ , Log-MNEXT use the values suggested by Denkowski and Lavie [11].

The metric aims to provide a more comprehensive and accurate evaluation of commit message generation tools compared to traditional metrics like BLEU.

$$Log - MNEXT = (1 - FragPenalty) * \left(\frac{PR}{\alpha P + (1 - \alpha)R} \right) \quad (3.13)$$

$$P = \frac{\sum w_i m_i}{L_{pred}} \quad (3.14)$$

$$R = \frac{\sum w_i m_i}{L_{ref}} \quad (3.15)$$

$$FragPenalty = \begin{cases} 0 & \text{if } Ref \equiv Pred \\ \gamma * \left[\frac{\#chunks}{\#unigrams\ matched} \right]^\beta & \text{otherwise} \end{cases} \quad (3.16)$$

3.2.6. Human Evaluation

The numeric metrics perform a quantitative assessment of performance changes based on a numeric analysis of the overlap of tokens. However, such evaluations are rigid and lack subtlety, and sometimes are not aligned with human judgement. Generated messages could have little overlap with human written messages, and yet convey a very similar meaning. Thus qualitative assessment is essential to give a more intuitive impression towards the performance change.

Human evaluation is a very effective method in qualitative analysis [9], which is widely used to validate the effectiveness of the commit message generation approaches in the previous work [14, 93, 35, 50, 48, 29, 76].

[35, 50, 48, 14] used the scoring to evaluate the commit message. Their human evaluation aimed to complement the automatic evaluation using the BLEU metric by focusing on the semantic similarity of the generated messages to the reference messages. They pay more attention to the correlation between generated messages and reference messages. By analyzing the distribution of these median scores and applying statistical tests, the authors can conclude whether their proposed method outperforms other methods in generating commit messages that are semantically similar to reference messages. [14] further conducted a Wilcoxon signed-rank test [87] between the scores of FIRA and the other techniques to statistically prove the reliability of their human evaluation.

[29] considered three different perspectives (Relevance, Usefulness, Content Adequacy) and asked participants to score. The final score was the average of the three scores. [76, 74] also considered three different perspectives (Conciseness, Expressiveness, Content Adequacy) and let the participants score range between 0 and 4. [76, 74] further conducted the Kendall rank correlation coefficient calculation [40] to verify the agreement of all raters.

Different from previous research methods, [93] randomly shuffled the human reference and generated message together, and asked participants to judge which commit message corresponding to the label best matched the current code diff based on the code difference. This selection process was based solely on the fit between the commit message and the code difference, without knowing the origin of each commit message.

Kendall Rank Correlation Coefficient (Kendall's Tau)

The Kendall rank correlation coefficient [40], also known as Kendall's Tau (τ), is a statistic used to measure the ordinal association between two measured quantities. It assesses the strength and direction of the relationship between two variables by comparing the concordance and discordance of their rankings.

Wilcoxon Signed-rank Tests

The Wilcoxon signed-rank tests [88] is a non-parametric statistical test used to determine whether there is a significant difference between paired samples. Unlike parametric tests, which assume a normal distribution of the data, the Wilcoxon signed-rank test does not make such assumptions, making it useful when the data doesn't meet parametric test assumptions.

We have summarized the content of the above human evaluation by examining aspects such as the number of candidates, agreement among raters, criteria, and significance tests. This summary compares past methods with the proposed method, as shown in the table below.

	Method	Number of candidate	Agreement among raters	Criteria	Significant difference test
Hybrid method	Fira [14]	6	No	Yes	Yes
	COME [29]	6	Yes	Yes	Yes
	RACE [74]	4	Yes	Yes	Yes
	KADEL [76]	3	Yes	Yes	No
LLM-based method	LLM zero-shot [93]	2	No	No	No
	LLM with retrieval demonstration [90]	0	No	No	No
	Our approach	4	Yes	Yes	Yes

Table 3.3: Comparison of Various Methods in the CMG Field Based on Human Evaluation

From **Table 3.3**, we can see that our method, compared to other LLM-based methods, not only increases the number of candidates but also introduces more detailed criteria for evaluating message generation quality from multiple aspects. Additionally, we have incorporated statistical tests to verify rater agreement and assess the significance of differences in evaluation metrics.

3.3. Discussion

In this section, we summarize the challenges faced in the current commit message generation (CMG) landscape and explain why our research focuses on large language models (LLMs) and prompt engineering methods. Additionally, we also highlight some critical aspects that were missing in past methods, which could contribute to making the experiments more comprehensive.

3.3.1. The Challenge Of Commit Message Generation Approaches

Balance high generated message quality and low cost: Despite the existence of various CMG approaches, including rule-based, generation-based, retrieval-based, and hybrid models, these methods often struggle to balance low training costs with high generation quality, as achieving high generation quality typically entails higher training costs. Recent advancements in LLMs have demonstrated their significant potential across various software engineering tasks, particularly in natural language generation. In some domains, LLMs can generate high-quality results at a very low cost through prompt engineering. Although LLMs may not always surpass existing methods in traditional evaluation metrics within the CMG domain, human evaluators often prefer LLM-generated results, suggesting that these models can produce more informative and expressive commit messages.

In this thesis, we will delve into the application of prompt engineering in the CMG domain using LLMs, with the goal of achieving a balance between low training costs and high generation quality.

Impact of RAG Architecture on LLM Performance: Previous research [90] has lacked a thorough exploration of the Retrieval-Augmented Generation (RAG) architecture, leading to an underutilization of LLMs' generation potential in commit message generation (CMG) tasks. To address this gap, this thesis will delve into the components of the RAG architecture and investigate what constitutes a good demonstration to improve the performance of LLMs. In summary, this thesis will provide a comprehensive investigation into the application of RAG prompt engineering in CMG tasks. This forms the basis for our research question RQ2.

Exploration of Prompt Engineering in LLMs: A notable feature of LLMs is their in-context learning ability, which allows them to perform tasks in a few-shot setting (using just a few examples) or even a zero-shot setting (using a natural language description of the task without examples) [4]. Despite the promise shown by LLMs [93, 90], there is still insufficient exploration of their potential in commit message generation. In this thesis, we propose a prompt engineering approach specifically designed for the task of commit message generation. This approach not only outperforms existing prompt engineering methods in various generation performance metrics but also surpasses state-of-the-art methods for the commit message generation (CMG) task across multiple indicators and comprehensive evaluations.

3.3.2. Lack of detailed evaluation of message quality

Although previous studies have conducted various evaluations on a range of NL indicators [77, 76, 93, 90], few have thoroughly explored the characteristics related to message quality represented by these indicators. This thesis aims to provide a more in-depth analysis of each metric, offering a more comprehensive evaluation of the commit message generation method.

3.3.3. Limited Human Evaluation

Many studies either have not conducted human evaluations [90] or have done so in a manner that lacks scientific rigour [93]. This lack of thorough and rigorous validation limits the ability to demonstrate the superiority and practical utility of the proposed methods. Human assessments provide essential feedback that can guide improvements in generation models. The absence of qualitative evaluations can lead to a disconnect between automated metrics and actual user satisfaction.

Based on this, this thesis will conduct detailed and rigorous human evaluations to ensure the completeness and integrity of the experimental research. This approach will help bridge the gap between automated metrics and real-world user experience, providing a more accurate assessment of the proposed methods' effectiveness. This forms the basis for our research question RQ3.

4

Methodology

In this section, we elaborate on the research questions introduced in Section 4.1, outline our hypotheses in response to those questions (Section 4.2), and provide a detailed explanation of our experimental design. This includes the process of collecting experimental data, the experimental testing of LLM-zero-shot, LLM-RAG, our proposed new method, and the metric analysis techniques used (Section 4.3).

4.1. Research Questions

To reiterate, this thesis aims to fully leverage the capabilities of large language models (LLMs) in the domain of commit message generation through prompt engineering.

We aim to achieve this by exploring the fundamental generative capabilities of LLMs and investigating how different structural configurations of RAG impact their generative performance, as well as how demonstrations influence LLM generation abilities. Building on these insights, we propose an optimized prompt engineering approach called ARC-PR (Adaptive Retrieval Augmented Generation with Commit Type Classification and Partitioned Retrieval). Our goal is to enhance the potential of LLMs through advanced prompt engineering, enabling them to achieve superior generative performance compared to previous methods.

The research questions we formulated for this were:

RQ1. What is the effectiveness of LLM zero-shot on commit message generation in a unified message style dataset compared to the existing methods?

RQ2. What is the impact of different component settings on the generation capabilities of the LLM RAG method? What constitutes a good demonstration of the LLM RAG method? Can the effectiveness of the LLM method in generating commit messages be improved by ARC-PR? And what is the effectiveness of the ARC-PR method compared to the state-of-the-art method? Additionally, what is the robustness and generalizability of the ARC-PR method?

RQ3. How does the effectiveness of the ARC-PR method compare to state-of-the-art approaches in generating commit messages based on human evaluation?

4.2. Hypotheses

For **RQ1**, we formulate the following hypothesis:

H.1.1 LLM performs worse than RACE, but better than NNGen on high-quality datasets on all metrics.

Motivation: This hypothesis is based on previous research involving LLMs, which has shown that in commonly used commit message generation (CMG) metrics such as BLEU-Norm, BLEU-Moses, and

ROUGE-L, LLMs tend to outperform NNGen but fall short of RACE on the MCMD dataset [93, 90]. Consequently, we hypothesize that LLMs will exhibit similar performance trends on high-quality datasets with a consistent style compared to these past results.

If hypothesis H.1.1 holds, our answer to RQ1 is that the LLM zero-shot method lags behind the other two methods in terms of the total amount of effective information, message format consistency, phrase matching accuracy, effective information density, and the balance between precision and recall. If hypothesis H.1.1 does not hold, for example, if the LLM zero-shot method performs better than the other methods on certain metrics, conclusions should be drawn based on the specific metrics.

RQ2 is first motivated by the observation that different component configurations of RAG may influence the generation results. Based on this understanding, we propose the following hypothesis:

H.2.1 Different chunking sizes and embedding models can have a significant effect on the RAG system.

If hypothesis H.2.1 holds, the selection of parameters for the ARC-PR method and the LLM RAG method should be based on the experimental results to identify the optimal chunking size and embedding model as the final components. If hypothesis H.2.1 does not hold, we will select no chunking and any embedding model as the components for the ARC-PR method and the LLM RAG method for subsequent comparative experiments.

Some studies [86] have shown that providing effective demonstrations can help LLMs infer latent variables containing task-related information, thereby improving their performance in context learning. However, within the CMG domain, there is no research specifying what constitutes a good demonstration for LLMs. Based on our analysis of the unified message dataset, we hypothesize that a good demonstration for an LLM is one where the input commit matches the commit type of the demonstration. This approach could significantly enhance the LLM's in-context learning capabilities and, consequently, improve its performance on CMG tasks.

H.2.2 The performance of the LLM RAG method can be improved with the demonstration of the commit that has the same commit type as the input commit

If hypothesis H.2.2 holds, then the answer to RQ2, "What constitutes a good demonstration of the LLM RAG method?" is that the demonstration of the commit that has the same commit type as the input commit is the good demonstration in CMG task, and the effectiveness of the LLM RAG method in generating commit messages can be improved by better demonstrations. If hypothesis H.2.2 does not hold, we will further analyze the reasons for its failure.

Based on the previous hypothesis, we will further validate the effectiveness and robustness of our proposed method. If H.2.2 holds true, we would reasonably expect that our method can provide better demonstrations, thereby enhancing the generation capabilities of the LLM. Therefore, we have the following hypothesis:

H.2.3 ARC-PC method can improve the retrieval accuracy of the RAG system. This, in turn, surpasses the generation effectiveness of the LLM zero-shot and LLM RAG method across various aspects, and it will exhibit a better degree of robustness and generalizability over the previous LLM-based method in another unified format dataset.

If hypothesis H.2.3 holds, then the answer to RQ2, "Can the effectiveness of the LLM method in generating commit messages be improved by ARC-PR?" is that the effectiveness of the LLM method in generating commit messages can indeed be improved by ARC-PR. The answer to RQ2, "What is the robustness and generalizability of the ARC-PR method?" is that ARC-PR exhibits better robustness and generalizability compared to the LLM RAG method. If hypothesis H.2.3 does not hold, we will further analyze the reasons for its failure.

Past research [90] has demonstrated that, with the enhancement of retrieval demonstrations, the generative capabilities of LLMs are comparable to hybrid approaches. In our study, we explore the generative capabilities of the ARC-PR method with the state-of-the-art (SOTA) hybrid approach.

Based on the assumption that the previous hypothesis holds, we propose the following hypothesis:

H.2.4 The ARC-PR method will surpass the RACE method in various evaluation aspects of message generation, such as information density, total information content, and phrase-matching accuracy. It will exhibit a better degree of robustness and generalizability over the RACE method in another unified format dataset.

If hypothesis H.2.4 holds, then the answer to RQ2, "What is the effectiveness of the ARC-PR method compared to the state-of-the-art method?" will be based on the specific qualitative improvements observed in effectiveness. If hypothesis H.2.4 does not hold, we will conduct a detailed analysis and discussion of the results to identify the reasons for any shortcomings in effectiveness.

RQ3 builds on the assumption that both H.2.2 and H.2.3 hold. The past research [77, 13] has demonstrated a high correlation between these metrics and human evaluations, if our method outperforms RACE across all metrics, it is likely to also surpass RACE in human assessments. Therefore, we propose the following hypothesis:

H.3.1 The ARC-PR method outperforms the state-of-the-art method RACE in human evaluation. Specifically, it surpasses RACE in both informativeness and expressiveness, demonstrating superior performance in generating more comprehensive and engaging commit messages.

If hypothesis H.3.1 holds, we believe that our method performs better than the RACE method in human evaluations, with a specific level of improvement depending on the differences in informativeness and expressiveness. If hypothesis H.3.1 does not hold, we will consider our method to be inferior to RACE in human evaluations and will further analyze the reasons for its shortcomings.

4.3. Method

This section provides an overview of the experimental setup used to address the research questions. It includes details on the research scenario, datasets utilized, the LLM zero-shot method, the LLM retrieval-augmented generation (RAG) method, and our proposed approach, Adaptive Retrieval Augmented Generation with Commit Type Classification and Partitioned Retrieval (ARC-PR). Additionally, we cover stability metrics and human evaluations. Beyond answering the research questions through these experiments, we will also analyze various commit message generation (CMG) evaluation metrics. This analysis will explore how different metrics highlight distinct aspects of the generated commit messages, enabling a more comprehensive evaluation of the results and a better-informed response to the research questions

To address Research Question 1 (RQ1), we first curated a unified message format dataset (Section 4.3.2) and evaluated the performance of the large language model (LLM) using zero-shot learning as well as existing methodologies on this dataset (Section 4.3.3). We employed metric analysis (Section 4.3.8) to assess the results comprehensively.

For Research Question 2 (RQ2), we designed a foundational Retrieval-Augmented Generation (RAG) system (Section 4.3.4) and analyzed its components (Embedding module and Chunking module) (Section 4.3.5) to identify potential modifications and improvements. Additionally, we designed a validation experiment to investigate what constitutes a good demonstration for LLM RAG. Based on the hypothesis H.2.2, we introduced a new approach that incorporates a commit-type classification and retrieval database partitioning module based on the RAG architecture and conducted a series of experiments to validate its superiority over the LLM RAG and LLM zero-shot methods. We then compared our proposed method with the SOTA method RACE and utilized metric analysis (Section 4.3.8) to evaluate its effectiveness.

In response to Research Question 3 (RQ3), we constructed a novel human evaluation dataset for the experiment and performed statistical analysis to interpret the results (Section 4.3.9).

Our main contributions include:

- A comprehensive comparison of LLM zero-shot methods with existing approaches, highlighting their respective strengths and limitations.
- Exploration of the impact of various component configurations on the performance of the LLM RAG method.

- Investigating what constitutes a good demonstration of the LLM RAG method for the CMG task.
- Proposing a new approach called Adaptive Retrieval Augmented Generation with Commit Type Classification and Partitioned Retrieval (ARC-PR). This method increases access to high-quality demonstrations related to code changes compared to existing CMG techniques, addressing the issues of low informativeness and expressiveness observed in previous state-of-the-art methods.
- Experimental results show that ARC-PR overall achieves state-of-the-art performance in commit message generation and each component in ARC-PR is effective.
- Introducing a comprehensive human evaluation, assessing our method and the state-of-the-art method RACE from multiple perspectives, including informativeness and expressiveness.

4.3.1. Engineering Scenario

This thesis focuses on a key application scenario where a company or project requires adherence to a standardized commit message format. While previous research has recognized the impact of message format on message quality [36, 76] and has explored various formats (such as Verb Direct Object (VDO) and Angular) in datasets to evaluate different CMG tools [35, 76], there is a significant lack of comprehensive studies assessing the generation capabilities of Large Language Models (LLMs) across these formats. This gap highlights the need for targeted research to evaluate how LLMs perform in generating commit messages within different formatting frameworks, providing a deeper understanding of their effectiveness and limitations in practical applications.

4.3.2. Dataset Collection

To effectively address Research Questions RQ1, RQ2, and RQ3, we need to align data collection with the constraints imposed by different commit message formats. Commit messages vary in format, such as the Angular Format and the V-DO Format, each with distinct constraints that affect message quality. This study aims to implement the most efficient prompt engineering methods. We will evaluate the performance of various LLM-based methods using the same unified message format dataset. Specifically, we will use the Angular Format Dataset to assess the effectiveness of different prompt engineering methods and existing approaches. Additionally, the V-DO Format Dataset will serve as an auxiliary dataset for a generalizability study, evaluating the robustness of the prompt engineering methods.

Since predecessors have collected a very detailed, unfiltered commit dataset [20], this thesis will filter and preprocess the data based on it instead of re-collecting and constructing the new dataset.

Basic Filtering Settings:

- Past research has often imposed certain restrictions on diff Length, mainly because many learning-based methods have limitations on input length [35, 47]. To study the generation of CMG methods on long commits, we set the diff length to 5000. We chose to use length as the unit rather than tokens because tokenization methods vary, and additionally, our subsequent research involves chunking, where using length as the unit makes it easier to conduct grouped experiments.
- Past research has imposed certain restrictions on the length of commit messages, primarily to prevent commit messages from containing excessive redundant information. In this experiment, the filtering parameter for commit message length is set to no longer than 30 tokens, which is consistent with the settings used in most previous experiments [20].
- In the data pre-sampling phase, we found that commits with change types 'add' or 'delete' that conform to Angular format and V-DO format were very limited. Therefore, during commit filtering, we only considered commits with change type 'modification'.
- Based on the findings of [76], JavaScript projects contain the most Angular Format commit messages. Therefore, we use JavaScript as a filtering criterion and collect commits only from JavaScript projects. To ensure that the programming language does not introduce additional variables, we also consider only JavaScript projects for the composition of the V-DO dataset.

Based on the above Basic Filtering Settings, we will filter CommitChonicle separately according to the Angular Format and the V-DO Format.

The specific method for Angular Format filtering The message must have the pattern "`<type><scope>`":

<subject>”. Here, “<type>”, “<scope>”, and “<subject>” are text placeholders, and “<type>” should be one of the ten types defined by Angular¹: build, chore, ci, docs, feat, fix, perf, refactor, style, test.

After the above filtering, we got 55850 commits from the train valid and test data in the CommitChronicle dataset. **Table 4.1** is an overview of the dataset that we collected:

Type	Count
build	1115
ci	1334
docs	10031
feat	10371
fix	10371
perf	489
refactor	6818
style	915
test	3935
chore	10371

Table 4.1: Angular types and counts in our collected database

And we did stratified sampling to get 3.5% test data from the entire collected dataset which has the same ratio among all types. **Table 4.2** is an overview of the test data that we collected:

Type	Count
build	39
ci	47
docs	359
feat	371
fix	371
perf	17
refactor	244
style	32
test	144
chore	371

Table 4.2: Angular types and their counts in test set

The specific method for V-DO Format filtering The message has a pattern like “<verb><object>”. Here “<verb>” and “<object>” are the text. To simplify some situations, for example, the group id 1 defined by [36] is “add, create, make, implement”, and here we only consider “add” type, and also for group id 4 is “update, upgrade”, we only consider “update” type. So “<type>” should be one of the fifteen types defined by [36]: add, fix, remove, update, ignore, use, move, prepare, improve, revert, handle, rename, allow, set, replace.

After the above filtering, we got 32587 commits from the train valid and test data in the CommitChronicle dataset. **Table 4.3** is an overview of the dataset that we collected:

¹<https://github.com/angular/angular/blob/22b96b9/CONTRIBUTING.md#-commit-message-guidelines>

Type	Count
add	4261
fix	4261
remove	4261
update	4261
ignore	625
use	1672
move	493
prepare	291
improve	3476
revert	745
handle	1367
rename	1430
allow	1815
set	2039
replace	1590

Table 4.3: VDO Types and Their Counts in Our Collected Database

And we did stratified sampling to get 3.5% test data from the entire collected dataset which has the same ratio among all types. **Table 4.4** is an overview of the test data that we collected:

Type	Count
add	261
fix	261
remove	261
update	261
ignore	38
use	102
move	30
prepare	17
improve	213
revert	45
handle	83
rename	87
allow	111
set	125
replace	97

Table 4.4: VDO Types and Their Counts in the Test Set

4.3.3. Comparison Settings of LLM-based methods With Existing Methods

In this section, we introduce our comparison experiment settings for RQ1 and RQ2. For RQ1, we designed a zero-shot prompting method for LLMs and compared it with existing state-of-the-art (SOTA) methods. For RQ2, we compared the ARC-PR method with the existing SOTA approaches.

LLM Experiment Settings: To address RQ1 and RQ2, we use the GPT model [63] as a representative LLM in our experiments. Specifically, we employed the GPT-3.5 model (gpt-3.5-turbo-0125) via the OpenAI API². We configured the API hyperparameters following prior work [61, 26, 7], setting the temperature to 0 to ensure deterministic output. Given that the context window of GPT-3.5 is 16,385 tokens, we do not need to crop the input in our subsequent experiments.

²<https://openai.com/index/chatgpt/>

To answer RQ1, we used zero-shot prompts to establish the baseline for LLM performance. We employed the same prompt as used in previous research [93] for the LLM zero-shot method:

```
1 The following is a diff which describes the code changes in a commit, Your task is to write a
  short commit message accordingly. [DIFF] According to the diff, the commit message
  should be:
```

Where [DIFF] is code diff slot.

The Hybrid methods share similar advantages and disadvantages with generation-based methods, while retrieval-based methods have characteristics akin to rule-based methods. Therefore, we consider RACE [74] as a representative of Hybrid methods (which is also the current state-of-the-art method in the field of commit message generation) and NNGen [50] as a representative of retrieval-based methods.

NNGen Since [50] open-sourced the relevant code of the NNGen model³, we reused their code and we replaced the train data with our own database data and replaced the test data with our own test data.

RACE [74] have open-sourced their code⁴ and on this basis [20] further provided a RACE model trained on the Chronicle dataset, so we used the transformers tool provided by hugging face [89] to call the pre-trained RACE model for testing.

To validate Hypothesis H.1.1, we use LLM zero-shot method and compare it with the existing methods on the Angular format dataset collected in Section 4.3.2.

To validate Hypothesis H.2.2, we use the method that we introduced in Section 4.3.7 and compare it with the LLM RAG method that we introduced in Section 4.3.4 and the LLM zero-shot method on the Angular format dataset collected in Section 4.3.2. For the configuration (the choice of chunking size and embedding model) of the LLM RAG method and the ARC-PR method, we use the conclusion from the experiment of Section 4.3.5 as the reference. Additionally, we will assess the ARC-PR method's generalizability on the V-DO format dataset collected in 4.3.2.

To validate Hypothesis H.2.3, we compare the ARC-PR method with the SOTA method RACE on the Angular format dataset collected in Section 4.3.2. We also evaluate its generalizability on the V-DO format dataset collected in 4.3.2.

4.3.4. LLM - RAG System Design

To answer RQ2 and verify hypothesis H.2.1, we first designed an RAG system for CMG tasks based on the definition of the basic modules of RAG.

In our LLM-RAG system design, there are a total of 4 modules, namely Chunking Module, Embedding Module, Retrieval Module and Generation Module. The following is a detailed introduction to the composition and functions of these four modules:

Chunking Module In the chunking module, all input code diffs are split into one or more chunked documents for subsequent processing based on the configured chunking size and the corresponding chunking algorithm. The chunking size defines the maximum length of each chunked document and the length of overlap with the context, while the chunking algorithm defines the type of words that are split between two chunked documents.

Embedding Module In the embedding module, all chunked documents are converted into computable vectors by the embedding model.

Retrieval Module In the retrieval module, the input vector will be compared with all the vectors stored in the database by cosine distance. The vector stored in the database with the closest distance will be retrieved as a candidate, combined with the calculated results of other input vectors, and finally accumulated to obtain the final most similar code diff for output. The detailed accumulation algorithm is mentioned in section 4.3.6.

³<https://github.com/Tbabm/nngen>

⁴<https://github.com/DeepSoftwareAnalytics/RACE>

Generation Module In this module similar code diff with message will be combined with input code diff to form a new LLM prompt, which is finally input into LLM and returns the corresponding message output. The following is the detailed process of prompt combination.

Prompt Design: For the retrieved augmented generation prompt, we performed the prompt with the following template:

```

1  You are an assistant for commit message generation tasks. Use the following
   pieces of retrieved-context, which is the most similar code diff that we
   retrieve from the database, learn the pattern and logic from them and
   generate the commit message for the input code diff.
2
3  Similar code diff: [context]
4  Similar code diff commit message: [msg]
5  Input Code diff: [diff]
6  Answer:

```

Where [context] represents the similar code diff retrieved by the RAG system, [msg] represents the corresponding commit message of that similar code diff. [diff] represents the code diff slot.

Combining all the modules (Chunking Module, Embedding Module, Retrieval Module and Generation Module) mentioned above constitutes our RAG pipeline, as shown in the following figure:

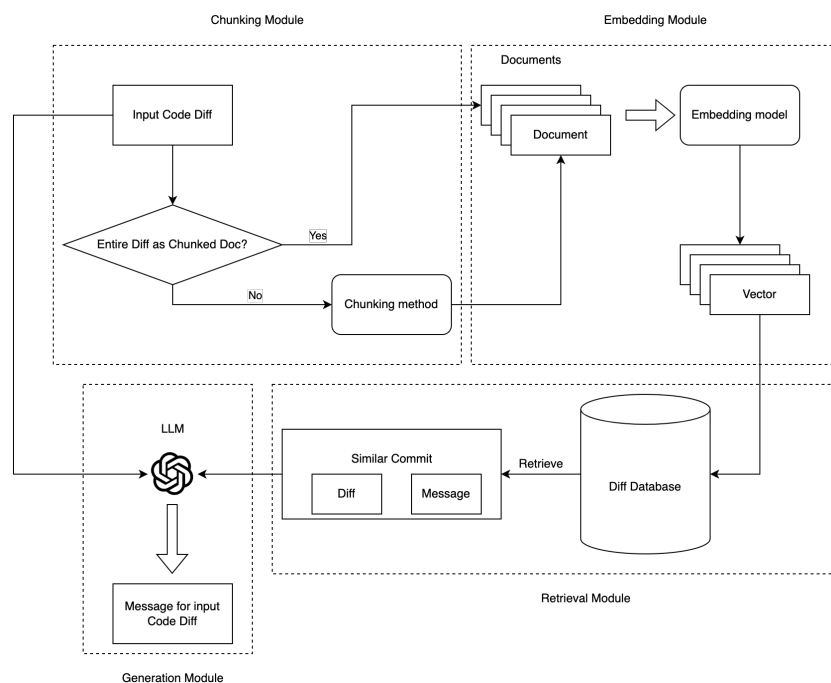


Figure 4.1: Overview of the LLM RAG Pipeline

As shown in **Figure 4.1**, The process starts with an "Input Code Diff," which is a set of code changes made to the project. A decision point checks whether the entire diff should be treated as a chunked document. If the chunking size is the entire code diff, no modification will be applied to the original input code diff. If the chunking size is smaller than the length of the entire code diff, the diff will be broken down into smaller chunks using a specific "Chunking method." After the chunking module, the input code diff will be processed into one or several subcode diffs and that's what we call a document in our system.

The document is then passed to an "Embedding model" which converts it into numerical representations called "Vectors." This is done to enable comparison with other documents. The result from the embedding model is a set of vectors representing the input code diff.

These vectors are then compared against a database, referred to as the "Diff Database," which contains vectors of previous code diffs along with their commit messages. The chunking method used to store data in the database depends on the chunking method used in the input code diff. That is, if the chunking size used in the chunking module for the input code diff is 500, then all previous code diffs in the diff databases will also use a chunking size of 500 during the preprocessing phase before being stored in the database. Based on cosine distance, the system searches for a "Similar Commit" within the database. This involves finding the most similar vectors to the input code diff. Each similar commit in the database is linked to a corresponding "Diff" and "Message." In terms of technology selection, specifically, the index database we chose is Chroma DB. Chroma DB is an open-source vector store used to store and retrieve vector embeddings. Similarity computation uses the HNSW index.

Finally, a new LLM prompt will be formed by the similar code diff with its corresponding message and the input code diff. After prompting the LLM, the LLM will return the corresponding message as output.

4.3.5. Impact Of RAG Module Components On CMG Task Performance

To answer RQ2 and verify hypotheses H.2.1, we designed two sets of experiments to verify the effects of different chunking sizes on RAG generation results, the effects of different embedding models on RAG generation results, and the effects of the presence or absence of classification and database partitioning modules on RAG generation results.

In the experimental section, we use the test set the same as RQ1 to validate the impact of these variables. We evaluate the effect of different system parameters by referencing the BLEU, BLEU-Norm, Rouge-L, Meteor and Log-MNEXT scores of the commit messages generated by the RAG on the test set. Suppose the scores of these two metrics under a certain configuration are significantly better than those under other configurations. In that case, we consider that particular configuration to significantly impact the system. Otherwise, it does not have a considerable impact.

Embedding Model Experiment

For the embedding model experiment, we selected three of the latest embedding models trained on code data: mxbai-embed-large-v1 (335M parameters), all-MiniLM-L6-v2 (22.7M parameters), and e5-small-v2 (33M parameters). We initially hypothesized that the impact of different embedding models on the RAG system is consistent across different chunking sizes (i.e., if using model A as the embedding model results in the best generation effect when chunking size=1000, then using model A as the embedding model should also result in the best generation effect when chunking size=2500). Therefore, in this experiment, we chose to use the entire code diff as the retrieval unit. We kept all other parameters constant and only evaluated the impact of different embedding models.

Chunking Method Experiment

For the chunking method experiment, we hypothesize that the impact of different chunking sizes is consistent across different embedding models (i.e., if a smaller chunking size enhances the result for model A, it should also do so for model B). Thus, in this experiment, we uniformly used e5-small-v2 as the system's embedding model. Since we are dealing with JavaScript project code, we employed the RecursiveCharacterTextSplitter from LangChain, which is specifically designed for JavaScript code. Therefore, aside from the chunking function, we focused on the impact of chunking size on retrieval results.

Given that the maximum length of our data is 5000, we set the chunking sizes to 5000 (entire length), 2500, 1000, and 500. For the chunk_overlap, we set it to 1/5 of the chunking size (e.g., if the chunking size is 1000, the overlap is 200). When the chunking size is not the entire length, a single code diff will be split into several sub-code diffs. Consequently, during RAG retrieval, the input test data must also be chunked using the same method. The specific method for similar search is as follows:

Similar Code Diff Retrieve Algorithm:

1. **With entire code diff as retrieval document unit:**

Given:

- $D_{\text{test}} = \{d_1, d_2, \dots, d_m\}$ is the set of test documents.

- $R(\cdot)$ represents the retrieval algorithm that retrieves the most similar document for a given input document.
- `similar_diff` is the list storing the most similar document for each test document.

For each test document $d_i \in D_{\text{test}}$, perform the retrieval:

$$\text{similar_diff}[i] = R(d_i)[0]$$

Where $R(d_i)[0]$ returns the most similar document for the content of d_i .

The overall process can be summarized as:

$$\text{similar_diff} = [R(d_i)[0] \mid d_i \in D_{\text{test}}]$$

2. With sub-code diff as retrieval document unit:

Given:

- For input code diff `DIFF_i`, firstly it will be divided into $D = \{d_1, d_2, \dots, d_n\}$ by using the corresponding chunking method, $D = \{d_1, d_2, \dots, d_n\}$ is the set of documents to be processed.
- C is the set of candidate documents in the database db .
- A is a dictionary storing aggregate scores for each candidate ID, initialized as $A(c) = 0$ for all $c \in C$.

Step 1: Initialization

$$A(c) = 0, \quad \forall c \in C$$

Step 2: Similarity Search and Score Aggregation

For each document $d_i \in D$, perform a similarity search to get relevance scores:

$$S(d_i) = \{(c_{i1}, s_{i1}), (c_{i2}, s_{i2}), \dots, (c_{im_i}, s_{im_i})\}$$

where c_{ij} is a candidate ID and s_{ij} is the similarity score for document d_i .

Update the aggregate score for each candidate ID c_{ij} :

$$A(c_{ij}) \leftarrow A(c_{ij}) + s_{ij}, \quad \forall (c_{ij}, s_{ij}) \in S(d_i)$$

Step 3: Determine the Candidate ID with the Maximum Score

Identify the candidate ID c^* with the highest aggregate score:

$$c^* = \begin{cases} \arg \max_{c \in C} A(c) & \text{if } A \neq \emptyset \\ -1 & \text{if } A = \emptyset \end{cases}$$

4.3.6. What constitutes a good demonstration of the LLM RAG method

During the experiments with the LLM RAG method, we found that the retrieved similar commits had low relevance to the input commit. Specifically, the angular type of the retrieved similar commit messages differed from that of the input commit message, indicating that the commit types were not the same. This poor demonstration quality ultimately led to the LLM generating messages with an angular type different from that of the input commit message. In contrast, high-quality demonstrations provide similar commits with the same angular type as the input commit for reference, which can enhance the quality of the LLM's output messages. This is illustrated in **Figure 4.2**.

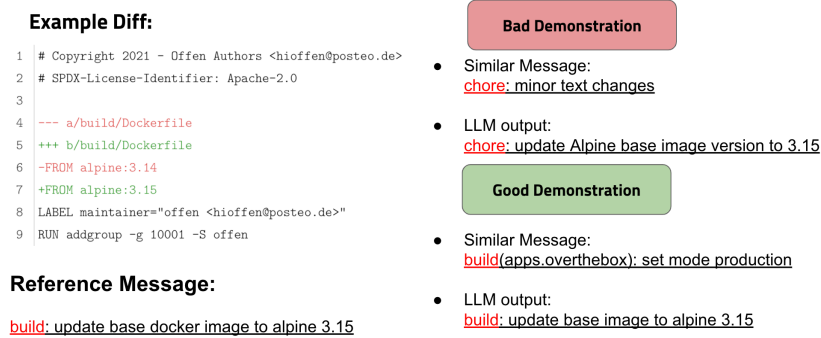


Figure 4.2: Examples of Similar Commit Retrieval

To verify H.2.2, we conducted the following verification tests:

1. We first partition the database in the LLM RAG method. The original database is divided into 10 sub-databases based on angular type.
2. We used the actual commit message type of the test data as input to query the relevant commits in the corresponding type-specific sub-database (for example, if the input commit type is "chore," we perform the query in the sub-database that stores commits of the "chore" type).
3. Finally, we constructed the following prompt 4.3.6 using the similar commits and the input commit, and then input it into the LLM to generate the corresponding commit message.

```

1 You are an assistant for commit message generation tasks. Use the
2 following pieces of retrieved-context, which is the most similar
3 code diff that we retrieve from the database, it has the same
4 change type as the input code diff, you need to learn the message
5 pattern and logic from the example code diff and write a short
6 commit message accordingly.
7
8 Similar code diff: {context}
9 Similar code diff commit message: {msg}
10 code diff type: {type}
11 Input Code diff: {diff}
12 Commit Message:

```

Where [context] represents the similar code diff retrieved by the RAG system, [msg] represents the corresponding commit message of that similar code diff. [type] represents the type that is classified by our method. [diff] represents the code diff slot.

Different from the original RAG prompt logic mentioned above, in this new method we explicitly tell LLM what type of code diff is input to assist LLM in generating commit messages.

4.3.7. Adaptive Retrieval Augmented Generation With Commit Type Classification And Partitioned Retrieval (ARC-PR)

Based on the conclusion of H.2.2, to validate H.2.3, we will first introduce our proposed method, Adaptive Retrieval Augmented Generation With Commit Type Classification And Partitioned Retrieval (ARC-PR). In this method, we introduced a classification module before the retrieval module to help the system identify the angular type of the input commit message. To match the classified data, we further split the original database into sub-databases according to categories. **Figure 4.3** shows the detailed design of the proposed system.

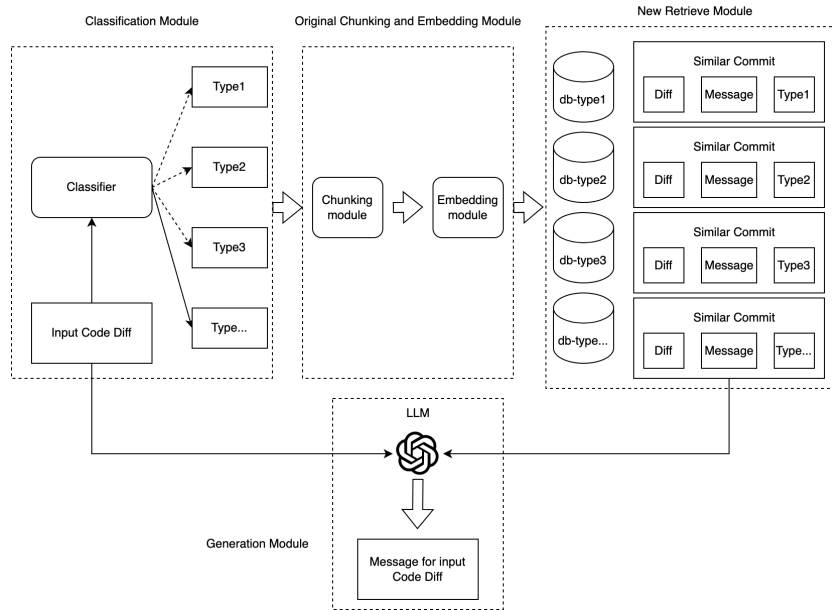


Figure 4.3: ARC-PR Method's Workflow

As shown in **Figure 4.3**, for the classification module, specifically, we fine-tuned a CodeBERT [23] model so that it can classify the input code diff into a specific type in the range of ten angular types. Here we provide the detailed fine-tuned configuration and process:

CodeBERT is a pre-trained model proposed by Microsoft [23]. The hyperparameters for CodeBERT are set as follows:

- base_model: codebert-base⁵
- Block_size: 256
- Num_train_epochs: 5
- warmup_steps: 0
- learning_rate: 0.00002
- batch_size: 8
- adam_epsilon: 0.00000001
- max_grad_norm: 1.0
- weight_decay: 0.0
- seed: 123456

And the environment that we used in finetune process is:

- GPU: RTX3070
- PyTorch: 2.2.2

we transformed the type into the label from 0 to 9, each number corresponds to a type. So for the training data, the format is like this:

```
1 {"code": "diff --git a/scss/_alert.scss b/scss/_alert.scss // Generate contextual modifier classes for colorizing the alert.\n@each $state, $map in $alert-variants {\n- $background: var(--#{$variable-prefix}alert-bg, map-get($map, \"background\"));\n+ $background: var(--#{$variable-prefix}alert-background, map-get($map, \"background\"));\n$border: var(--#{$variable-prefix}alert-border-
```

⁵<https://huggingface.co/microsoft/codebert-base>

```

color, map-get($map, \"border\");\n$color: var(--#{ $variable-
prefix}alert-color, contrast-ratio-correction(map-get($map, \"color
\"), map-get($map, \"background\"), abs($alert-color-scale), $state
));\n$link-color: var(--#{ $variable-prefix}alert-link-color, map-
get($map, \"link-color\");\n\",
"label": 6}

```

For the data during model training, we split the database part of the original data set into a train set and a valid set in a ratio of 9:1 and then used the test data in the original data set as the test set.

For the chunking module and embedding module we reused the design in section 4.3.4. For the retrieval module, we split the original database according to these 10 angular types, and each sub-database only stores one of the angular types. So when we get the classification result of a code diff, the system will find the sub-database of the corresponding type to query the most relevant code diff and output it.

For the generation module, the similar code diff with the corresponding message will be combined with the input code diff to form a new LLM prompt, which is finally input into LLM and returns the corresponding message output. The following is the detailed process of prompt combination.

Prompt Design: For the ARC-PR prompt, we performed the prompt with the template that we used in 4.3.6.

4.3.8. Metric Evaluation And Analysis

To validate H.2.3 and H.2.4, we provide a more detailed definition and explanation of the metrics used for evaluation. In this section, we first introduce the five metrics that we use in our experiments, along with some adjustments we have made. Based on this, we present our method for metric analysis. We explain the specific meaning represented by each metric's calculation method so that these analytical approaches can be applied in the subsequent analysis of experimental results.

Metric Evaluation

Table 4.5 shows some NL (natural language) numeric evaluation metrics that we will use in the experiment.

Metrics
BLEU ⁶
B-Norm [77]
Rouge-L ⁷
Meteor ⁸
Log-MNEXT [13]

Table 4.5: Evaluation Metrics for the RQ2 Experiment

Log-MNEXT For Log-MNEXT we reused the script provided by [13], while the same core computing components are retained, we made some minor adjustments to the data preprocessing to make its logic more reasonable. For the previous preprocessing script, the commit message "fix(release): need to depend on latest rxjs and zone.js" will be processed as "fixrelease need to depend on latest rxjs and zonejs". Obviously "fixrelease" is not a word and has no real meaning. So we improved it by replaced the punctuation with a space, so after our processing script the processed message would be like "fix release need to depend on latest rxjs and zone js".

Metric Analysis

BLEU BLEU is very sensitive to the length of the generated message in relation to the reference message, especially when the length of the generated message is much shorter than the reference message. In such cases, the brevity penalty will be less than 1, which significantly reduces the BLEU score. Therefore, during the result analysis phase, we will focus on the relationship between the average length of the messages generated by each method and the average length of the reference messages.

BLEU-Norm Compared to BLEU, BLEU-Norm is less sensitive to the case of words (as all words in both reference and prediction are converted to lowercase). It also includes a smoothing method that increases the precision of some n-grams that would otherwise be zero. Consequently, the final score obtained using BLEU-Norm is generally higher than that of BLEU.

Rouge-L Rouge-L balances the evaluation of precision and recall because it uses the Longest Common Subsequence (LCS) to compute these metrics. As a result, it is sensitive to the order of information matching the reference, favoring information that appears in the same sequence as in the reference. Therefore, achieving a high score on the Rouge-L metric indicates that the generated message's information sequence is closer to that of the reference, and it also reflects the accuracy of information matching to some extent.

Meteor Meteor and Rouge-L have some similarities: both utilize the F1 Score. While Rouge-L balances recall and precision, Meteor places more emphasis on precision, which is reflected in the greater alpha weight given to precision in the F Score. Additionally, Meteor introduces a fragmentation penalty to account for gaps and differences in word order. When a method achieves a higher Meteor score compared to others, it indicates that this method's precision is superior, meaning it includes more information matching the reference in the generated message. In other words, for the same length, this method contains the highest amount of accurate information.

What sets Meteor apart is its concern with avoiding very "fragmented" translations. For example, if the reference translation is "A B C D" and the model-generated translation is "B A D C," while each unigram matches, the translation would receive a significant penalty due to the disruption in word order.

Log-MNEXT : One significant difference between Log-MNEXT and Meteor is: Log-MNEXT performs string lower-casing and removes punctuation as a preprocessing step before searching for word matches or performing word alignment between the predicted and reference texts. In contrast, because the reference message in angular format is in the format type(scope): subject before preprocessing, Meteor tokenizes the colon and parentheses into separate tokens, while Log-MNEXT does not. As a result, Log-MNEXT places less emphasis on the format to some extent.

For the BLEU and BLEU-Norm metrics, we will delve into the underlying n-gram precisions and brevity penalties to compare the phrase-matching accuracy and message lengths across different CMG methods. For the Rouge-L metric, we will investigate the average length of the longest common subsequences and the average message length to assess the density and total amount of effective information provided by each CMG method. For the METEOR and Log-MNEXT metrics, we will analyze the fragment penalty and average F1 score to evaluate the message format consistency and the balance between precision and recall for each CMG method. Additionally, in this metric evaluation section, we link expressiveness to message format consistency and the balance between precision and recall, while associating informativeness with phrase-matching accuracy, total effective message length, and effective message density.

4.3.9. Human Evaluation

To address RQ3 and validate H.3.1, we designed and conducted a human evaluation comparing our proposed method with the SOTA method RACE in the CMG domain.

We randomly sampled 50 commits from testing commits and we shuffled the order of all candidate messages.

Since the original code diff format 4.1 has poor readability, we used ChatGPT to reformat the original code diff, shown in 4.2.

Listing 4.1: Code diff before reformatting

```
diff --git a/src/common/reducers/layout/index.js b/src/common/reducers
/layout/index.js @@ -43,17 +43,10 @@ export function layout (state:
  State = initialState, action: Action): State {\n}\nswitch (action.
type) {\ncase APPLICATION_INIT:\n- case UI_WINDOW_RESIZE: {\n-
const {innerWidth} = action.payload\n- const {isMobile, isMobileSM,
isMobileXS} = computeMobileStatuses(\n- innerWidth\n- )\n+ case
```

```

UI_WINDOW_RESIZE:\nreturn {\n...state,\n- isMobile,\n- isMobileSM,\n
n- isMobileXS\n- }\n+ ...computeMobileStatuses(action.payload.
innerWidth)\n}\nncase UI_OPEN_SIDEBAR:\nreturn {\n

```

Listing 4.2: Code diff after reformatting

```

1 diff --git a/src/common/reducers/layout/index.js b/src/common/reducers
2 /layout/index.js
3 @@ -43,17 +43,10 @@ export function layout (state: State =
4 initialState, action: Action): State {
5 switch (action.type) {
6 case APPLICATION_INIT:
7 - case UI_WINDOW_RESIZE: {
8 -   const { innerWidth } = action.payload
9 -   const { isMobile, isMobileSM, isMobileXS } =
10 computeMobileStatuses(
11 -   innerWidth
12 -   )
13 + case UI_WINDOW_RESIZE:
14   return {
15     ...state,
16     isMobile,
17     isMobileSM,
18     isMobileXS
19   }
20 +   ...computeMobileStatuses(action.payload.innerWidth)
21   }
22 case UI_OPEN_SIDEBAR:
23   return {

```

For this experiment we shuffled all messages that generated by the ARC-PR method and the RACE method for each commit, the entire dataset that needs to be evaluated by human beings is listed below.

- Highly readable code difference snippet in the .diff files
- Commit message generated by the RACE method.
- Commit message generated by the ARC-PR method.

Listing A.1 shows an example data in this human evaluation dataset.

In this test, the types of all sample data are:

type	count	type	count
docs	10	chore	5
feat	9	style	1
fix	7	test	5
build	1	refactor	8
ci	4		

Table 4.6: Distribution of Types in Human Evaluation Experiment

We define criteria in two aspects for manual labelling as shown in Table 4.7 which shows the meaning of scores and is used to guide the raters to score following previous works [76].

Expressiveness	Description
0	Cannot read and understand.
1	Is hard to read and understand.
2	Is somewhat readable and understandable.
3	Is mostly readable and understandable.
4	Is easy to read and understand.

Informativeness	Description
0	No information about the code change is provided.
1	Some crucial information is missing, making it hard to understand the code changes.
2	Some information is missing, but the missing parts are not essential for understanding the code changes.
3	There is some missing information, but none of it is necessary to understand the code changes.
4	All necessary information about the code change is included.

Table 4.7: Human Evaluation Scoring Criteria

To verify the agreement of all raters, we will calculate Kendall rank correlation coefficient (Kendall's Tau) values [40].

By summarizing the rating information from all participants, we can obtain the average scores and standard deviations for expressiveness and informativeness for both RACE and our method. Additionally, we use Wilcoxon signed-rank tests [88] to evaluate the extent of differences between the two methods in terms of expressiveness and informativeness.

5

Result

This chapter presents the experimental results that address our research questions. It is divided into three sections, each corresponding to a specific research question (Sections 5.1, 5.2, and 5.3).

In Section 5.1, we provide a detailed analysis of the commit message data generated by three models, including LLM (ChatGPT)-zero-shot, across various natural language (NL) indicators.

Section 5.2 presents a series of validation experiments investigating the impact of different configurations on the generation performance of the RAG system. This section also examines how the composition of research demonstrations affects the performance of LLM RAG methods. We compare our proposed prompt engineering method with previous methods and state-of-the-art (SOTA) techniques, thoroughly addressing RQ2.

Finally, in Section 5.3, we report on the human evaluation experiment and use statistical methods to analyze the experimental results in detail.

5.1. Experimental Results For RQ1

The goal of this first set of experiments is to either corroborate or contradict H.1.1 as an answer to RQ1. We use ChatGPT as our LLM model and use zero-shot prompt to generate the commit message for the testing data. At the same time, we also use the previous retrieval-based SOTA method NNGEN and the hybrid SOTA method RACE as our baseline for comparison.

In terms of the test data, we use 3.5% commits collected by stratified sampling in the entire collected dataset as our test data set.

The results are summarized in **Table 5.1**. These show the scores of the three commit message generation methods mentioned above on the five commonly used CMG task metrics on the test dataset.

Pred Model	BLEU	B-Norm	Rouge-L	Meteor	log-MNEXT
LLM zero-shot	2.38	3.18	21.63	13.39	21.09
NNGen	2.49	2.92	9.99	18.89	6.91
RACE	2.98	3.63	26.39	17.24	19.57

Table 5.1: Comparison of the LLM Zero-Shot Method and Previous CMG Methods Across Five Metrics

As shown in **Table 5.1**, for log-MNEXT, zero-shot LLM achieves the highest score and NNGEN achieves the lowest score. However, the results are the opposite for the Meteor metric, where NNGEN achieves the highest score and LLM achieves the lowest score. This is mainly due to the difference between log-MNEXT and Meteor. The main difference between log-MNEXT and Meteor is that log-MNEXT has a preprocessing method, which removes all punctuation.

In the calculations for Meteor and Log-MNEXT, the most important components are the F1 score and Fragment Penalty. Therefore, we collected and compiled the average F1 score and average Fragment Penalty for all three methods. The results are shown in the table below:

	Meteor		Log-MNEXT	
	Average Frag Penalty	Average F1	Average Frag Penalty	Average F1
NNGen	0.353	0.268	0.252	0.123
LLM zero-shot	0.332	0.208	0.304	0.309
RACE	0.321	0.248	0.292	0.282

Table 5.2: Comparison of Meteor and Log-MNEXT Metrics for the LLM Zero-Shot Method and Existing Methods

From **Table 5.2**, we observe that both NNGen and LLM zero-shot methods show a decrease in Fragment Penalty on Log-MNEXT. However, regarding the F1 score, NNGen’s score drops significantly when transitioning from Meteor to Log-MNEXT, while LLM zero-shot’s F1 score actually increases. This is because NNGen-generated content often exhibits fragmented content matching (mostly single-word matches). However, due to symbols, the format of the generated message resembles that of the reference message, resulting in high character matching scores on the Meteor metric. On the other hand, Log-MNEXT removes the influence of symbols during preprocessing, causing NNGen’s F1 score to drop considerably, leading to much lower scores on Log-MNEXT compared to the zero-shot LLM method. Although LLM zero-shot performs better on the Log-MNEXT metric, its lower score on Meteor and higher score on Log-MNEXT also indicate that LLM lacks the ability to generate normalized messages in the angular format dataset, which is why its score on the Meteor metric is the lowest. In addition, we can observe that the average F1 score and average Fragment Penalty for the LLM zero-shot method and the RACE method on the Log-MNEXT metric are very close, indicating that after character processing, the LLM zero-shot method and the RACE method perform similarly in terms of the balance between precision and recall. However, on Meteor, the RACE method’s average F1 score surpasses that of the LLM zero-shot method, suggesting that before character processing, the RACE method performs better in balancing precision and recall. Overall, the RACE method demonstrates the best performance in terms of balancing precision and recall.

From **Table 5.1**, we can also observe that NNGen scores higher than LLM zero-shot on BLEU but lower on BLEU-Norm. The main difference between BLEU and BLEU-Norm is that BLEU-Norm is insensitive to word cases and includes a smoothing method. We tested both metrics separately, as shown in **Table 5.3**.

	BLEU-Norm(Lowercase + Smooth)	BLEU(Lowercase)	BLEU(Smooth)	BLEU
NNGen	2.92	2.90	2.50	2.49
LLM	3.18	3.16	2.39	2.38

Table 5.3: Comparison of NNGen and LLM Zero-Shot on BLEU Score with and Without Lowercasing and Smoothing

The results in **Table 5.3** indicate that the case-insensitivity of BLEU-Norm is the reason why LLM zero-shot scores higher than NNGen on BLEU-Norm. From a reader’s perspective, case sensitivity does not affect their understanding of the content, so BLEU-Norm provides a more complete reflection of the information in the current context compared to BLEU. The higher score of LLM zero-shot on BLEU-Norm compared to NNGen suggests that its character-matching accuracy is superior to that of retrieval-based methods.

Additionally, we found that RACE achieved the highest scores on both BLEU and BLEU-Norm. To further analyze and compare the n-gram matching accuracy of these three methods, we calculated the average brevity penalty and n-gram precisions for each method, as shown in the following table:

	Average BP	1-gram	2-gram	3-gram	4-gram	BLEU
LLM zero-shot	0.93	0.162	0.036	0.014	0.005	2.38
NNGen	0.93	0.232	0.064	0.008	0.004	2.49
RACE	0.35	0.349	0.111	0.047	0.027	2.98

Table 5.4: Comparison of Average Brevity Penalty and BLEU Precision Scores for LLM Zero-Shot and Existing Methods

From **Table 5.4**, we can observe that although the RACE method has a very low brevity penalty, indicating that the generated messages are much shorter compared to the reference messages, its n-gram precisions are higher than those of the other two methods. This suggests that RACE achieves better phrase-matching accuracy, which is why it still scores the highest on both BLEU and BLEU-Norm.

On Rouge-L, RACE scores are higher than those of zero-shot LLM, which could be due to several factors, primarily the length of common subsequence strings and the length of the generated messages. For example, given the same length of generated messages, RACE might have longer common subsequence strings than zero-shot LLM; alternatively, for the same length of common subsequence strings, RACE might have shorter generated messages than zero-shot LLM. Therefore, we calculated the average LCS and the average length of generated messages for both zero-shot LLM and RACE. The statistical results are as follows:

	Average LCS	Average message length
LLM zero-shot	1.88	10.22
RACE	1.64	5.35
NNGen	0.73	10.19

Table 5.5: Comparison of Average LCS and Message Length for Zero-Shot LLM and Existing Methods

From **Table 5.5**, we can see that the average length of messages generated by the NNGen method is similar to that of the zero-shot LLM method, but the average LCS of zero-shot LLM is much higher than that of NNGen. As a result, NNGen has the lowest Rouge-L score. In contrast, the average LCS of the RACE method is close to that of the zero-shot LLM method, but the average length of messages generated by RACE is only half that of zero-shot LLM. In summary, the proportion of information contained in a message per unit length that matches the reference message is higher for RACE compared to zero-shot LLM, which is why RACE has the highest Rouge-L score. However, we can also observe that the average LCS length of zero-shot LLM is still slightly higher than that of RACE, indicating that zero-shot LLM retains a certain level of informativeness compared to RACE. This highlights one of the research potentials of LLMs.

Conclusion: The LLM zero-shot method doesn't fully tap into the potential of LLMs, resulting in performance that falls short in terms of expressiveness and informativeness compared to state-of-the-art methods.

5.2. Experimental Results For RQ2

This set of experiments aims to answer RQ2 by corroborating or contradicting three hypotheses H.2.1, H.2.2, H.2.3. For RQ2 we focus primarily on comparing our proposed method with the LLM zero-shot method, the LLM RAG method, and the SOTA method RACE. Therefore, the subsequent evaluation of experimental results will exclusively involve these three methods.

To verify H.2.1, we set up multiple sets of RAG system configuration parameters for experiments and conducted detailed analyses.

To verify H.2.2, we use the conclusions obtained in H.2.1 as the LLM RAG experimental configuration and our proposed method's configuration. We do the validation test to fully verify the impact of the most relevant code diff retrieved on LLM generation and showcase the retrieval ability between the LLM RAG method and our proposed method. Following, we compare our proposed method with LLM RAG and LLM zero-shot method on the angular format dataset. Additionally, we validate the robustness

and generalizability of the proposed method by testing these three methods on a new unified message format dataset.

To verify H.2.3, we evaluated our method with RACE in various aspects of effectiveness by comparing five numerical metrics and performing analysis.

5.2.1. The Impact Of Chunking Method And Embedding Method To LLM RAG

In this section, we will investigate the impact of different chunking sizes and different embedding models on the generated results of the RAG system.

In this part, we randomly sampled 500 commits from the test dataset as validation data for this experiment.

First, we explored the influence of different chunking sizes while controlling other configuration variables. **Table 5.6** Shows the scores of log-MNEXT and B-Norm for the generated results under all different chunking size configurations.

Pred Model	BLEU	B-Norm	Rouge-L	Meteor	Log-MNEXT
500 chunk	5.42	6.36	22.61	29.60	22.27
1000 chunk	6.01	7.21	22.96	30.32	22.36
2500 chunk	5.38	6.64	22.57	29.67	22.32
No chunk	5.47	6.60	22.89	29.67	22.38

Table 5.6: Comparison of Different Chunking Size Settings Across Five Metrics

From **Table 5.6**, we found that, except for the Log-MNEXT metric, the 1000-chunk option shows a significant advantage across the other four metrics among all chunking size options. Specifically, for the BLEU and BLEU-Norm metrics, the 1000-chunk option outperforms the other three options. We compared the n-gram precisions and brevity penalties of these four methods, as shown in the table below.

	Average BP	1-gram	2-gram	3-gram	4-gram	BLEU
500 chunk	1.0	0.24	0.078	0.03	0.015	5.42
1000 chunk	1.0	0.257	0.087	0.034	0.017	6.01
2500 chunk	1.0	0.249	0.08	0.03	0.014	5.38
No chunk	1.0	0.251	0.081	0.031	0.014	5.47

Table 5.7: Comparison of Average Brevity Penalty and BLEU Precision Scores Across Different Chunking Size Settings in LLM RAG Methods

From **Table 5.7**, we found that the n-gram precisions with the 1000-chunk setting are higher than those of the other three settings. Therefore, we conclude that the 1000-chunk setting is the best option for n-gram precisions among the various configurations of the LLM RAG method.

From **Table 5.6**, we can see that the rouge-L score is the highest under the 1000-chunk setting. We compared the average LCS and the average message length across these four settings, as shown in the table below.

	Average LCS	Average message length
500 chunk	2.17	15.30
1000 chunk	2.16	14.36
2500 chunk	2.16	14.64
No chunk	2.18	14.66

Table 5.8: Comparison of Average LCS and Message Length for LCS and Average Message Length Across Different Chunking Size Settings in LLM RAG Methods

From **Table 5.8**, we can see that although the 1000-chunk setting does not achieve the highest LCS, it has the shortest average message length, indicating that the 1000-chunk setting has a higher effective message density. Therefore, we conclude that the 1000-chunk setting has the highest effective message density among all the options.

From **Table 5.6**, we can see that the 1000-chunk setting outperforms the other three settings in the Meteor metric, while it is slightly lower than the No chunk setting in the Log-MNEXT metric. We compared the average fragment penalty and average F1 score for these four methods in the Meteor and Log-MNEXT metrics, as shown in the table below.

	Meteor		Log-MNEXT	
	Average FP	Average F1	Average FP	Average F1
500 chunk	0.304	0.396	0.330	0.335
1000 chunk	0.297	0.402	0.325	0.335
2500 chunk	0.302	0.396	0.326	0.333
No chunk	0.304	0.397	0.326	0.334

Table 5.9: Comparison of Meteor and Log-MNEXT Metrics Across Different Chunking Size Settings in LLM RAG Methods

From **Table 5.9**, we can observe that among all the chunking size options in Meteor metric, the 1000-chunk setting has the lowest average fragment penalty and the highest average F1 score. In contrast, the scores for all methods in the Log-MNEXT metric are very close and can be considered as performing similarly. Therefore, we conclude that the 1000-chunk option offers the best balance between precision and recall, as well as message format consistency, among all the chunking size options.

Secondly, under the same conditions of controlling other configuration variables, we explored the influence of different embedding models, **Table 5.10** Shows the scores of BLEU, BLEU-Norm, Rouge-L, Meteor and Log-MNEXT for the generated results of the RAG system under the embedding model configuration with different parameter sizes.

Pred Model	BLEU	B-Norm	Rouge-L	Meteor	Log-MNEXT
e5-small-v2	5.47	6.60	22.89	29.67	22.38
mxbai-embed-large-v1	5.53	6.44	22.73	29.64	22.44
all-MiniLM-L6-v2	5.69	6.74	22.55	29.42	22.20

Table 5.10: Performance Metrics Across Different Embedding Model Settings

From the **Table 5.10**, we found that although different methods each have their own advantages across the five metrics, the differences are quite close. Therefore, we further explored these three embedding models by investigating aspects related to message generation effectiveness behind these five metrics. From the analysis in the three **Tables 5.11, 5.12, 5.13**, we observed that the three embedding models are quite similar in terms of n-gram precision, effective information density, and the balance between precision and recall. Thus, we conclude that different parameter sizes of embedding models have a limited impact on the RAG system. As a result, in future experimental setups, we will select the e5-small-v2 embedding model as part of the configuration for the optimal RAG pipeline.

	Average BP	1-gram	2-gram	3-gram	4-gram	BLEU
e5-small-v2	1	0.251	0.081	0.031	0.014	5.47
mxbai-embed-large-v1	1.0	0.245	0.080	0.031	0.015	5.53
all-MiniLM-L6-v2	1.0	0.254	0.083	0.032	0.016	5.69

Table 5.11: Comparison of Average Brevity Penalty and BLEU Precision Scores Across Different Embedding Models in LLM RAG Methods

	Average LCS	Average message length
e5-small-v2	2.18	14.66
mxbai-embed-large-v1	2.17	14.93
all-MiniLM-L6-v2	2.15	14.61

Table 5.12: Comparison of Average LCS and Message Length Across Different Embedding Model Settings in LLM RAG Methods

	Meteor		Log-MNEXT	
	Average FP	Average F1	Average FP	Average F1
e5-small-v2	0.304	0.397	0.326	0.334
mxbai-embed-large-v1	0.304	0.397	0.322	0.335
all-MiniLM-L6-v2	0.308	0.396	0.327	0.334

Table 5.13: Comparison of Meteor and Log-MNEXT Metrics Across Different Embedding Model Settings in LLM RAG Methods

Conclusion: Our findings indicate that in the task of commit message generation, the embedding model has minimal effect on the RAG system’s output while the chunking size has a certain effect on the RAG system’s output. Therefore, the choice of configuration parameters of the embedding model can be considered negligible for subsequent experiments and for the chunking size we consider using 1000 as the chunking size.

5.2.2. Evaluating The Hypothesis: Enhancing Model Generation Quality By Good Demonstration

In this section, we design a validation experiment to discuss whether providing good demonstrated similar code diffs (with the same commit type as the input code diff) can effectively improve the quality of messages generated by RAG, and we will compare it with the original RAG system on several indicators to verify our hypothesis.

In this experiment, we used the commit type in the test dataset as the predictor condition and input it into the RAG system. At the same time, we reconstructed 10 sub-databases of commit type to retrieve similar code diffs. That is, for the input code diff, we retrieved similar code diffs that are consistent with its commit type from the sub-database as part of the prompt. At the same time, we also explicitly put the commit type into the prompt to let LLM perform commit message generation.

The method involved in this experiment does not really exist, because we cannot accurately know the commit type of the input code diff in advance without giving its commit message. This experiment is only used to verify whether such a method can improve the generation effectiveness of the existing RAG system if it exists so that we can carry out further method design in the future to approach this ideal method.

Table 5.14 shows the generation results of the RAG method under the ideal conditions mentioned above and the scores of the current RAG method on several metrics.

Pred Model	BLEU	B-Norm	Rouge-L	Meteor	Log-MNEXT
LLM RAG with golden_matching	8.29	9.95	29.00	34.84	25.28
LLM RAG without golden_matching	6.01	7.21	22.96	30.32	22.36

Table 5.14: Comparison of LLM RAG with golden matching and without golden matching

From **Table 5.14**, we can intuitively see that the ideal RAG method has a significant improvement over the existing RAG method in all metrics.

For BLEU and BLEU-Norm, we compare these two methods with their brevity penalty and n-gram precisions, as shown in the table below:

	Average BP	1-gram	2-gram	3-gram	4-gram	BLEU
RAG with golden matching	1	0.318	0.118	0.045	0.025	8.05
RAG without golden matching	1	0.251	0.081	0.031	0.014	5.47

Table 5.15: Comparison of Average Brevity Penalty and BLEU Precision Scores for LLM RAG with and without Golden Matching

From **Table 5.15**, we can clearly see that the LLM RAG with golden matching method shows a significant improvement in every n-gram precision metric compared to the LLM RAG without golden matching. This indicates that providing additional type-matching information enables more effective matching of similar commits, thereby allowing the LLM to generate more accurate commit messages.

In terms of the Rouge-L metric, we also observe a significant score improvement with the LLM RAG with the golden matching method compared to the LLM RAG without the golden matching method. Therefore, we further compared the average LCS and message length between these two methods, with the results shown in the table below:

	Average LCS	Average message length
RAG with golden matching	2.50	12.13
RAG without golden matching	2.18	14.66

Table 5.16: Comparison of Average LCS and Message Length for LLM RAG with and without Golden Matching

From **Table 5.16**, we can see that the LLM RAG with the golden matching method contains longer common subsequences with reference messages in shorter commit messages compared to the LLM RAG without the golden matching method. In other words, it includes a higher proportion of effective information per unit length of the commit message. This explains why it performs significantly better than the LLM RAG without the golden matching method on the Rouge-L metric. This indicates that the LLM RAG with the golden matching method can generate commit messages with more effective information.

Finally, we observe a significant score improvement for the LLM RAG with the golden matching method over the LLM RAG without the golden matching method on the METEOR and Log-MNEXT metrics. To further assess its performance in message format consistency as well as its accuracy and recall, we have calculated the average F1 scores and average Fragment Penalty for both methods on these metrics. The results are shown in the table below:

	Meteor		Log-MNEXT	
	Average FP	Average F1	Average FP	Average F1
RAG with golden matching	0.266	0.444	0.338	0.390
RAG without golden matching	0.297	0.402	0.325	0.335

Table 5.17: Comparison of Meteor and Log-MNEXT Metrics for LLM RAG with and without Golden Matching

Where Average FP stands for Average Fragment Penalty, Average F1 stands for Average F1 score.

From **Table 5.17**, we can see that on the METEOR metric, the LLM RAG with the golden matching method has a reduced Fragment Penalty compared to the LLM RAG without the golden matching method. Additionally, the proportion of commit messages generated in angular format with the LLM RAG with the golden matching method increased from 97.84% to 98.55%, indicating a further improvement in commit message format consistency. Moreover, the average F1 score for the LLM RAG with the golden matching method is higher than that of the LLM RAG without the golden matching method, which further demonstrates an improvement in both accuracy and recall of the generated messages.

On the Log-MNEXT metric, the Fragment Penalty for the LLM RAG with the golden matching method remains almost unchanged compared to the LLM RAG without the golden matching method. The improvement in the average F1 score is the reason for the higher final Log-MNEXT score of the LLM

RAG with the golden matching method over the previous RAG method. This also confirms a significant improvement in accuracy and recall with the LLM RAG with the golden matching method.

Conclusion: Our validation experiments highlight the positive impact of precise matching of angular types on the effectiveness of LLM RAG commit message generation. A demonstration of the commit that has the same commit type as the input commit can be a good demonstration of the LLM RAG method. Increasing the number of good demonstrations in the LLM RAG method can significantly improve the effectiveness of the LLM RAG method.

5.2.3. Evaluate The Matching Accuracy Of The LLM RAG Method And ARC-PR Method

In this section, we first evaluate the matching accuracy of the LLM RAG method and use the confusion matrix method to intuitively show its shortcomings. Then we also evaluate the matching accuracy of the ARC-PR method and analyze its advantages and disadvantages using the confusion matrix method.

First, we compared the commit type corresponding to the similar code diff retrieved by the RAG method with the commit type corresponding to the target code diff. The results showed that the matching accuracy was only 26.57%. **Figure 5.1** is the confusion matrix analysis diagram of the similar code diff result retrieved by the original RAG method.

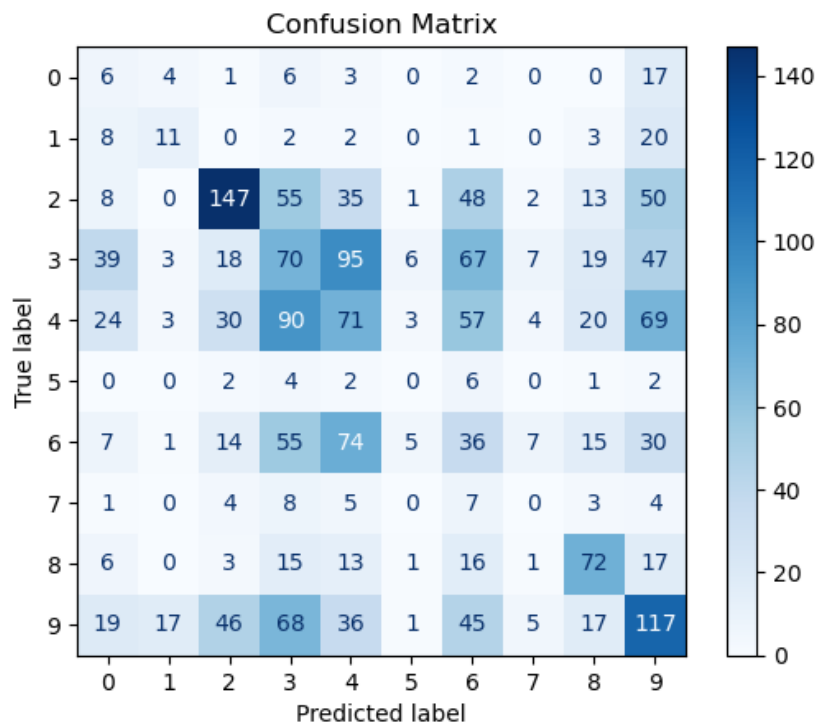


Figure 5.1: Confusion Matrix for the LLM RAG Method

Firstly, the above matrix shows a relatively strong diagonal presence, especially for classes like 2 and 9, indicating that the model is able to correctly predict the commit type for these classes most of the time. For example, class 2 has 159 correct predictions, and class 9 has 125 correct predictions. Conversely, the model struggles with classes like 3 and 6, which have more dispersed predictions across multiple other classes. While the model demonstrates good performance for some classes, the spread of misclassifications for others indicates there is still room for improvement.

In summary, while the original RAG LLM shows strong performance in correctly predicting certain commit types, particularly classes 2 and 9, it also reveals areas where the model struggles, particularly with classes 3 and 6. From the perspective of overall prediction accuracy, the accuracy of the original RAG method retrieval is not high enough.

Next, we tested the matching accuracy of the newly proposed classification module on the test set, and we found that the accuracy was 42.61%, which is about 58% higher than the previous RAG method. **Figure 5.2** is the confusion matrix analysis diagram of the similar code diff result retrieved by our proposed classification method.

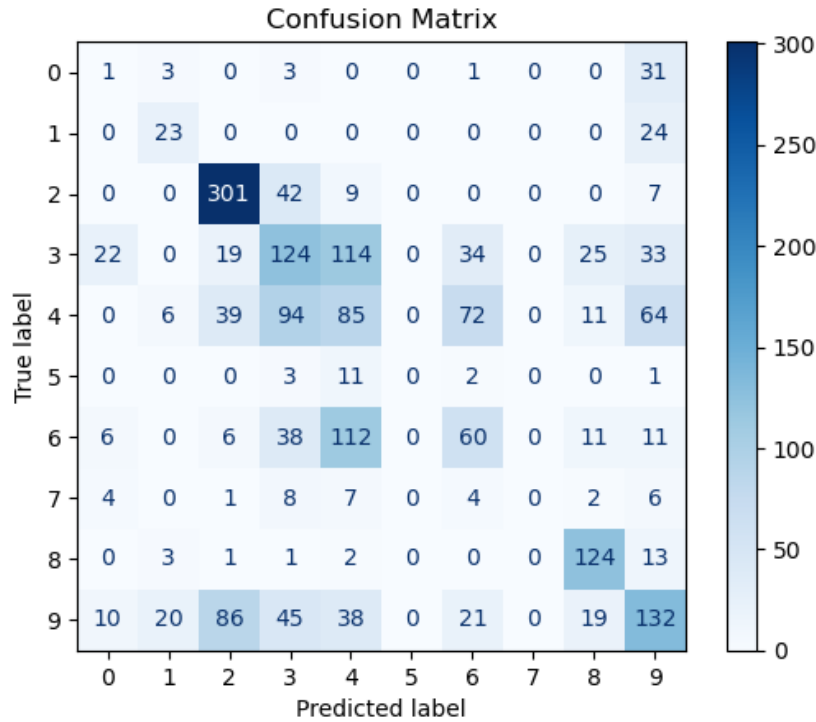


Figure 5.2: Confusion Matrix for the ARC-PR Method

The above matrix again shows a relatively strong diagonal, indicating that the model performs well in predicting certain commit types correctly. For example, class 2 has 301 correct predictions, and class 9 has 132 correct predictions. Compared with the original RAG method's confusion matrix, our proposed method's confusion matrix shows a notable improvement in the model's accuracy and performance compared to the first matrix. There is a stronger diagonal dominance and fewer scattered misclassifications, suggesting that the model has improved in predicting the correct commit types.

Misclassifications are reduced, particularly for classes that previously showed significant dispersion. However, there are still areas for improvement, especially in classes like 0 and 8, which continue to show some level of confusion.

The enhanced performance in the second confusion matrix indicates that improvements in the model impacted its ability to predict commit types accurately.

Conclusion: In the Angular format dataset, our proposed method achieves a type-matching accuracy for retrieved similar commit messages that is higher than that of the LLM RAG method, improving by nearly 58%.

5.2.4. Comparative Analysis of ARC-PR With LLM RAG And LLM Zero-Shot Approaches

To verify the effectiveness of our proposed method in the commit message generation task compared to the previous LLM-based methods, we tested these three methods on the angular format dataset. **Table 5.18** shows the score comparison of our method with the original RAG method and the SOTA method RACE in various metrics.

Pred Model	BLEU	B-Norm	Rouge-L	Meteor	log-MNEXT
LLM zero-shot	2.16	2.87	21.24	12.84	21.09
LLM RAG	6.01	7.21	22.96	30.32	22.36
ARC-PR	8.02	9.60	28.68	34.27	25.27

Table 5.18: Comparison of ARC-PR method and previous LLM-based methods Across Five Metrics

From **Table 5.18**, we can observe that our proposed method shows further improvement over the original RAG method and shows significant improvements over the LLM zero-shot method in terms of BLEU and BLEU-Norm metrics. To validate the superior n-gram matching capabilities of our proposed method, we have statistically analyzed and compared the Brevity Penalty and n-gram precisions for these methods. The results are shown in the table below:

	Average BP	1-gram	2-gram	3-gram	4-gram	BLEU
LLM zero-shot	0.93	0.162	0.036	0.014	0.005	2.38
LLM RAG	1.0	0.257	0.087	0.034	0.017	6.01
ARC-PR	1	0.323	0.119	0.044	0.025	8.02

Table 5.19: Comparison of Average Brevity Penalty and BLEU Precision Scores for the ARC-PR Method, LLM RAG, and LLM Zero-Shot Method

From **Table 5.19**, we can clearly see that the ARC-PR method shows improvements in each n-gram precision metric compared to the LLM RAG method and LLM zero-shot method. This indicates that the similar commits queried by our proposed method have greater informational value than those queried by the LLM RAG method, thereby enhancing its n-gram matching capabilities.

On the Rouge-L metric, we observe a significant score improvement with the ARC-PR method compared to the LLM RAG method and the LLM zero-shot method. To further assess the amount of information contained in messages of unit length, we have calculated the average LCS and average message length for the three methods listed in the table. The results are shown in the table below:

	Average LCS	Average message length
LLM zero-shot	1.88	10.22
LLM RAG	2.18	14.66
ARC-PR	2.49	12.28

Table 5.20: Comparison of Average LCS and Message Length for the ARC-PR Method, LLM RAG, and LLM Zero-Shot Method

From **Table 5.20**, we can observe that our proposed method shows an improvement in average LCS compared to the LLM RAG method, and it reduces the average message length by about 16%. This results in a higher proportion of effective information per unit length of the message, which contributes to an improvement in the final Rouge-L score. Furthermore, compared to the LLM zero-shot method, our approach has both a longer LCS and a shorter average message length, indicating that our method contains more effective information overall and has a higher density of effective information. In summary, the ARC-PR method improves the density of effective information compared to the LLM RAG method and improves the overall effective information compared to the LLM zero-shot method.

Lastly, on the METEOR and Log-MNEXT metrics, our proposed ARC-PR method outperforms both the LLM RAG method and the LLM zero-shot method. To further evaluate the performance of our proposed method in terms of message format consistency as well as the balance between accuracy and recall, we have calculated the Average Fragment Penalty and Average F1 Score for the three methods on these metrics. The results are shown in the table below:

	Meteor		Log-MNEXT	
	Average FP	Average F1	Average FP	Average F1
LLM zero-shot	0.332	0.208	0.304	0.309
LLM RAG	0.297	0.402	0.325	0.335
ARC-PR	0.269	0.443	0.336	0.391

Table 5.21: Comparison of Meteor and Log-MNEXT Metrics for the ARC-PR Method, LLM RAG, and LLM Zero-Shot Method

From **Table 5.21**, we can see that on the METEOR metric, the ARC-PR method shows a reduction in Fragment Penalty compared to the LLM RAG method. Additionally, the proportion of angular format commit messages generated by the ARC-PR method increased from 97.84% to 98.44% compared to the LLM RAG method, indicating a further improvement in commit message format consistency. Moreover, the ARC-PR method achieves a higher average F1 score than the LLM RAG method and the LLM zero-shot method, demonstrating a balanced improvement in both accuracy and recall of the generated messages.

On the Log-MNEXT metric, the ARC-PR method shows almost no change in Fragment Penalty compared to the LLM RAG method. The slight improvement in the average F1 score is the reason for its higher final Log-MNEXT score compared to the LLM RAG method. Although the average Fragment Penalty of the ARC-PR method is higher than that of the LLM zero-shot method, its average F1 score is significantly better than that of the LLM zero-shot method. These all demonstrate that the ARC-PR method achieves a more balanced accuracy and recall compared to the previous LLM methods.

Conclusion: In the analysis of the results across five metrics, we find that our proposed ARC-PR method shows improvements over the LLM RAG method and the LLM zero-shot method in terms of effective information density, phrase-matching accuracy, message format consistency, and balanced accuracy and recall. It is the optimal method among these three LLM-based methods in terms of all aspects of effectiveness.

5.2.5. Comparative Analysis Of ARC-PR With SOTA Method RACE

Pred Model	BLEU	B-Norm	Rouge-L	Meteor	log-MNEXT
RACE	2.98	3.63	26.39	17.24	19.57
ARC-PR	8.02	9.6	28.68	34.27	25.27

Table 5.22: Comparison Of ARC-PR method And RACE Method Across Five Metrics

From **Table 5.22**, we can observe that our proposed method shows further improvement over the RACE method in terms of BLEU and BLEU-Norm metrics. To validate the superior n-gram matching capabilities of our proposed method, we have statistically analyzed and compared the Brevity Penalty and n-gram precisions for these methods. The results are shown in the table below:

	Average BP	1-gram	2-gram	3-gram	4-gram	BLEU
RACE	0.35	0.349	0.111	0.047	0.027	2.98
ARC-PR	1	0.323	0.119	0.044	0.025	8.02

Table 5.23: Comparison of the Average Brevity Penalty and Precisions in BLEU of the ARC-PR method with the RACE method

From **Table 5.23**, we can see that our method performs much better than RACE in terms of brevity penalty, indicating that the texts generated by our method are generally longer than the reference messages, while RACE tends to generate shorter texts. Moreover, our method's n-gram precision is very close to that of RACE, and it even surpasses RACE in 2-gram precision. Based on these observations, we conclude that our proposed method is comparable to the RACE method in terms of n-gram precision while generating longer texts.

On the ROUGE-L metric, we observed that our method achieves higher scores compared to the RACE method. To further assess the amount of information contained in messages of unit length, we have calculated the average LCS (Longest Common Subsequence) and average message length for the three methods listed in the table. The results are shown in the table below:

	Average LCS	Average message length
ARC-PR	2.49	12.28
RACE	1.64	5.35

Table 5.24: Comparison of Average LCS and Message Length Between the ARC-PR Method and the RACE Method

From **Table 5.24**, we can observe that our proposed method shows a greater Longest Common Subsequence (LCS) compared to the RACE method—an improvement of nearly 32%—indicating that our method contains more effective information overall. Our method also generates messages with a much longer average length, nearly 129% greater than that of the RACE method. Since the rate of increase in LCS exceeds the rate of increase in message length, this results in our method having a higher ROUGE-L score compared to RACE. This suggests that our method has a higher density of effective information.

Lastly, on the METEOR and Log-MNEXT metrics, our proposed method outperforms the RACE method. To further evaluate the performance of our proposed method in terms of message format consistency as well as the balance between accuracy and recall, we have calculated the Average Fragment Penalty and Average F1 Score for the two methods on these metrics. The results are shown in the table below:

	Meteor		Log-MNEXT	
	Average FP	Average F1	Average FP	Average F1
ARC-PR	0.269	0.443	0.336	0.391
RACE	0.321	0.248	0.292	0.282

Table 5.25: Comparison of Meteor and Log-MNEXT Metrics Between the ARC-PR Method and the RACE Method

From **Table 5.25**, we can see that on the METEOR metric, in comparison with the RACE method, the ARC-PR method exhibits a lower Average Fragment Penalty and a higher Average F1 score. This indicates that our proposed method not only generates messages that are more closely aligned with the angular format (with a proportion of angular format messages significantly higher than the 40.3% achieved by the RACE method) but also maintains a more balanced accuracy and recall.

On the Log-MNEXT metric, the ARC-PR method has a higher Average Fragment Penalty than the RACE method. From our previous analysis of Rouge-L, we know that the RACE method has a higher density of effective information. This suggests that the RACE method not only has a higher density of effective information but also better coherence of that information. However, the RACE method has a much lower Average F1 Score compared to our proposed method. This indicates that while our method performs better in terms of information matching accuracy and recall, the information becomes more fragmented as the message length increases, resulting in a higher Fragment Penalty compared to the RACE method.

Conclusion: Our method outperforms the RACE method in most aspects of message generation evaluation, and in other areas, it is on par with RACE. Therefore, our method is a better commit message generation method than the current state-of-the-art (SOTA) approach.

5.2.6. Generalizability Evaluation: VDO Format Dataset

In this section, we designed a series of experiments to verify that our proposed method is still robust on datasets of other formats compared to the LLM RAG method. We first analyze the matching accuracy and confusion matrix of the original RAG method, and then we analyze the matching accuracy and confusion matrix of our proposed method. Finally, we test the scores of various indicators on the designed test dataset and compare our method with LLM RAG and LLM zero-shot method.

We collected a dataset in a different format called VDO (Verb-Direct Object) format. We use the same stratified sampling method to calculate the proportion distribution of each type of data on the collected training data set, and then sample the commitchronicle test data set to obtain the VDO format test data set.

First, we compared the commit type corresponding to the similar code diff retrieved by the RAG method with the commit type corresponding to the target code diff. The results showed that the matching accuracy was only 14.16%. **Figure 5.3** is the confusion matrix analysis diagram of the similar code diff result retrieved by the original RAG method.

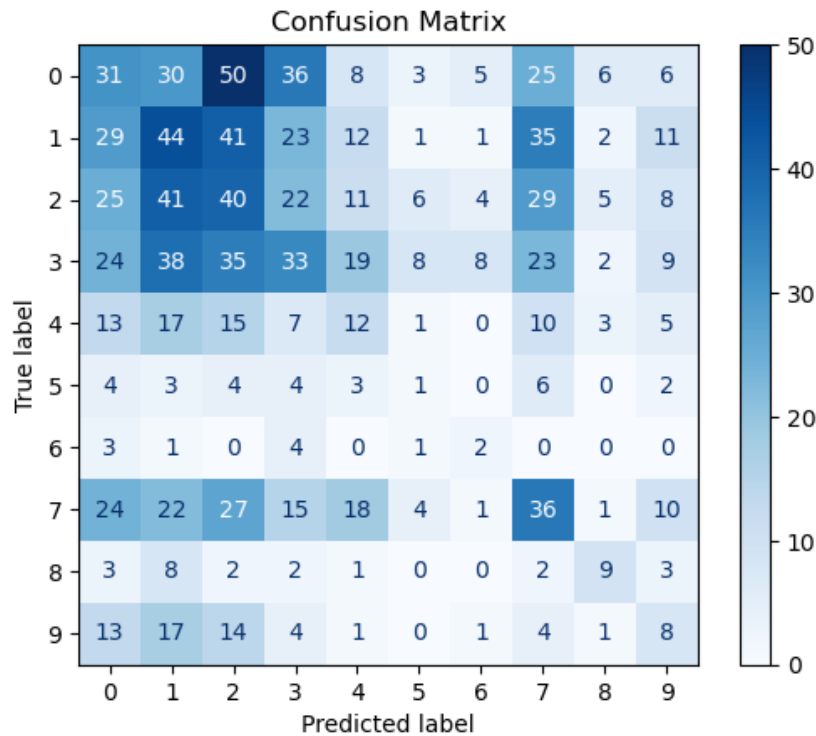


Figure 5.3: Confusion Matrix for the LLM RAG Method on the VDO Format Dataset

From the confusion matrix above, we can see that the correlation between the commit type of the similar code diff retrieved by the original RAG method and the commit type of the target code diff is very low, and the information value of the similar code diff retrieved by the original RAG method is obviously insufficient.

Next, we tested the matching accuracy of the newly proposed classification module on the test set, and we found that the accuracy was 28.86%, which is about 104% higher than the previous RAG method. **Figure 5.4** is the confusion matrix analysis diagram of the similar code diff result retrieved by our proposed classification method.

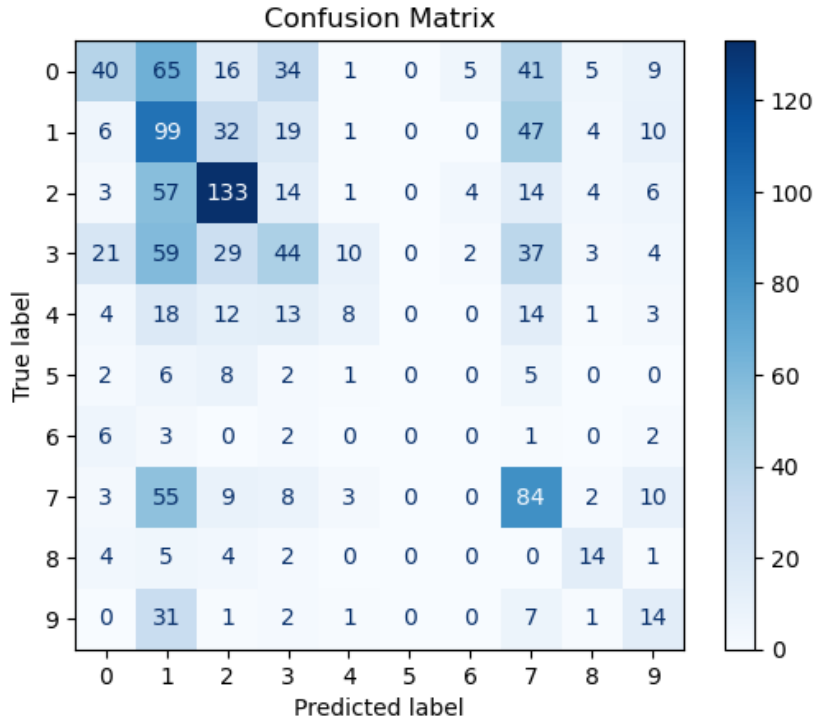


Figure 5.4: Confusion Matrix for the ARC-PR Method on the VDO Format Dataset

The above matrix again shows a relatively strong diagonal, indicating that the model performs well in predicting certain commit types correctly. For example, class 2 has 133 correct predictions, and class 7 has 84 correct predictions. Compared with the original RAG method’s confusion matrix, our proposed method’s confusion matrix shows a notable improvement in the model’s accuracy and performance compared to the first matrix. There is a stronger diagonal dominance and fewer scattered misclassifications, suggesting that the model has improved in predicting the correct commit types.

Finally, to verify the effectiveness of our proposed method in the commit message generation task, we tested our proposed method on the VDO format test dataset. **Table 5.26** shows the score comparison of our method with the original RAG method and the SOTA method RACE in various metrics.

Pred Model	BLEU	B-Norm	Rouge-L	Meteor	Log-MNEXT
LLM zero-shot	2.54	3.13	19.44	16.95	19.83
LLM RAG	2.72	3.03	18.41	16.98	19.60
ARC-PR	2.85	3.25	20.16	17.21	19.84
RACE	3.12	4.11	23.37	14.63	17.81

Table 5.26: Comparison of the ARC-PR Method, Previous LLM-Based Methods, and RACE on the VDO Format Dataset Across Five Metrics

From **Table 5.26**, we can see that our proposed method shows improvements over the previous LLM-based methods in both BLEU and BLEU-Norm metrics. However, it still trails behind the RACE method in these metrics. To further investigate the reasons behind these score improvements, we have calculated the average n-gram precisions and average brevity penalty for these three methods. The results are shown in the table below:

	Average BP	1-gram	2-gram	3-gram	4-gram	BLEU
LLM zero-shot	1	0.142	0.034	0.014	0.006	2.54
LLM RAG	1	0.130	0.031	0.015	0.009	2.72
ARC-PR	1	0.156	0.035	0.015	0.008	2.85
RACE	0.64	0.211	0.055	0.0263	0.018	3.12

Table 5.27: Comparison of Average Brevity Penalty and BLEU Precision Scores for the ARC-PR Method, LLM RAG, and LLM Zero-Shot Method on the VDO Format Dataset

From **Table 5.27**, we observe that our proposed method shows an improvement in n-gram precision compared to the LLM RAG method as well as LLM zero-shot method, indicating higher phrase-matching accuracy. The Average Brevity Penalty for the RACE method is significantly lower than for our method, suggesting that RACE generates shorter messages, and the text generated by our method is more consistent in length with the reference texts than the RACE method. Due to RACE’s higher n-gram matching precision, it achieves the best BLEU score among the three methods. In summary, on the VDO format dataset, our method does show an improvement in phrase-matching accuracy over the previous LLM-based methods. However, it still lags behind the RACE method in terms of matching accuracy.

From **Table 5.26**, we also observe that our proposed method shows an improvement in Rouge-L compared to the previous LLM-based methods, but it is still lower than the score achieved by the RACE method. To further investigate the density of effective information contained in the messages generated by these three methods, we have calculated the average length of common subsequences and the average message length for each method. The results are shown in the table below:

	Average LCS	Average message length
LLM zero-shot	1.62	10.65
LLM RAG	1.70	13.31
ARC-PR	1.61	10.05
RACE	1.34	5.11

Table 5.28: Comparison of Average LCS and Message Length for LLM RAG with and without Classification Module on the VDO Format Dataset

From **Table 5.28**, we can observe that although our proposed method does not show an improvement in average LCS compared to the previous LLM-based method, it reduces the average message length by approximately 6% comparing to the LLM zero-shot method and approximately 25% comparing to the LLM RAG method. This results in a higher proportion of effective information per unit length of the message, contributing to an increase in the final Rouge-L score. Compared to the RACE method, although the average common subsequence length of our method is longer, the RACE method generates messages that are only half as long as ours. This means that the density of effective information in our method is lower than in RACE, leading to a lower final Rouge-L score. In summary, our method still enhances the density of effective information over the previous LLM-based method on the VDO format dataset. Although this density is lower than that of the RACE method, the total amount of effective information remains higher than that of RACE.

On the METEOR and Log-MNEXT metrics, our proposed optimal LLM RAG method outperforms both the LLM-based methods and the RACE method. To further evaluate the performance of our proposed method in terms of message format consistency, as well as the balance between accuracy and recall, we have calculated the Average Fragment Penalty and Average F1 Score for the three methods on these metrics. The results are shown in the table below:

	Meteor		Log-MNEXT	
	Average FP	Average F1	Average FP	Average F1
LLM zero-shot	0.321	0.260	0.299	0.290
LLM RAG	0.326	0.264	0.299	0.289
ARC-PR	0.331	0.266	0.300	0.291
RACE	0.300	0.226	0.283	0.282

Table 5.29: Comparison of Meteor and Log-MNEXT Metrics for ARC-PR, LLM RAG, LLM Zero-Shot, and RACE Methods on the VDO Format Dataset

From **Table 5.29**, we can see that, on the METEOR and Log-MNEXT metrics, our proposed method does not show significant differences from the other two LLM-based methods in terms of average F1 score and average fragment penalty. This indicates that our method does not offer substantial improvements in message format consistency or in balancing accuracy and recall.

When comparing our method with RACE on the METEOR metric, RACE has a lower Fragment Penalty due to its higher VDO format message generation rate of 78%, compared to 77% for our method. However, our method surpasses RACE in the Average F1 Score, demonstrating a more balanced accuracy and recall. This is further reflected in the higher Average F1 Score of our method on the Log-MNEXT metric. In summary, on the VDO format dataset, our proposed method does not show significant differences compared to the LLM zero-shot method in terms of message format consistency or balancing accuracy and recall. However, compared to the RACE method, our approach achieves a better balance between precision and recall.

In the generalizability study, we observed that compared to the RACE method, the advantages of our method are significantly reduced when evaluated on the Angular format dataset. Analyzing the results from the Confusion Matrix section, we believe that the main reason is the insufficient classification accuracy of our model on the VDO format dataset. This limitation has, to some extent, reduced the proportion of good demonstrations, which is one of the areas for future research.

To confirm that the reduced performance of our method on the VDO format dataset is due to the current method’s insufficient accuracy, we compared the LLM RAG with golden matching against the RACE method. The results are shown below.

Pred Model	BLEU	B-Norm	Rouge-L	Meteor	Log-MNEXT
RAG with golden matching	4.44	5.09	25.76	21.51	24.57
RACE	3.12	4.11	23.37	14.63	17.81

Table 5.30: Comparison of the LLM RAG with golden matching and RACE on the VDO Format Dataset Across Five Metrics

As seen in **Table 5.30**, when the proportion of high-quality demonstrations reaches 100%, the LLM method still outperforms the RACE method. This supports our hypothesis regarding the impact of good demonstrations and indicates that our classification model has significant potential for further improvement.

Conclusion: The generalizability evaluation shows that our method demonstrates a certain level of robustness. When applied to datasets with different formats, it continues to enhance phrase-matching accuracy and increase the density of effective information compared to previous LLM-based methods. However, the comparison with the RACE method highlights some accuracy issues in our current classification model.

5.3. Experimental Results For RQ3

The goal of this first set of experiments is to validate or refute H.3.1 in response to RQ1. We organized a human evaluation involving 10 volunteers, for whom we designed questionnaires and evaluation forms. Initially, we used statistical methods to select the most relevant participants and analyzed their backgrounds. We then extracted the average scores and standard deviations for our proposed method

and the state-of-the-art (SOTA) method, RACE, focusing on expressiveness and informativeness. Additionally, we recorded participants' preferences for the two message-generation methods to support or challenge our hypothesis.

We had a total of 10 volunteers participate in this human evaluation test. Among them, 70% are master's students and 30% are employees. Half of the participants have 4-5 years of development experience, and most are proficient in Python and C++. All participants have experience with Git development, with 60% being skilled in various Git command lines. Relevant background information about the participants can be found in the appendix B.

We randomly sampled 50 commits, including 9 out of 10 Angular types. For each commit, we provided messages generated by both RACE and our method. The order of the messages for each commit was randomized to minimize evaluator bias, and evaluators were not informed which message was generated by which method.

We evaluated the quality of the commit messages based on expressiveness and informativeness. Evaluators rated each commit message on a scale of 0 to 4 for both dimensions. Additionally, for each code diff, we asked evaluators to choose the commit message they felt best aligned with the current code diff.

To ensure consistency among the raters, we calculated the Kendall rank correlation coefficient values [40]. We selected 4 participants from the 10 who had the highest pairwise Kendall's Tau values, ranging from 0.60 to 0.75. These values indicate a high degree of agreement among the 4 volunteers, suggesting that their scores are reliable. All selected volunteers have experience in Python and C++ and each has at least 3 years of development experience.

Table 5.31 and **Figure 5.5** show the result of human evaluation.

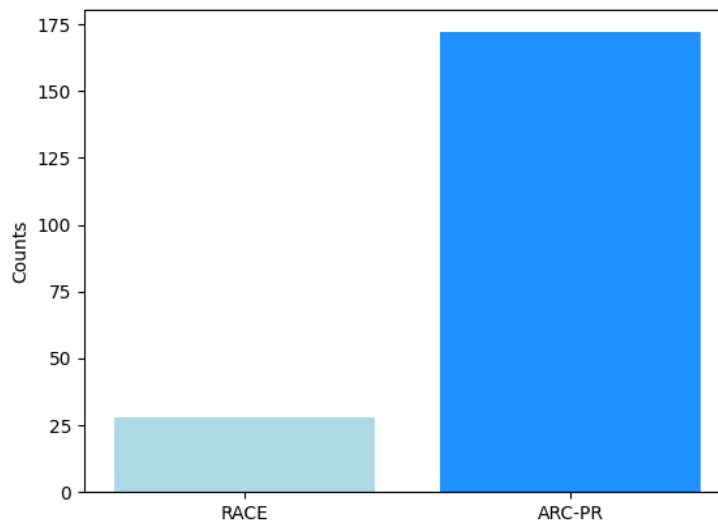


Figure 5.5: Human Evaluation Results: Number of Times Selected as the Best Commit Message for Each Code Diff

Figure 5.5 shows that in 86% of commits, people prefer the messages generated by our method, which means our method has better message quality than RACE in 86% of the evaluated commits.

Model	Expressiveness	Informativeness
RACE	2.86 (± 0.45)	2.51 (± 0.35)
Ours	3.76 (± 0.15)	3.50 (± 0.07)

Table 5.31: Human Evaluation Results (Standard Deviation in Parentheses)

Table 5.31 illustrates that our method performs comparably to RACE in terms of expressiveness and is much better than RACE in terms of informativeness. In our method, informativeness and expressiveness have lower standard deviations, which means our method receives consistent positive feedback in people’s evaluations.

To test whether there are significant differences between our method and the RACE method in terms of expressiveness and informativeness, we performed the t-test.

First, we performed the Shapiro–Wilk test [72] to examine whether the differences between the two methods followed a normal distribution. The results indicated that the differences in informativeness and expressiveness did not meet the criteria for normal distribution. Given that the differences did not follow a normal distribution, we introduced the Wilcoxon signed-rank test [88] to verify whether or not there was a significant difference between the two methods’ mean scores.

Model	Informativeness	Expressiveness
RACE	$1.27e^{-5} \pm 1.31e^{-5}$	$4.53e^{-4} \pm 7.84e^{-4}$

Table 5.32: Statistical Significance (p-Value) of Our Proposed Method Compared to RACE Approaches in Human Evaluation

Table 5.32 shows that improvement of our method over RACE is statistically significant with Informativeness’s and Expressiveness’s p-values smaller than 0.05 at 95% confidence level, therefore, we can reject the null hypothesis, and the median difference is not zero. In other words, an effect exists in our proposed method over the RACE method.

In summary, our method is particularly outstanding compared to RACE in terms of the expressiveness and informativeness indicator, which is why the message quality generated by our method in most commits is higher than that of RACE.

Conclusion: Through human evaluation, we found that our proposed enhanced LLM RAG method is not only better than the state-of-the-art method RACE in terms of expressiveness but also significantly surpasses RACE in informativeness. Additionally, people prefer the commit messages generated by our method.

6

Discussions

In this chapter, we first address the key findings of our research by answering the research questions based on the experimental results (Section 6.1). Next, we discuss the limitations of our study (Section 6.2), and finally, we explore potential directions for future work (Section 6.3)

6.1. Key Findings

The purpose of this study is to explore how prompt engineering can be used to facilitate Large Language Models to generate high-quality commit messages, and further, to help developers save time writing commit messages and help the project manager better organize projects.

To address RQ1 What is the effectiveness of LLM zero-shot on commit message generation in a unified message style dataset compared to the existing methods?

We conducted an experimental test using an Angular format commit message dataset to evaluate both LLMs and state-of-the-art methods in hybrid and retrieval-based approaches. Using five evaluation metrics, we performed a detailed analysis of effectiveness and conducted a comparative review. Our findings indicate that LLMs have certain advantages over hybrid and retrieval-based methods in terms of the overall amount of effective information in the messages. However, LLMs still fall short compared to state-of-the-art methods in other areas. Specifically, NNGen excelled in adhering to the format of the generated messages, while the RACE method demonstrated superior performance in phrase-matching accuracy, information density, and balancing precision and recall.

The LLM zero-shot method doesn't fully tap into the potential of LLMs, resulting in performance that falls short in terms of expressiveness and informativeness compared to state-of-the-art methods.

To address RQ2 What is the impact of different component settings on the generation capabilities of the LLM RAG method? What constitutes a good demonstration of the LLM RAG method? Can the effectiveness of the LLM method in generating commit messages be improved by ARC-PR? And what is the effectiveness of the ARC-PR method compared to the state-of-the-art method? Additionally, what is the robustness and generalizability of the ARC-PR method?

First, we conducted a validation experiment to assess the impact of different component settings on the generation capabilities of the LLM RAG method. Our findings indicate that, for commit message generation tasks, the embedding model has minimal effect on the output of the LLM RAG method, while the chunking size does have a noticeable impact. Consequently, the choice of embedding model configuration can be considered negligible for subsequent experiments, and we recommend using a chunking size of 1000.

To determine if demonstrations with the same type as the target code diff improve LLMs' ability

to generate relevant commit messages, we designed an ideal scenario where the LLM RAG method achieves 100% prediction accuracy (referred to as LLM RAG with golden matching). By comparing this with the LLM RAG method that does not use type-specific demonstrations, we found that the LLM RAG with golden matching significantly outperforms the original RAG method in several areas: matching accuracy, consistency in commit message format, and the proportion of information contained in messages of unit length. This highlights the positive impact of precise angular type matching on the effectiveness of LLM RAG commit message generation.

Building on the above findings, we propose a new LLM-based method called Adaptive RAG with Commit Type Classification and Partitioned Retrieval (ARC-PR). This approach introduces a Classification Module and a Database Partitioning Module, significantly increasing the proportion of high-quality demonstrations compared to the original LLM RAG method, bringing it closer to the ideal prediction scenario.

Through further experimental comparisons, we found that the ARC-PR method shows notable improvements over previous LLM-based methods. It excels in the accuracy of matching generated commit messages, word and phrase sequence matching, and message format consistency.

Additionally, we compared our proposed method with RACE, the state-of-the-art method in the CMG domain. While our method demonstrates comparable performance to RACE in phrase-matching accuracy, it surpasses RACE in terms of effective information density, message format consistency, and the balance between precision and recall. Overall, our method shows superior performance to the RACE method across all evaluation metrics.

Finally, to assess the generalizability of our method, we collected a VDO format dataset to evaluate whether our approach can achieve similar improvements in commit message generation as observed with the Angular format dataset. The generalizability evaluation reveals that our method demonstrates significant robustness. When applied to datasets with different formats, our method continues to enhance phrase-matching accuracy and increase the density of effective information compared to previous LLM-based methods.

Although there are some gaps in effective information density and n-gram matching accuracy compared to the RACE method—primarily due to the lower proportion of high-quality demonstrations in the VDO format dataset—our approach still achieves a better balance between precision and recall while maintaining a richer overall amount of effective information.

To address RQ3 How does the effectiveness of ARC-PR method compare to state-of-the-art approaches in generating commit messages based on human evaluation?

We conducted a manual evaluation to assess the advantages and disadvantages of our proposed method compared to the current state-of-the-art (SOTA) method, focusing on expressiveness and informativeness.

To mitigate errors arising from varying evaluation preferences among participants, we calculated Kendall's Tau correlation to ensure agreement among the volunteers. From the ten volunteers, we selected four who exhibited high agreement for evaluation.

In analyzing the results, we employed Wilcoxon signed-rank tests to statistically determine if there were significant differences between paired samples. The results show that our proposed ARC-PR method not only outperforms the SOTA method RACE in expressiveness but also significantly excels in informativeness. Furthermore, evaluators prefer the commit messages generated by our method.

6.2. Limitations

Although this study offers valuable insights into improving the commit message generation capabilities of LLMs, we must acknowledge several limitations that may impact the broader applicability of our findings.

We have identified the following main limitations:

- *Existing Methods Choosing.* In this thesis, we have compared LLM methods only with state-of-the-art techniques in the CMG domain, without extending the comparison to other existing methods. This approach may overlook the possibility that other methods could outperform the state-of-the-art in specific aspects, potentially introducing some error.
- *Programming Languages.* We conducted experiments exclusively on the JavaScript programming language. Although our framework is designed to be language-agnostic, model performance can vary across different programming languages. Therefore, additional experiments are needed to validate the generalizability of our framework. We plan to extend our study to include other programming languages in future work.
- *Amount of code diff.* Our research focuses solely on commits involving a single file modification and does not address commits with multiple file modifications.
- *Data Collection.* Due to the uneven distribution of data across different Angular types, our training and test datasets may be relatively small for some types, potentially introducing test bias. For the VDO format dataset, we simplified multiple similar types within the same sequence number into a single type. This simplification could lead to situations where the VDO type generated by the CMG method and the type in the reference message are the same, but our simplification process might result in inadequate recognition of these cases, potentially introducing errors.
- *Data sampling.* For the test data sampling method, we employed stratified sampling, collecting our training data in proportion to the distribution of types present in the training dataset, rather than using a sampling strategy with an equal number of types. Consequently, this sampling approach may also influence the results to some extent.
- *Human evaluation.* The number of commits evaluated was limited, and the evaluators had varying levels of experience in JavaScript development, which may introduce a degree of subjectivity to the evaluation. Additionally, due to the diverse backgrounds of the evaluators, we selected evaluation content from only two participants to ensure consistency. This selection process may also contribute to certain errors.

6.3. Future Work

Based on this study, we proposed the following recommendations for future study.

For future research, several directions can be explored:

1. **Enhanced Classification Methods:** This study conducted preliminary exploratory research on improving the accuracy of code diff type prediction to enhance the matching accuracy of the RAG system and assist LLMs in generating higher-quality commit messages. However, the classification accuracy of the current method is still quite low, indicating significant room for improvement. Future work will involve introducing more advanced classification models to address this issue.
2. **Alternative Prompting Methods:** Exploring other prompting techniques, such as Chain-of-Thought, could further investigate and harness the full potential of LLMs.
3. **Specialized Embedding Models:** Training an embedding model specifically tailored for commit message generation could be beneficial. This would involve manually collecting relevant code diff data to create a robust training dataset for the RAG system.

7

Conclusion

This thesis explores more efficient prompt engineering methods to enhance the performance of Large Language Models (LLMs) in the commit message generation (CMG) domain. We adopted an empirical approach to investigate the impact of LLMs using zero-shot methods and Retrieval-Augmented Generation (RAG) configurations on commit message generation within software development environments. Additionally, we proposed a new LLM-based method to address gaps identified in previous research.

Our study employed a detailed experimental design involving various methods and datasets to evaluate the effectiveness of these technologies in improving the quality and efficiency of commit messages. Our findings demonstrated that our proposed method exhibits superior generation capabilities on certain unified message format datasets.

Specifically, our experiments revealed that the LLM zero-shot method, compared to state-of-the-art hybrid and retrieval-based methods, offers notable advantages in the total amount of effective information included in commit messages. However, despite this advantage, LLMs still fall short in areas such as effective information density, message format consistency, and phrase-matching accuracy when compared to established methods like NNGen and RACE.

Next, we conducted further research on the existing LLM Retrieval-Augmented Generation (RAG) method, investigating how different configurations impact its generation capabilities through controlled experiments. We found that, for the commit message generation (CMG) task, the choice of the embedding model had minimal effect on the performance of the LLM RAG method, while the chunking size did have a notable influence.

We also explored what constitutes an effective demonstration of the LLM RAG method. Through our observations, we identified an issue where the retrieved similar message types did not always align with the ground truth message types, potentially degrading the quality of the demonstrations. To address this, we conducted an idealized experiment focusing on type-aligned demonstrations. The results revealed that improving the alignment of message types significantly enhances the effectiveness of the LLM RAG method.

To address this issue, we introduced an enhanced method called Adaptive Retrieval-Augmented Generation with Commit Type Classification and Partitioned Retrieval (ARC-PR). This method incorporates a classification module and a database partitioning module to better align retrieved message types with ground truth types. Experimental validation showed that ARC-PR significantly improved type-matching accuracy, effective information density, message format consistency, and the balance between precision and recall compared to both the LLM RAG and zero-shot methods.

Moreover, our method outperformed the state-of-the-art RACE method across several metrics, including effective information density, message format consistency, and the balance of accuracy and recall. This evidence underscores that ARC-PR offers stronger generation capabilities and practical value in the CMG field compared to existing SOTA methods.

The generalizability of our method was tested using another unified format dataset, demonstrating that our proposed approach significantly improves phrase-matching accuracy and effective information density compared to previous LLM-based methods. When compared to the state-of-the-art RACE method, our approach also exhibits a more balanced performance in terms of accuracy and recall, along with a higher total amount of effective information. However, its shortcomings in other areas indicate that there is still room for improvement in our method's classification model.

Finally, human evaluations confirmed that our method not only surpasses the state-of-the-art RACE method in expressiveness but also excels in informativeness. Evaluators preferred our method for its superior description quality.

Our contributions can be summarized as follows:

- **We provide a comprehensive assessment of LLM-based commit message generation approaches in comparison to existing CMG methods, highlighting their respective strengths and limitations.**
- **We explored the impact of various component configurations on the performance of the LLM RAG method.**
- **We investigated what constitutes a good demonstration of the LLM RAG method for the CMG task.**
- **We propose a new prompt engineering approach called Adaptive Retrieval Augmented Generation with Commit Type Classification and Partitioned Retrieval (ARC-PR). This method increases access to high-quality demonstrations related to code changes compared to existing CMG techniques, addressing the issues of low informativeness and expressiveness observed in previous state-of-the-art methods.**
- **Experimental results show that ARC-PR overall achieves state-of-the-art performance in commit message generation and each component in ARC-PR is effective.**
- **We introduced a comprehensive human evaluation, assessing our method and the state-of-the-art method RACE from multiple perspectives, including informativeness and expressiveness.**

While this study provides valuable insights into the effectiveness of LLMs and RAG configurations, it also acknowledges certain limitations and suggests several avenues for future research. These include:

1. **Broader Comparisons:** Expanding comparisons to include other existing methods beyond the state-of-the-art to ensure a more comprehensive evaluation.
2. **Validation Across Languages and Scenarios:** Testing the framework across different programming languages and commit scenarios to confirm its generalizability and robustness.
3. **Data Distribution and Sampling Challenges:** Addressing issues related to data distribution and sampling to improve the reliability of the results.
4. **Multiple file modifications:** Adapting the current approach to apply to the submission scenario of multiple file modifications.

Future work should focus on:

1. **Optimizing classification model:** Introducing more efficient benchmark pre-trained models to participate in classification tasks, thereby achieving higher classification accuracy to improve performance and applicability.
2. **Exploring Alternative Prompting Methods:** Investigating other prompting techniques, such as Chain-of-Thought, to fully leverage LLM capabilities.
3. **Integration in Educational and Development Contexts:** Effectively integrating these technologies into diverse educational and development environments to maximize their utility.

Overall, this research demonstrates that effective prompt engineering can significantly enhance the capabilities of LLMs, enabling them to outperform state-of-the-art methods in commit message generation while offering greater flexibility and lower training costs.

References

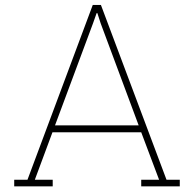
- [1] Josh Achiam et al. “Gpt-4 technical report”. In: *arXiv preprint arXiv:2303.08774* (2023).
- [2] Toufique Ahmed and Premkumar Devanbu. “Few-shot training LLMs for project-specific code-summarization”. In: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 2022, pp. 1–5.
- [3] Satanjeev Banerjee and Alon Lavie. “METEOR: An automatic metric for MT evaluation with improved correlation with human judgments”. In: *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*. 2005, pp. 65–72.
- [4] Tom Brown et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.
- [5] Raymond PL Buse and Westley R Weimer. “Automatically documenting program changes”. In: *Proceedings of the 25th IEEE/ACM international conference on automated software engineering*. 2010, pp. 33–42.
- [6] Mark Chen et al. “Evaluating large language models trained on code”. In: *arXiv preprint arXiv:2107.03374* (2021).
- [7] Zhoujun Cheng et al. “Binding language models in symbolic languages”. In: *arXiv preprint arXiv:2210.02875* (2022).
- [8] Luis Fernando Cortés-Coy et al. “On automatically generating commit messages via summarization of source code changes”. In: *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. IEEE. 2014, pp. 275–284.
- [9] John W Creswell and Cheryl N Poth. *Qualitative inquiry and research design: Choosing among five approaches*. Sage publications, 2016.
- [10] Marco D’Ambros, Michele Lanza, and Romain Robbes. “Commit 2.0”. In: *Proceedings of the 1st Workshop on Web 2.0 for Software Engineering*. 2010, pp. 14–19.
- [11] Michael Denkowski and Alon Lavie. “Meteor-next and the meteor paraphrase tables: Improved evaluation support for five target languages”. In: *Proceedings of the joint fifth workshop on statistical machine translation and MetricsMATR*. 2010, pp. 339–342.
- [12] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).
- [13] Samanta Dey et al. “Evaluating commit message generation: to BLEU or not to BLEU?” In: *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*. 2022, pp. 31–35.
- [14] Jinhao Dong et al. “Fira: fine-grained graph-based code change representation for automated commit message generation”. In: *Proceedings of the 44th International Conference on Software Engineering*. 2022, pp. 970–981.
- [15] Jinhao Dong et al. “Revisiting learning-based commit message generation”. In: *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE. 2023, pp. 794–805.
- [16] Qingxiu Dong et al. “A survey on in-context learning”. In: *arXiv preprint arXiv:2301.00234* (2022).
- [17] Mengnan Du et al. “Shortcut learning of large language models in natural language understanding”. In: *Communications of the ACM* 67.1 (2023), pp. 110–120.
- [18] Xueying Du et al. “Evaluating large language models in class-level code generation”. In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 2024, pp. 1–13.

- [19] Robert Dyer et al. “Boa: A language and infrastructure for analyzing ultra-large-scale software repositories”. In: *2013 35th International Conference on Software Engineering (ICSE)*. IEEE. 2013, pp. 422–431.
- [20] Aleksandra Eliseeva et al. “From commit message generation to history-aware commit message completion”. In: *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2023, pp. 723–735.
- [21] Khashayar Etemadi and Martin Monperrus. “On the relevance of cross-project learning with nearest neighbours for commit message generation”. In: *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. 2020, pp. 470–475.
- [22] Angela Fan et al. “Large language models for software engineering: Survey and open problems”. In: *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. IEEE. 2023, pp. 31–53.
- [23] Zhangyin Feng et al. “Codebert: A pre-trained model for programming and natural languages”. In: *arXiv preprint arXiv:2002.08155* (2020).
- [24] Daniel Fried et al. “InCoder: A generative model for code infilling and synthesis”. In: *arXiv preprint arXiv:2204.05999* (2022).
- [25] Shuzheng Gao et al. “Learning in the wild: Towards leveraging unlabeled data for effectively tuning pre-trained code models”. In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 2024, pp. 1–13.
- [26] Shuzheng Gao et al. “What makes good in-context demonstrations for code intelligence tasks with llms?” In: *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2023, pp. 761–773.
- [27] Yunfan Gao et al. “Retrieval-augmented generation for large language models: A survey”. In: *arXiv preprint arXiv:2312.10997* (2023).
- [28] Mingyang Geng et al. “Large language models are few-shot summarizers: Multi-intent comment generation via in-context learning”. In: *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 2024, pp. 1–13.
- [29] Yichen He et al. “COME: Commit Message Generation with Modification Embedding”. In: *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2023, pp. 792–803.
- [30] Thong Hoang et al. “Cc2vec: Distributed representations of code changes”. In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 2020, pp. 518–529.
- [31] Xinyi Hou et al. “Large language models for software engineering: A systematic literature review”. In: *arXiv preprint arXiv:2308.10620* (2023).
- [32] Yizheng Huang and Jimmy Huang. “A Survey on Retrieval-Augmented Text Generation for Large Language Models”. In: *arXiv preprint arXiv:2404.10981* (2024).
- [33] Yuan Huang et al. “Learning human-written commit messages to document code changes”. In: *Journal of Computer Science and Technology* 35 (2020), pp. 1258–1277.
- [34] Yuan Huang et al. “Mining version control system for automatically generating commit comment”. In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE. 2017, pp. 414–423.
- [35] Siyuan Jiang, Ameer Armaly, and Collin McMillan. “Automatically generating commit messages from diffs using neural machine translation”. In: *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2017, pp. 135–146.
- [36] Siyuan Jiang and Collin McMillan. “Towards automatic generation of short summaries of commits”. In: *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE. 2017, pp. 320–323.
- [37] Tae-Hwan Jung. “Commitbert: Commit message generation using pre-trained programming language model”. In: *arXiv preprint arXiv:2105.14242* (2021).

- [38] Huzefa Kagdi, Michael L Collard, and Jonathan I Maletic. "A survey and taxonomy of approaches for mining software repositories in the context of software evolution". In: *Journal of software maintenance and evolution: Research and practice* 19.2 (2007), pp. 77–131.
- [39] Sungmin Kang, Juyeon Yoon, and Shin Yoo. "Large language models are few-shot testers: Exploring llm-based general bug reproduction". In: *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE. 2023, pp. 2312–2323.
- [40] Maurice G Kendall. "The treatment of ties in ranking problems". In: *Biometrika* 33.3 (1945), pp. 239–251.
- [41] Zhenzhong Lan et al. "Albert: A lite bert for self-supervised learning of language representations". In: *arXiv preprint arXiv:1909.11942* (2019).
- [42] Caroline Lemieux et al. "Codamosa: Escaping coverage plateaus in test generation with pre-trained large language models". In: *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE. 2023, pp. 919–931.
- [43] Jia Allen Li et al. "Editsum: A retrieve-and-edit framework for source code summarization". In: *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2021, pp. 155–166.
- [44] Chin-Yew Lin. "Rouge: A package for automatic evaluation of summaries". In: *Text summarization branches out*. 2004, pp. 74–81.
- [45] Chin-Yew Lin and Franz Josef Och. "Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics". In: *Proceedings of the 42nd annual meeting of the association for computational linguistics (ACL-04)*. 2004, pp. 605–612.
- [46] Mario Linares-Vásquez et al. "Changescribe: A tool for automatically generating commit messages". In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 2. IEEE. 2015, pp. 709–712.
- [47] Qin Liu et al. "Generating commit messages from diffs using pointer-generator network". In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE. 2019, pp. 299–309.
- [48] Shangqing Liu et al. "Atom: Commit message generation based on abstract syntax tree and hybrid ranking". In: *IEEE Transactions on Software Engineering* 48.5 (2020), pp. 1800–1817.
- [49] Yiheng Liu et al. "Summary of chatgpt-related research and perspective towards the future of large language models". In: *Meta-Radiology* (2023), p. 100017.
- [50] Zhongxin Liu et al. "Neural-machine-translation-based commit message generation: how far are we?" In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 2018, pp. 373–384.
- [51] Cristina V Lopes et al. "Commit Messages in the Age of Large Language Models". In: *arXiv preprint arXiv:2401.17622* (2024).
- [52] Pablo Loyola, Edison Marrese-Taylor, and Yutaka Matsuo. "A neural architecture for generating natural language descriptions from source code changes". In: *arXiv preprint arXiv:1704.04856* (2017).
- [53] Pablo Loyola et al. "Content aware source code change description generation". In: *Proceedings of the 11th International Conference on Natural Language Generation*. 2018, pp. 119–128.
- [54] Shuai Lu et al. "Codexglue: A machine learning benchmark dataset for code understanding and generation". In: *arXiv preprint arXiv:2102.04664* (2021).
- [55] Walid Maalej and Hans-Jörg Happel. "Can development work describe itself?" In: *2010 7th IEEE working conference on mining software repositories (MSR 2010)*. IEEE. 2010, pp. 191–200.
- [56] Walid Maalej and Hans-Jörg Happel. "From work to word: How do software developers describe their work?" In: *2009 6th IEEE International Working Conference on Mining Software Repositories*. IEEE. 2009, pp. 121–130.
- [57] Gary Marcus. "The next decade in AI: four steps towards robust artificial intelligence". In: *arXiv preprint arXiv:2002.06177* (2020).

- [58] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).
- [59] Mockus and Votta. "Identifying reasons for software changes using historic databases". In: *Proceedings 2000 international conference on software maintenance*. IEEE. 2000, pp. 120–130.
- [60] Gail Murphy. "Attacking information overload in software development". In: *2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE. 2009, pp. 4–4.
- [61] Noor Nashid, Mifta Sintaha, and Ali Mesbah. "Retrieval-based prompt selection for code-related few-shot learning". In: *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE. 2023, pp. 2450–2462.
- [62] Lun Yiu Nie et al. "Coregen: Contextualized code representation learning for commit message generation". In: *Neurocomputing* 459 (2021), pp. 97–107.
- [63] OpenAI. *ChatGPT: A Large Language Model*. <https://www.openai.com/chatgpt>. 2023.
- [64] Ipek Ozkaya. "Application of large language models to software engineering tasks: Opportunities, risks, and implications". In: *IEEE Software* 40.3 (2023), pp. 4–8.
- [65] Kishore Papineni et al. "Bleu: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318.
- [66] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.
- [67] Matthew E. Peters et al. "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Ed. by Marilyn Walker, Heng Ji, and Amanda Stent. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. URL: <https://aclanthology.org/N18-1202>.
- [68] Fabio Petroni et al. "Language models as knowledge bases?" In: *arXiv preprint arXiv:1909.01066* (2019).
- [69] Alina Petukhova, Joao P Matos-Carvalho, and Nuno Fachada. "Text clustering with LLM embeddings". In: *arXiv preprint arXiv:2403.15112* (2024).
- [70] Chengwei Qin et al. "Is ChatGPT a general-purpose natural language processing task solver?" In: *arXiv preprint arXiv:2302.06476* (2023).
- [71] Sarah Rastkar and Gail C Murphy. "Why did this code change?" In: *2013 35th International Conference on Software Engineering (ICSE)*. IEEE. 2013, pp. 1193–1196.
- [72] Samuel Sanford Shapiro and Martin B Wilk. "An analysis of variance test for normality (complete samples)". In: *Biometrika* 52.3-4 (1965), pp. 591–611.
- [73] Jinfeng Shen et al. "On automatic summarization of what and why information in source code changes". In: *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. IEEE. 2016, pp. 103–112.
- [74] Ensheng Shi et al. "Race: Retrieval-augmented commit message generation". In: *arXiv preprint arXiv:2203.02700* (2022).
- [75] Weisong Sun et al. "Automatic code summarization via chatgpt: How far are we?" In: *arXiv preprint arXiv:2305.12865* (2023).
- [76] Wei Tao et al. "KADEL: Knowledge-Aware Denoising Learning for Commit Message Generation". In: *ACM Transactions on Software Engineering and Methodology* (2024).
- [77] Wei Tao et al. "On the evaluation of commit message generation models: An experimental study". In: *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2021, pp. 126–136.
- [78] Yida Tao et al. "How do software engineers understand code changes? An exploratory study in industry". In: *Proceedings of the ACM SIGSOFT 20th International symposium on the foundations of software engineering*. 2012, pp. 1–11.

- [79] Yingchen Tian et al. “What makes a good commit message?” In: *Proceedings of the 44th International Conference on Software Engineering*. 2022, pp. 2389–2401.
- [80] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).
- [81] Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. “Cider: Consensus-based image description evaluation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 4566–4575.
- [82] Chaozheng Wang et al. “No more fine-tuning? an experimental evaluation of prompt tuning in code intelligence”. In: *Proceedings of the 30th ACM joint European software engineering conference and symposium on the foundations of software engineering*. 2022, pp. 382–394.
- [83] Chong Wang et al. “Teaching Code LLMs to Use Autocompletion Tools in Repository-Level Code Generation”. In: *arXiv preprint arXiv:2401.06391* (2024).
- [84] Haoye Wang et al. “Context-aware retrieval-based deep commit message generation”. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30.4 (2021), pp. 1–30.
- [85] Weishi Wang et al. “Rap-gen: Retrieval-augmented patch generation with codet5 for automatic program repair”. In: *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2023, pp. 146–158.
- [86] Xinyi Wang et al. “Large language models are latent variable models: Explaining and finding good demonstrations for in-context learning”. In: *Advances in Neural Information Processing Systems* 36 (2024).
- [87] Frank Wilcoxon. “Individual comparisons by ranking methods”. In: *Breakthroughs in statistics: Methodology and distribution*. Springer, 1992, pp. 196–202.
- [88] Frank Wilcoxon, S Katti, Roberta A Wilcox, et al. “Critical values and probability levels for the Wilcoxon rank sum test and the Wilcoxon signed rank test”. In: *Selected tables in mathematical statistics* 1 (1970), pp. 171–259.
- [89] Thomas Wolf et al. “Huggingface’s transformers: State-of-the-art natural language processing”. In: *arXiv preprint arXiv:1910.03771* (2019).
- [90] Yifan Wu, Ying Li, and Siyu Yu. “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*. 2024, pp. 124–129.
- [91] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. “Automated program repair in the era of large pre-trained language models”. In: *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE. 2023, pp. 1482–1494.
- [92] Shengbin Xu et al. “Commit message generation for source code changes”. In: *IJCAI*. 2019.
- [93] Linghao Zhang et al. “Using Large Language Models for Commit Message Generation: A Preliminary Study”. In: *arXiv preprint arXiv:2401.05926* (2024).
- [94] Quanjun Zhang et al. “A critical review of large language model on software engineering: An example from chatgpt and automated program repair”. In: *arXiv preprint arXiv:2310.08879* (2023).
- [95] Quanjun Zhang et al. “A Systematic Literature Review on Large Language Models for Automated Program Repair”. In: *arXiv preprint arXiv:2405.01466* (2024).
- [96] Yubo Zhang et al. “RetCom: Information Retrieval-Enhanced Automatic Source-Code Summarization”. In: *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*. IEEE. 2022, pp. 948–957.
- [97] Yuxia Zhang et al. “Automatic commit message generation: A critical review and directions for future work”. In: *IEEE Transactions on Software Engineering* (2024).
- [98] Zibin Zheng et al. “A survey of large language models for code: Evolution, benchmarking, and future trends”. In: *arXiv preprint arXiv:2311.10372* (2023).



Human evaluation example

Listing A.1: Human evaluation example

```
1
2 Item 1
3
4 diff --git a/src/common/reducers/layout/index.js b/src/common/reducers
  /layout/index.js @@ -43,17 +43,10 @@ export function layout (state:
    State = initialState, action: Action): State {
5 }
6 switch (action.type) {
7 case APPLICATION_INIT:
8 - case UI_WINDOW_RESIZE: {
9 - const {innerWidth} = action.payload
10 - const {isMobile, isMobileSM, isMobileXS} = computeMobileStatuses(
11 - innerWidth
12 - )
13 + case UI_WINDOW_RESIZE:
14 return {
15 ...state,
16 - isMobile,
17 - isMobileSM,
18 - isMobileXS
19 - }
20 + ...computeMobileStatuses(action.payload.innerWidth)
21 }
22 case UI_OPEN_SIDEBAR:
23 return {
24
25 1: 'fix: mobile status'
26
27 Expressiveness:
28 Informativeness:
29
30 2: 'refactor(layout): simplify UI_WINDOW_RESIZE case in layout reducer
31
32 Expressiveness:
33 Informativeness:
34
35 Best description number:
```


B

Human evaluation Participants Background

Years of Experience in Software Development
10 responses

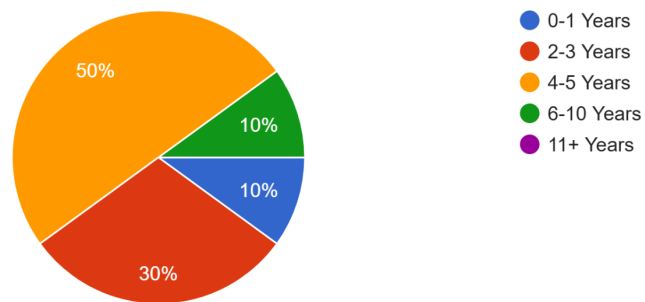


Figure B.1: Year Of Experience

Primary Programming language(s) (multi-options)

10 responses

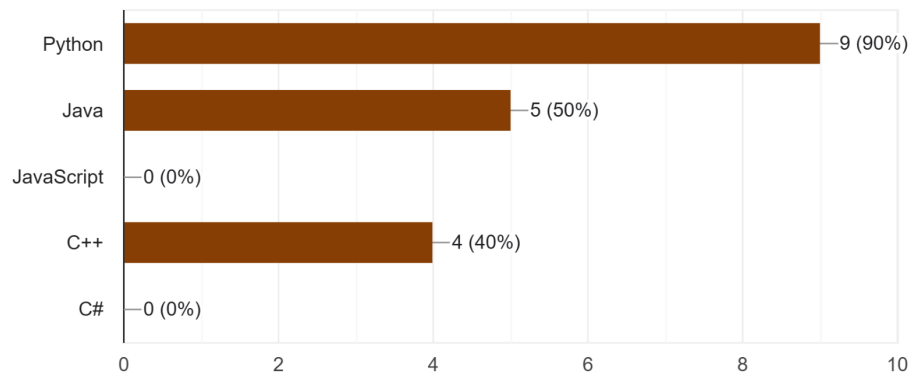


Figure B.2: Programming Language

Experience with Version Control Systems (e.g., Git)

10 responses

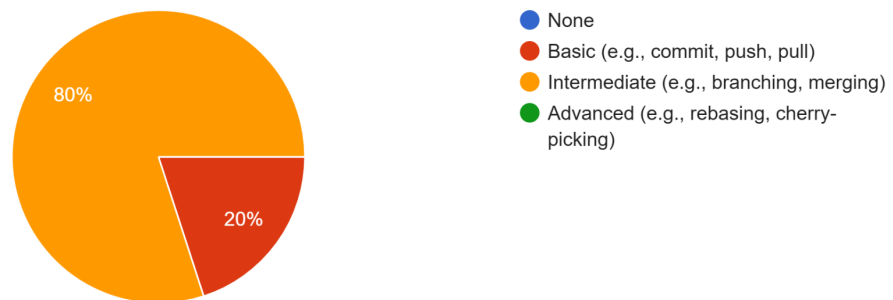


Figure B.3: Experience With Version Control System

Experience with Writing Commit Messages

10 responses

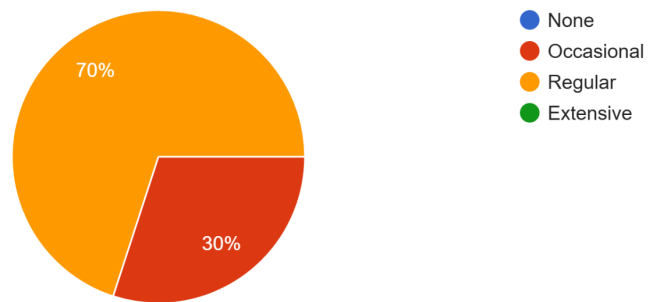


Figure B.4: Experience With Writing Commit Message