

# Automated Commit Message Generation with Large Language Models: An Empirical Study and Beyond

Pengyu Xue<sup>‡</sup>, Linhao Wu<sup>‡</sup>, Zhongxing Yu, Zhi Jin, Zhen Yang<sup>§</sup>, Xinyi Li, Zhenyu Yang, and Yue Tan

**Abstract**—Commit Message Generation (CMG) approaches aim to automatically generate commit messages based on given code *diffs*, which facilitate collaboration among developers and play a critical role in Open-Source Software (OSS). Very recently, Large Language Models (LLMs) have demonstrated extensive applicability in diverse code-related tasks owing to their powerful generality. Yet, in the CMG field, few studies systematically explored their effectiveness. This paper conducts the first comprehensive experiment to investigate how far we have been in applying LLM to generate high-quality commit messages and how to go further beyond in this field.

Motivated by a pilot analysis, we first clean the most widely-used CMG dataset following practitioners’ criteria. Afterward, we re-evaluate diverse state-of-the-art CMG approaches and make comparisons with recent LLMs, demonstrating the superior performance of LLMs against state-of-the-art CMG approaches. To delve deeper into LLMs’ ability, we further propose four manual metrics following the practice of OSS, including Accuracy, Integrity, Applicability, and Readability, and assess various LLMs accordingly. Results reveal that GPT-3.5 performs best overall, but different LLMs carry different advantages.

To further boost LLMs’ performance in the CMG task, we propose an Efficient Retrieval-based In-Context Learning (ICL) framework, namely ERICommitter, which leverages a two-step filtering to accelerate the retrieval efficiency and introduces semantic/lexical-based retrieval algorithm to construct the ICL examples, thereby guiding the generation of high-quality commit messages with LLMs. Extensive experiments demonstrate the substantial performance improvement of ERICommitter on various LLMs for code *diffs* of different programming languages. Meanwhile, ERICommitter also significantly reduces the retrieval time while keeping almost the same performance. Our research contributes to the understanding of LLMs’ capabilities in the CMG field and provides valuable insights for practitioners seeking to leverage these tools in their workflows.

**Index Terms**—Commit Message Generation, Large Language Model, Empirical Study, In-Context Learning.

## I. INTRODUCTION

**I**N the domain of software development, well-structured commit messages are essential for they provide insights into related information of code changes, facilitating better understanding and collaboration among team members. However,

writing commit messages manually can be time-consuming, and they are sometimes uninformative or even absent [1]. To overcome this issue, there have been substantial efforts in developing techniques to automatically generate commit messages, such as retrieval-based [2]–[4], and learning-based approaches [5]–[7].

In recent years, Large Language Models (LLMs) have shown exceptional performance across various domains of code intelligence, such as comment generation [8], [9], code translation [10], and automated program repair [11]. Moreover, some studies have also conducted preliminary investigations on the performance of LLM in the CMG field. Nonetheless, they either only tested on limited LLMs [7], [12], such as ChatGPT, or merely included few and monolingual experimental samples [13], lacking a systematic and comprehensive evaluation. Another study [14] focuses on improving pre-trained language models’ CMG performance by including commit-related issues as extra information. However, recent studies [15]–[18] revealed that most of the commits (62.9%) are not linked to issues, such as a bug fix or feature request, where false issue-commit links and less informative issues are prevalent. In addition, parameter tuning is also required, making it hard to apply to most LLMs in practice.

To fill the above gap, this paper conducted a systematic empirical study on diverse LLMs via automatic and manual evaluation, as well as proposed a training-free approach to boost their performance in the CMG task further. Initially, a pilot analysis in this paper found that human-authored commit messages in the most widely used CMG dataset, namely MCMD, are generally of poor quality, often lacking critical “why” (i.e., the reason or purpose behind code changes) and “what” (i.e., the specific changes made to the code) information, which are deemed two essential elements for a commit message [19]. Considering most previous proposed CMG approaches [2], [20], [21] were assessed on these datasets (e.g., MCMD [5]), leading to misleading guidance for developers in practice. This paper first cleans up the MCMD dataset according to the requirements of practitioners, ensuring all samples contain both “what” and “why” information. Subsequently, we further re-evaluate a wide range of state-of-the-art CMG approaches of various categories against recent LLMs, including GPT-3.5, LLaMA-7B/13B, and Gemini, in terms of a series of automated evaluation metrics following previous studies [2], [4], [22], [23], showing that LLMs demonstrate superior performance against previous proposed CMG approaches. In particular, GPT-3.5 performs best and surpasses the best-performing CMG approach, namely RACE, by 83.85% and 27.20% in the METEOR and BLEU metrics,

• Pengyu Xue, Linhao Wu, Xinyi Li, and Yue Tan are with the School of Political Science and Public Administration, Shandong University, Qingdao, China. E-mail: {xuepengyu, wulinhao, Li\_XinYi, tanyue}@mail.sdu.edu.cn

• Zhen Yang, Zhongxing Yu, and Zhenyu Yang are with the School of Computer Science and Technology, Shandong University, Qingdao, China. E-mail: {zhenyang, zhongxing.yu}@sdu.edu.cn, yangzycs@mail.sdu.edu.cn

• Zhi Jin is with the Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, and the School of Computer Science, Peking University, Beijing, China. E-mail: zhi-jin@pku.edu.cn

<sup>‡</sup> Pengyu Xue and Linhao Wu are co-first authors.

<sup>§</sup> Zhen Yang is the corresponding author.

respectively. Notably, a purely retrieval-based CMG approach, namely NNGen, outperforms many learning-based approaches on the cleaned dataset, demonstrating the importance of similar examples in the CMG task. To delve deeper into the CMG capability of LLMs, aligning with the requirements of OSS practice, we further conduct extensive manual assessments, covering aspects of Accuracy, Integrity, Applicability, and Readability. Manual assessments demonstrate that GPT-3.5 still performs the best among its various counterparts, but different LLMs carry different advantages. LLMs tend to convey more about “What” information but less detailed concerning “Why” owing to the limited available code context.

To better tap into the potential of LLMs and enhance the quality of commit messages generated by them, considering the impressive enhancement of ICL with LLM in extensive domains [24]–[27], and the high impact of similar examples in the CMG task as mentioned before, we propose an Efficient Retrieval-based In-Context Learning framework (ERICommitter) applicable to various LLMs in the CMG task. ERICommitter leverages two-step filtering to extract high-quality example candidates to reduce the volume of the retrieval database, thereby accelerating retrieval efficiency. Subsequently, ERICommitter retrieves a group of the most similar examples for ICL, where both semantic and lexical retrieval engines are examined. Our experimental results demonstrate notable improvements in different metrics compared to using LLMs alone. Specifically, GPT-3.5 obtains improvements of 15.2%, 4.75%, and 13.72% at most in terms of METEOR, BLEU, and ROUGE-L, respectively. Moreover, ERICommitter boosts the CMG performance of Gemini by 199.43%, 155.36%, and 167.77% at most in terms of each evaluation metric in order. Besides, our proposed two-step filtering method reduces the retrieval time to 6.22% for lexical-based retrieval and 7.15% for semantic-based retrieval, concurrently keeping almost the same performance, which effectively improves the practicality of the LLM-based CMG system. This research contributes significantly to the realm of commit message generation, summarized as follows:

- We constructed a high-quality, multi-lingual CMG test set using a three-step filtration process on the MCMD dataset to meet the practitioner’s needs.
- We carry out the first systematic empirical study to investigate the performance of various recent LLMs against state-of-the-art CMG approaches of diverse categories. Besides, manual assessment aligning with the practical utility from a wide range of perspectives is proposed in this paper and conducted on LLM-generated commit messages.
- We propose an Efficient Retrieval-based In-context learning framework applicable to LLMs in the CMG field, namely ERICommitter. Extensive experiments are conducted to demonstrate its effectiveness.

## II. RELATED WORKS

### A. Commit Message Generation

Commit Message Generation (CMG) aims to produce appropriate messages to describe a commit of code changes

(i.e., code *diffs*) in version control systems, which greatly facilitates collaboration among developers in open-source software practice. Over the years, various automated CMG approaches have been successively proposed. Liu et al. [3] propose a simpler and faster approach, named NNGen, to generate concise commit messages using the nearest neighbor algorithm. Tao et al. [5] perform a human evaluation and find the BLEU metric that best correlates with the human scores for the task. Liu et al. [28] propose ATOM, which advances commit messages by explicitly incorporating abstract syntax trees to represent code changes. Moreover, ATOM integrates both retrieved and generated commit messages using hybrid ranking. On top of that, the approach [29] coined as ChangeScribe is proposed to generate commit messages automatically from change sets. RACE is a method that treats retrieved similar commits as an exemplar and utilizes it to generate readable and informative commit messages [2]. Recently, introducing retrieved relevant results into the training process has been found useful in most generation tasks [30]–[32]. Shi et al. [2] treat the retrieved similar commit as an exemplar and train the model to utilize the exemplar for enhancing commit message generation. Additionally, many neural-based approaches have been used to learn the semantics of code *diffs* and translate them into commit messages. For example, NMTGen [33] and Commit-Gen [34] adopt the Seq2Seq neural network with different attention mechanisms for translating them into commit messages. CommitBERT [35] leverages CodeBERT as an initial model to resolve the large gap between programming language and natural language, making it easier for the Neural Machine Translation (NMT)-based model to learn the contextual representation. PtrGNCMsg [6] outperforms recent approaches based on the NMT structure, and first enables the prediction of out-of-vocabulary words. Very recently, researchers started to explore the potential of LLMs in the CMG field. Zhang et al. [13] explored the CMG performance of UniXcoder [36] and ChatGPT<sup>1</sup> for long code *diffs* and whole message generation. Lopes et al. [7] carried out a series of automatic and manual analyses on ChatGPT to investigate its CMG performance against previous proposed CMG approaches. However, different from this work, they either only experiment on limited LLMs or lack practical improvement and systematic evaluations for LLMs.

### B. What Is A Good Commit Message

It is undeniable that the quality of commit messages affects the effectiveness of communication among developers. However, commit messages are often of poor quality as developers lack the time and motivation to craft a good message. Hence, there is an urgent need to establish a universal standard for defining what constitutes a good commit message. Ma et al. [37] suppose that the quality of the commit message is in terms of informativeness, clearness, and length. On the other hand, Tian et al. [19] contend that the most frequently recognized expectation of a commit message is to summarize the changes in this commit (noted as “What”) and describe the reasons for the changes (noted as “why”) through a survey

<sup>1</sup><https://openai.com/blog/chatgpt/>

of both academic papers and developer forums and validate the standards with experienced OSS developers. On top of that, they propose three classification models that can automatically identify and construct a high-quality commit message dataset. Additionally, some researchers have conducted studies based on Tian’s understanding of what constitutes a good commit message. For instance, Li et al. [18] develop a machine learning classifier to automatically identify whether a commit message contains “What” and “Why” information, taking into account both the commit messages and the issue report/pull request links. In summary, it is commonly acknowledged that a good commit message should explain what was changed, and why a change was made.

### C. Large Language Models on Code

Recently, a number of Large Language Models (LLMs) that are pre-trained on source code have been proposed, which mainly consist of three categories from the perspective of model structure, i.e., encoder-only models, decoder-only models, and encoder-decoder models.

(1) **Encoder-only models** contain an encoder only and are normally pre-trained with a series of code comprehension tasks, such as masked language modeling [38] and replaced token detection [39], leading to their powerful capability in code representation. Typical examples include CodeBERT [40] and GraphCodeBERT [41]. (2) **Decoder-only models** are pre-trained with the objective of next-token prediction language modeling in an unsupervised fashion [42], [43], where GPT [44]–[46] series in the Natural Language Processing (NLP) field are successful paradigms. To this end, many decoder-only models optimized for code have been proposed based on the similar idea, such as LLaMA [47], CodeX [48], GPT-CC [49], and CodeGen [50], which can be utilized in generation tasks. (3) **Encoder-decoder models** are composed of an encoder and a decoder, which are typically pre-trained with denoising-based tasks. For example, CodeT5 [21] is pre-trained with tasks of identifier tagging, masked identifier prediction, and bimodal dual generation. UniXCoder [36] and PLBART [51] are pre-trained with the denoising sequence-to-sequence modeling task. Due to their encoder-decoder structure, tasks of both representation and generation can be successfully applied.

## III. PILOT ANALYSIS

Previous studies [5], [16], [17], [19] have confirmed that the quality of manually written commit messages on Version Control Systems (VCS) is generally poor. However, mainstream commit message datasets, such as the MCMD, have not yet been verified for quality issues, nor has there been research on cleaning and curating a high-quality dataset. This pilot analysis aims to conduct a series of quantitative and qualitative studies based on sampling to examine whether the MCMD dataset impacts the objective evaluation of commit message generation models and whether it requires cleaning.

### A. Experimental design

1) *Data set preparation*: This study employs a Multi-programming-language Commit Message Dataset (MCMD)

for experimentation, which has been extensively applied in previous CMG studies [2], [3], [5]. The MCMD dataset covers five major Programming Languages (PLs), including Java, Python, JavaScript, C++, and C#. To control experimental costs while ensuring a statistically valuable evaluation, we sample data points from the MCMD test set for Java, Python, and JavaScript, using a 95% statistical confidence and a 5% confidence interval method, focusing our evaluation on these three prevalent programming languages. Detailed sample volumes of each PL are shown in Table I. This design enhances the statistical significance of experimental results, strengthening the research’s empirical foundation.

TABLE I  
DATA VOLUME BEFORE AND AFTER SAMPLING

PL	Test Sets Volume	Sampling volume
Java	20159	376
Python	25837	378
JavaScript	24773	378

2) *Experimental Model Preparation*: In this preliminary experiment, we utilize a state-of-the-art LLM, i.e., GPT-3.5, for experiments. GPT-3.5 is a decoder-only LLM, pre-trained on a huge amount of human-written text/code, capable of capturing complex language patterns and contextual information. In the experiment, we adopt the version of gpt-3.5-turbo and set the following model parameters: *max\_tokens* is set to 50 for controlling the length of generated messages. *temperature* and *top\_p* are set to 0.8 and 0.95 by default, respectively. Besides, we design a basic prompt for GPT-3.5 and obtain its first output for evaluation. The basic prompt can be formally defined as:

“`{Code_Diff}`\nYou are a programmer who makes the above code changes. Please write a commit message for the above code change.”

where `{Code_Diff}` is the placeholder for a code diff snippet. The generated results are collected and compared with those produced by the state-of-the-art CMG approach, i.e., RACE [2] in terms of automated assessment metrics mentioned in III-A3.

3) *Automated assessment*: Following the previous studies in the CMG field [2], [7], [13], we adopt METEOR, BLEU, and ROUGE-L to assess the performance of each model.

- METEOR [52]: This metric measures the word-level matching between machine-generated text and reference text. METEOR considers several factors, including exact word matches, synonym matches, and stemming (morphological variations). It also accounts for word order by incorporating a penalty for misaligned words.
- BLEU [53]: The computation of this metric relies on the overlap of n-grams (contiguous sequences of n words) between the generated text and one or more reference texts.
- ROUGE-L [54]: This metric focuses on the longest common subsequence between the generated text and the reference text, which measures the longest sequence of words that appears in both texts in the same order.

Element	Explanation	Example
<b>What</b>	Describe the changes, features, or fixes introduced by the commit. This includes clear and concise descriptions to enable other developers to quickly understand the purpose of the commit.	Added user authentication feature / Fixed page loading speed issues / Optimized database queries etc.
<b>Why</b>	Explain the reasons and purposes behind the changes introduced by the commit. Provide context to help other developers understand why this specific change was necessary.	Addressed a security vulnerability / Improved user experience / Adapted to new business requirements etc.

Fig. 1. Two Key Elements Included in A Good Commit Message.

### B. Experimental results and analysis

Comparing the results between GPT-3.5 and RACE on the sampled dataset, we find that GPT-3.5 does not outperform RACE regarding automated assessment metrics as shown in Table II, which contradicts the usual impression reflected in the literature concerning LLM-applied code intelligence [10], [11], [55]. To delve deeper into the reasons behind this abnormal result, we introduce manual evaluation. Specifically, we select commit messages generated by GPT-3.5 with BLEU scores lower than 50% of those generated by RACE for human analysis and strictly follow the criterion of good commit messages to proceed with the assessment, thereby exploring why GPT-3.5 does not perform as well as we expected.

TABLE II  
PERFORMANCE COMPARISON BETWEEN GPT-3.5 AND RACE ON THE SAMPLED DATASET

Model	Java			Python			JS		
	BLEU	Met.	Rou.	BLEU	Met.	Rou.	BLEU	Met.	Rou.
RACE	<b>25.66</b>	<b>15.46</b>	<b>32.02</b>	<b>21.79</b>	<b>14.68</b>	<b>28.35</b>	<b>25.55</b>	<b>16.31</b>	<b>31.79</b>
GPT-3.5	18.94	12.79	15.27	17.85	8.88	11.36	15.56	8.31	12.00

According to the widely acknowledged definition in previous studies [14], [18], [19], a good commit message contains two key elements of “What” (i.e., what was done) and “Why” (i.e., why it was done). The “What” part reveals the specific changes made in the code, such as fixing a bug, adding a new feature, and improving existing code. The “Why” section provides background information on the changes, explaining why such modifications are necessary, such as enhancing performance, addressing security vulnerabilities, or improving user experience. Fig. 1 explicitly describes the purpose and reasons behind commits.

During the manual assessment, we select the results of GPT-3.5 and ground truths for comparison, thereby analyzing the specific areas where GPT-3.5 falls short. To ensure a thorough analysis, we use a structured approach involving a diverse group of evaluators (including the authors of this article and additional programmers with 3-5 years of development experience). Each evaluator is assigned a set of ground truth and GPT-3.5-generated commit message pairs for assessment. To ensure the accuracy and reliability of the assessment results, we adopt a dual assessment mechanism [56] whereby two independent evaluators assess each commit message together.

Subsequently, we use the Kappa consistency test [57] to measure the agreement of the assessment results between the two evaluators. For each ground truth and GPT-3.5-generated commit message sample, evaluators assess the following two aspects:

**Element 1: Existence of “What” Content (0/1):** Check whether the generated statements clearly present a description or expression of what has been done.

**Element 2: Existence of “Why” Content (0/1):** Examine whether the generated text provides the reasons or background for why this was done.

In the manual assessment, the Kappa score of the assessment results was 0.65, indicating a high degree of consistency between the evaluators, thus demonstrating the accuracy and reliability of our assessment method. The specific evaluation results are shown in Fig. 2, where each bar denotes a comparison between GPT-3.5-generated commit messages and ground truths in terms of the percentage of their commit messages containing or missing “What”/“Why” information. For example, GPT-3.5-Java represents the evaluation result of commit messages for the Java PL generated by GPT-3.5.

The research results indicate that some ground truth commit messages themselves are not entirely accurate or detailed. Many ground truth messages lack descriptions of “what” and “why,” while the GPT-3.5 demonstrates a relatively superior ability to generate these contents. This implies that, from the perspectives of “what” and “why,” GPT-3.5-generated commit messages are more informative and of higher quality compared with ground truths. Therefore, the ground truths of the MCMD dataset are inherently flawed because their samples are extracted from VCS, such as Github, in the wild, which cannot accurately reflect the capabilities of CMG approaches. Considering most of the previous CMG approaches are evaluated on the MCMD dataset [2], [3], [6], [23], [33], [34], and there is no other widely acknowledged multi-lingual CMG dataset, it is urgent to clean this dataset up and reassess previous mainstream CMG approaches, including LLMs.

### ✎ Conclusion of Pilot Analysis:

- (1) The MCMD dataset, widely used in the CMG tasks, can not accurately reflect the current capabilities of the CMG approaches due to the low data quality.
- (2) There is an urgent need for a thorough cleanup of the MCMD dataset and reassessment to ensure more objective and precise evaluations of the CMG techniques in the following sections.

## IV. CONSTRUCTION OF THE HIGH-QUALITY TEST SET

As mentioned above, the quality of ground truths significantly affects the objectivity and authenticity of assessment. Therefore, a pivotal step in this study is to construct a meticulously curated, higher-quality, and reliable CMG dataset covering mainstream PLs for assessing and validating the CMG performance of diverse models systematically. Considering the trade-off between limited human effort and assessment validity, we meticulously curate 500 high-quality samples for

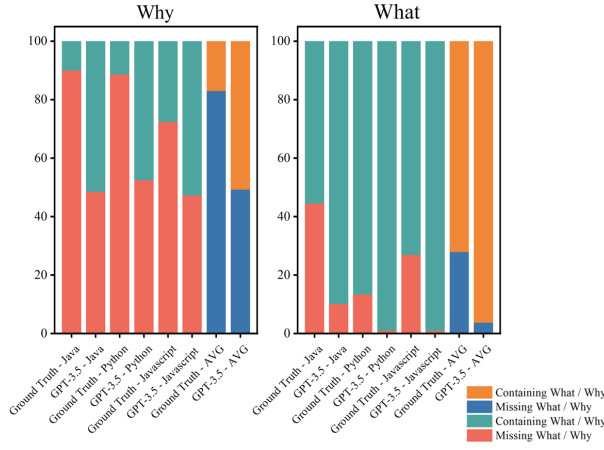


Fig. 2. The Proportion (%) of Containing/Missing “What”/“Why” Elements in Comparison between GPT-3.5-Generated Commit Messages and Ground Truths.

each PL from the MCMD test set. Throughout this process, to improve the efficiency of sample selection as much as possible, we conduct meticulous three-stage screening of commit messages in all PLs (including Java, Python, JavaScript, C++, C#) covered in the MCMD dataset. We hope the newly constructed test set can not only be used to evaluate the performance of CMG models but also serve as reference templates for developers to enhance code quality and collaboration efficiency within teams. This section outlines the three-step screening procedure for curating this test set, including (1) filtering out non-target languages, (2) automatically filtering good commit messages through a deep learning model, and finally (3) further selecting high-quality samples manually.

#### Step 1: Precise Filtering of Non-Target PL samples.

Based on our observations, although the MCMD dataset is divided into five categories according to PLs, each sub-dataset of a PL is mixed with a large amount of data from other PLs. For instance, when dealing with a sub-dataset targeting Java, we encounter the inclusion of many code *diffs* from other PLs, such as C++, Python, and even HTML. This issue arises because PLs are simply partitioned by projects according to the explanation of the original authors, which severely affects the accuracy of previous studies’ experimental results under different PLs and poses a major obstacle to the evaluation in this paper.

To tackle this problem, we employ regular expressions as the primary filtering tool, where files with suffixes (e.g., .java, .py, .js, .cpp, .cs) that do not belong to the target PL are filtered out, thereby completing the first-step cleaning. Through this approach, we significantly enhance the PL purity of each sub-dataset, establishing a sturdy and precise basis for subsequent data analysis and model assessment. The data volume after this first-step cleaning is shown in Table III, over half of the non-target samples for each PL have been eliminated, effectively reducing the human effort in the following manual selection step.

#### Step 2: Automated Filtering of samples Containing “What” and “Why”.

TABLE III  
NUMBER OF DATASETS AFTER FILTERING AT EACH STEP

PL	Original	Step1	Step2	Step3	Kappa
Java	20159	11746	4167	500	0.59
Python	25837	14012	4601	500	0.56
C#	18702	9224	3432	500	0.62
C++	20141	5778	2029	500	0.57
JavaScript	24773	10816	3855	500	0.61

TABLE IV  
PERFORMANCE OF BI-LSTM IN OUR STUDY

Metrics	C-Why	C-What	C-Good
Precision	44.60%	85.37%	68.56%
Recall	52.03%	34.80%	51.34%
F1	48.03%	47.01%	58.47%

In our study, we recognize that manually filtering out commit messages to identify those containing both “What” (what was done) and “Why” (why it was done) elements is time-consuming and labor-intensive. To address this issue, we introduce an automated method based on Bi-LSTM [58] to efficiently identify high-quality commit messages, which was proposed by [19] and adhered to the definition of “good commit message” during the identification.

Since Tian et al. [19] did not publish the trained parameters, we try our best to replicate this Bi-LSTM model following their instructions and train the model from scratch. Afterward, we apply it to our commit messages following their preprocessing procedures such as text cleaning, vocabulary standardization, and encoding.

The replication results are shown in Table IV, where “C-Why”, “C-What”, and “C-Good” represent the performance of Bi-LSTM on identifying commit messages containing “Why”, “What”, and both elements. Although the replicated performance is slightly weaker than that reported in the original paper [19], it still effectively filters out those high-quality commit messages to a great extent. After the filtering process, approximately one-third of the data remained from the step-1 filtering, as shown in Table III, which further improves the density of high-quality samples and is helpful in reducing the efforts of manual selection in the third step.

#### Step 3: Manual Selection of Samples with “What” and “Why”.

After the automated filtering process, we proceeded to the stage of manual assessment, aiming to select high-quality commit messages from the remaining samples precisely. To ensure the accuracy and objectivity of the selection, we purposely selected individuals with 3-5 years of programming experience to serve as evaluators. The specific process of manual selection is as follows:

We employ a dual-evaluator mechanism, dividing evaluators into pairs and conducting independent evaluations in a back-to-back manner [56] to minimize mutual influence on opinions. Each evaluator assigns a score of 0 or 1 to each commit message. Here, a score of 0 indicates that the commit lacked either the “What” or “Why” element, while a score of 1 indicates

that the commit contained a clear and detailed description of “What” and “Why.” When both evaluators provide consistent scores (either both 0 or both 1), deciding whether to retain or discard the commit is simple: discard for all 0s or retain for all 1s. In instances of conflicting scores, a third evaluator will be brought in for arbitration.

To validate the effectiveness of the assessment, we compute Cohen’s kappa coefficient [57] of agreement for the scores assigned by the two evaluators. This statistical measure is employed to assess the Consistency between evaluators, with a high kappa score indicating a higher level of reliability in the evaluation results. The kappa scores for each language are calculated as shown in Table III. In our case, the overall mean value of the kappa score is 0.59, which shows that our assessment process is effective.

Through such screening, we ultimately construct a high-quality, multi-lingual test set comprising 500 commit messages for each of the five PLs. The commit messages in the test set are comprehensive and explanatory, with each one being unanimously recognized by evaluators as an outstanding commit message that meets our quality standards. We have open-sourced this dataset on GitHub. The data distribution for each language before and after each cleaning and filtering step is presented in Table III.

## V. STUDY DESIGN

This section details the design of specific experiments, including studied models, implementation, and evaluation methodology.

### A. Models and Implementations

In this study, we select the most widely used and state-of-the-art CMG approaches for a systematic comparison with a series of recent LLMs of diverse sizes and families.

- NNGen [3]: A retrieval-based approach employs the nearest neighbor algorithm to fetch the top-k similar code differences, determined by cosine similarity between bag-of-words vectors of code differences. It then selects the most similar result based on BLEU scores between each of them (top-k results) and the input code *diffs* to generate commit messages.
- CommitGen [34]: A model that treats code diffs as plain texts and adopts a Seq2Seq neural network with different attention mechanisms to translate them into commit messages.
- NMTGen [33]: In this context, NMTGen is used to learn the semantics of code *diffs* and translate them into commit messages.
- PTrGNCMSG [6]: A model that incorporates the pointer-generator network into the Seq2Seq model to handle out-of-vocabulary words.
- CoRec [23]: A hybrid model that considers retrieved results during the inference stage. It uses an encoder-decoder neural model to encode input code diffs during training and generate commit messages. At the inference stage, it uses the trained encoder to retrieve the most similar code *diff* from the training set, then reuses a trained

encoder-decoder to encode the input and retrieved code *diff*, combining the probability distributions (obtained by two decoders) of each word to generate commit messages.

- CodeBERT [35]: Leverages CodeBERT, a pre-trained language model for source code, to learn the semantic representations of code *diffs* and adopts a Transformer-based decoder to generate commit messages.
- LLaMA [47]: An open and efficient large foundational language model released by Meta AI. LLaMA has four versions with parameter sizes of 7B, 13B, 33B, and 65B. These models can be used for various natural language processing tasks, including code-related tasks, making them suitable for commit message generation tasks. In this article, we use both llama-7b-chat and llama-13b-chat. They can generate accurate commit messages by learning the semantics of code differences.
- Gemini [59]: An artificial intelligence model released by Google DeepMind, capable of handling text, images, audio, video, and code. Gemini can understand and generate high-quality code in mainstream PLs and provides comprehensive security assessments. We use the Gemini-pro model for this article. Its deep understanding of code enables it to play a role in commit message generation tasks, producing accurate and descriptive commit messages.
- GPT-3.5 [46]: A model based on the GPT-3 architecture, performing exceptionally well in various natural language processing tasks. GPT-3.5’s robust language understanding and generation capabilities make it suitable for commit message generation tasks, where it can generate accurate and descriptive commit messages by understanding the context of code changes. The same as the pilot analysis, we adopt the version of gpt-3.5-turbo for experiments.

For the implementation of LLMs, we follow the hyper-parameter setting in Section III-A2, while for other state-of-the-art CMG approaches, we extract their published CMG results for comparison, as their experiments were all conducted on the MCMD dataset. As such, the assessment among LLMs and other CMG approaches is fair.

### B. Evaluation Methodology

To achieve a systematic evaluation, we first adopt automated assessment in terms of METEOR, BLEU, and ROUGE-L, as mentioned in Section III-A3, to make comparisons among current CMG approaches and recent LLMs. This solves the problem: **RQ1: How is the performance of recent LLMs against current CMG approaches?**

To delve deeper into the LLMs’ performance in the CMG task, we further arrange a series of manual assessments. Aligning with the practice of OSS, we not only focus on the existence of “What”/“Why” elements (i.e., Integrity) for LLM-generated commit messages but also involve Accuracy, Applicability, and Readability in the assessment, as these aspects impact the internal and external quality of the software product and their usability in OSS practice [60], [61].

Specifically, we employ three-level Likert scales which allows participants to express their degrees of positive or

TABLE V  
KAPPA SCORE OF MANUAL ASSESSMENTS

Language	Java	Python	C#	C++	JavaScript	Average
Kappa score	0.54	0.53	0.54	0.57	0.57	0.55

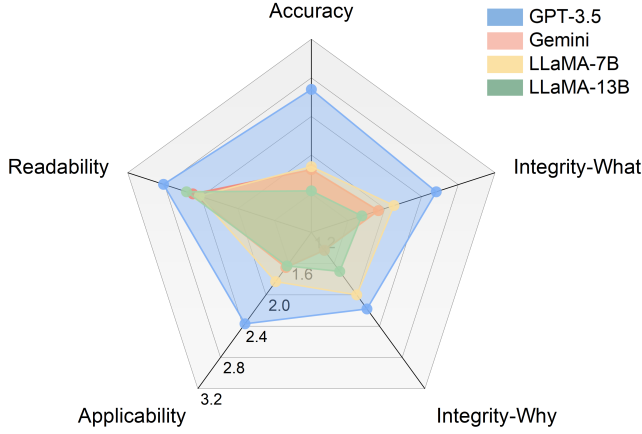


Fig. 3. An Overview of Various LLMs' Performance Among Different Aspects of Manual Assessment

negative attitudes, from “disagree” to “neutral” to “agree,” or from “poor” to “neutral” to “good” [62] in expert interviews to collect high-quality feedback from participants with experience and professional backgrounds based on their experience and observations. Then we design five survey questionnaires corresponding to the five PLs studied, inviting two industry experts for each language to participate in interviews. Each questionnaire comprises ten test items, exploring the performance of different LLMs in generating commit messages through various problem descriptions, including Accuracy, Integrity, Applicability, and Readability. This design facilitates a comprehensive understanding of participants' perceptions across different aspects, offering multi-dimensional and comprehensive qualitative feedback.

Then, we obtain 10 interview results, covering five PLs. To validate the evaluation's effectiveness and the questions' universality, we separately calculated Cohen's kappa consistency coefficients between the scores given by the two experts for each language. Since each evaluator is required to evaluate the full range of questions for each PL, the Kappa Score is calculated by PLs, which are presented in Table V. It can be observed that Cohen's kappa consistency coefficients for each PL are all larger than 0.5, indicating a high level of Consistency. We list a radar chart in Fig. 3 to present the average performance of various LLMs among different aspects of the human evaluation. This manual assessment addresses the following research questions RQ2-RQ5:

**RQ2: How accurate is the commit message generated by LLMs?** Different from the automated metric mentioned above, Accuracy in manual assessment reflects more on the semantic equivalency, which is highly important for developers to understand the code changes. Human evaluators can assess

the relevance and adequacy of commit messages beyond strict literal Consistency, considering the contextual and semantic nuances that automated metrics may overlook.

Towards the Accuracy aspect, evaluators are required to examine whether the commit messages accurately reflect the specific technical details of code changes and analyze whether the commit messages are closely associated with the content and purpose of the code changes, ensuring that no vital change information is omitted and no irrelevant information is included.

**RQ3: How integral is the commit message generated by LLMs?** Integrity measures whether LLM-generated commit messages contain “What” and “Why” elements, which are hard to precisely detected in an automated manner. Commit messages implying clear modification illustration and motivation allow for an easier understanding of code changes, simplify the code review process, and aid debugging in OSS practice.

Evaluators should assess how well a commit message describes the code changes (“What”) and clarify the reasons or motivations behind them (“Why”). Evaluators assign corresponding scores based on the thoroughness of the commit messages.

**RQ4: How applicable is the commit message generated by LLMs?** Applicability assesses to what extent OSS developers accept to use the LLM-generated commit messages. As commit messages are critical documentation for code changes, aiding in code review, version control, and team communication, evaluating their Applicability helps determine whether LLM-generated commit messages can effectively fulfill these roles and contribute to a smoother and more efficient OSS development process.

In order to understand individual behavior and decision-making in the Applicability assessment, we use an “immersive context” approach, a technique commonly used in sociological research [63]. Specifically, evaluators were guided to place themselves in a hypothetical role, assuming they were the programmers writing the evaluated information. We inquired with evaluators about their willingness to adopt the provided commit message if they had authored the code in question.

**RQ5: How readable is the commit message generated by LLMs?** The motivation for evaluating the Readability of commit messages generated by LLMs lies in the importance of clear and understandable documentation in software development. Readable commit messages enhance communication among team members, facilitate code maintenance, and contribute to effective project management.

Evaluators are required to evaluate commit messages based on factors such as fluency, clarity of structure, accuracy of expression, and correct use of grammar. This scoring reflects the Readability and comprehensibility of the commit message.

Through the process mentioned above, we were able to holistically assess the overall capabilities of LLMs in commit message generation, providing valuable insights and guidance for the field of software development.

TABLE VI  
PERFORMANCE COMPARISON OF LLMs AGAINST STATE-OF-THE-ART CMG APPROACHES

Model		Java			Python			C#			C++			JS			Avg.		
		Met.	BLEU	Rou.	Met.	BLEU	Rou.	Met.	BLEU	Rou.	Met.	BLEU	Rou.	Met.	BLEU	Rou.	Met.	BLEU	Rou.
IR-based	NNGen	14.28	22.23	18.85	8.66	20.58	15.82	12.41	21.54	17.96	7.82	16.53	13.38	10.60	19.84	16.25	10.75	20.14	16.45
	CommitGen	2.38	3.77	6.89	1.71	4.01	6.94	1.92	3.87	4.83	1.12	2.91	4.38	1.98	3.83	6.12	1.82	3.68	5.83
End-to-end	NMTGen	2.64	3.93	7.30	2.41	4.84	8.35	1.89	3.36	4.86	1.62	3.37	5.00	3.28	6.48	6.96	2.37	4.40	6.49
	PTGNCMSG	11.86	14.95	17.02	7.48	16.55	15.68	8.57	12.42	12.64	6.8	12.35	13.05	8.05	13.46	14.21	8.55	13.95	14.52
Hybrid	CoRec	6.42	10.30	12.04	4.85	10.49	12.32	4.90	8.41	9.80	2.79	6.58	7.83	5.24	9.66	10.46	4.84	9.09	10.49
	RACE	11.54	21.41	<b>19.08</b>	10.61	23.75	<b>21.00</b>	12.11	22.74	<b>20.46</b>	6.66	16.01	12.57	11.09	21.41	<b>19.23</b>	10.40	21.06	<b>18.47</b>
Pre-Trained	CodeBERT	9.59	14.01	16.80	7.66	16.56	17.73	7.72	12.46	14.38	4.51	11.23	9.87	7.58	13.12	15.72	7.41	13.48	14.90
LLM	LLaMA-7B	19.43	23.90	14.01	14.71	20.94	12.14	17.44	23.23	15.01	15.10	20.43	12.61	17.65	22.88	14.90	16.87	22.28	13.73
	LLaMA-13B	17.09	25.97	15.10	13.88	24.80	14.04	16.27	25.39	15.52	13.21	23.04	13.04	15.47	24.32	14.70	15.18	24.70	14.48
	Gemini	16.53	26.16	17.62	11.87	24.96	14.34	15.33	11.01	6.30	12.76	24.20	15.13	13.76	24.78	15.97	14.05	22.22	13.87
	GPT-3.5	<b>21.19</b>	<b>27.72</b>	19.04	<b>18.00</b>	<b>26.86</b>	17.80	<b>20.3</b>	<b>27.25</b>	19.10	<b>17.05</b>	<b>24.99</b>	<b>16.70</b>	<b>18.94</b>	<b>27.15</b>	19.04	<b>19.10</b>	<b>26.79</b>	18.34
		<b>↑48.39</b>	<b>↑24.70</b>	↓-0.21	<b>↑69.65</b>	<b>↑13.09</b>	↓-15.24	<b>↑63.58</b>	<b>↑19.83</b>	↓-6.65	<b>↑118.03</b>	<b>↑51.18</b>	<b>↑24.81</b>	<b>↑70.78</b>	<b>↑26.81</b>	↓-0.99	<b>↑83.85</b>	<b>↑27.20</b>	↓-0.72

## VI. RESULTS AND DISCUSSION

### A. RQ1: How is the performance of recent LLMs against current CMG approaches?


Table VI re-evaluates the performance of state-of-the-art CMG approaches of diverse categories against various recent LLMs on our cleaned multi-lingual CMG test set.

As observed, NNGen, a purely retrieval-based model, outperforms most learning-based CMG approaches and even performs neck-to-neck with RACE, the latest state-of-the-art CMG approach, which comes out with an opposite conclusion to prior studies [2]. A plausible explanation is that learning-based approaches are deviated by low-quality training samples in the MCMD dataset, which neglect the expression of “What” and “Why” elements when generating commit messages. Conversely, similar code *diffs* are likely to have identical commit messages; thus, those filtered high-quality test examples are more likely to find high-quality similar examples in the training set. As a result, NNGen, a purely retrieval-based CMG approach, surpasses the performance of many learning-based CMG approaches, underscoring the importance of similar exemplars in the CMG task.

Furthermore, when comparing CMG methods with LLMs, LLMs exhibit a substantial advantage in multiple metrics. Because LLMs, such as GPT-3.5, possess a more significant number of parameters and are trained on much more extensive human-written code/text, which enables them to demonstrate more robust performance even without specialized training for specific tasks. However, although RACE has fewer parameters than LLMs, it surpasses most LLMs in terms of the ROUGE metric and performs neck-to-neck with GPT-3.5 on this metric, demonstrating the powerful ability of RACE to generate high-quality commit messages in a long range of continuous sequences. Focusing on the comparison between GPT-3.5 and RACE, we find that GPT-3.5 significantly outperforms RACE, the latest state-of-the-art CMG approach, with an improvement of 83.85% in terms of METEOR and 27.20% in terms of BLEU, which completely opposite to the experimental results in the pilot analysis, showing that the original MCMD dataset indeed distorts the performance evaluation among models and the construction of the cleaned test set is necessary.

In the comparison among LLMs, GPT-3.5 surpasses other LLMs in most evaluation metrics, which may be attributed to its larger number of parameters. Regarding the comparison

between GPT-3.5 and Gemini, the conclusions of this study are consistent with those of Akter et al. [64], indicating that GPT-3.5 has a relatively clear advantage in tasks related to coding. To be specific, in terms of the METEOR metric, GPT-3.5 leads by an average of 13.21%-35.94% compared to other LLMs. In the BLEU metric, GPT-3.5’s average lead ranges from 8.46% to 20.57%. Furthermore, in the ROUGE-L metric, GPT-3.5 maintains an average lead of 26.66%-33.57% over its counterparts.

 **Answer to RQ1:** LLMs have demonstrated impressive performance in the CMG domain, overall surpassing all current CMG approaches, with GPT-3.5 leading among LLMs, particularly on our cleaned high-quality CMG test set.

### B. RQ2: How accurate is the commit message generated by LLMs?

Fig. 4 demonstrates the manual assessment results in terms of Accuracy, where GPT-3.5 still performs the best among different LLMs. In particular, GPT-3.5 has an average score of 2.68 in the question, while the other three LLMs have mean scores of 1.85, 1.88, and 1.63, respectively. Compared with the automated metrics, manually evaluated Accuracy tends to measure the overall semantic equivalency instead of literal equivalency, showing that GPT-3.5 has an outstanding ability to capture the entire semantics of code *diffs*. In addition, we note that LLaMA-13B outperforms LLaMA-7B overall according to Table VI, yet it underperforms LLaMA-7B in terms of the aspect of Accuracy during the manual evaluation. Considering the focus on the semantic equivalency of the Accuracy, we conjecture that LLaMA-7B is superior in generating high-quality commit messages from the perspective of overall semantics, while LLaMA-13B prefers to generate commit messages with higher literal accuracy.

Taking a Python code *diff* as an example, we list different LLMs’ generated commit messages in Figure 5, it can be observed that among the commit messages generated by the four LLMs, the commit message generated by GPT-3.5 directly and clearly emphasizes the core content of the code change, i.e., ensuring that the message history is returned in list form. It

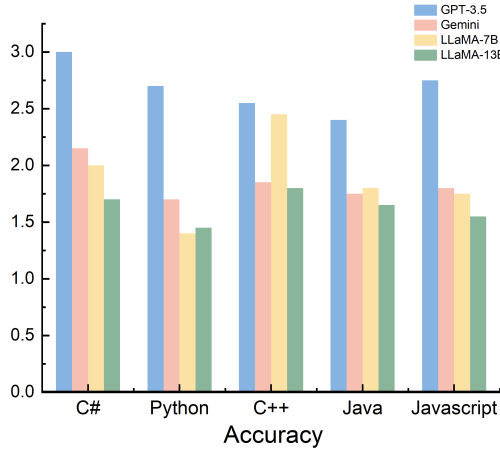


Fig. 4. Manual Assessment Results in Terms of Accuracy

Code Diff	
Python	
<pre>def get_message_edit_history(request: HttpRequest, user_profile: UserProfile,                              # Fill in all the extra data that will make it usable                              fill_edit_history_entries(message_edit_history, message)):     - return json_success({"message_history": reversed(message_edit_history)})     + return json_success({"message_history": list(reversed(message_edit_history))})  PROPAGATE_MODE_VALUES = ["change_later", "change_one", "change_all"]  @has_request_variables # Continue with the rest of your code</pre>	
Model	Commit Message
Ground Truth	message_edit: output a list, not a reversed iterator.
GPT-3.5	[message_edit_history] Ensure message history is returned as a list.
Gemini	Allow to return the message edit history in reverse chronological order.
LLaMA-7B	Added functionality to display message edit history in reverse order.
LLaMA-13B	Added support for retrieving message edit history with fill_edit_history_entries and returned in reversed order.

Fig. 5. A Python Example of Generated Commit Messages on Accuracy.

conveys the essence of the code change without introducing unnecessary information or incorrect statements, aligning more closely with the original code change situation. Looking at the commit messages generated by the other three LLMs for this code change, we find that these commit messages only mention “return in reverse order” and do not explicitly state the change key of “return in the form of a list.”

**Answer to RQ2:** Manual assessment has demonstrated that GPT-3.5 exhibits the best performance among all LLMs in terms of Accuracy in the CMG task, showcasing its exceptional ability to capture technical details and accurately reflect code changes.

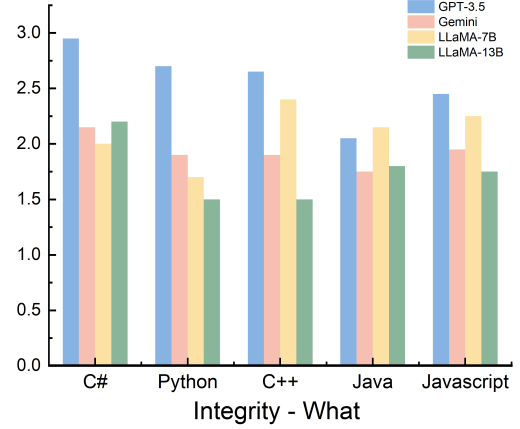
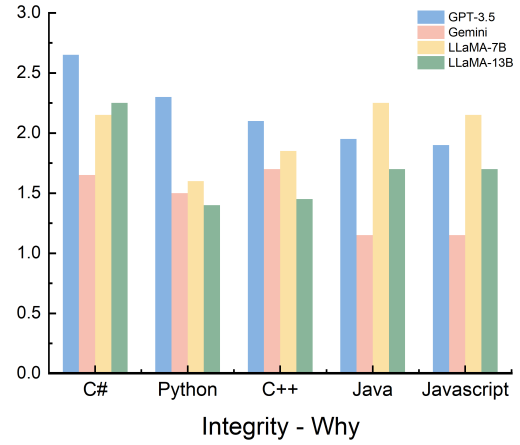


Fig. 6. Manual Assessment Results in Terms of Integrity



### C. RQ3: How integral is the commit message generated by LLMs?

Fig. 6 demonstrates the Integrity concerning “What” and “Why” information of the commit messages generated by different LLMs. Firstly, we focus on evaluating commit messages in the “What” aspect. As can be seen, GPT-3.5 excels with a mean score of 2.56, while the other three LLMs have mean scores of 1.93, 2.1, and 1.75, respectively. This indicates that commit messages generated by GPT-3.5 are often more detailed and comprehensive, reflecting the actions the code changes have taken. Fig 7 demonstrates an example of a code *diff* in C#, and it is observed that among the commit messages generated by the four models, GPT-3.5-generated commit messages contain more precise and more explicit “what” information. It provides an apparent description of the essence of the code change, namely “add debug assertion” in the code. At the same time, the commit messages from other models are relatively vague or simplified.

Secondly, we examine the evaluation of commit messages in the “Why” aspect. Apparently, GPT-3.5 also performs exceptionally well, with a mean score of 2.18, while the other three LLMs have mean scores of 1.43, 2.1, and 1.7, respectively. This indicates that commit messages generated

Code Diff	
C#	
<pre>private static bool SemiColonIsMissing(SyntaxNode currentNode) {     case SyntaxKind.AddAccessorDeclaration:         return ((AccessorDeclarationSyntax)currentNode).SemicolonToken.IsMissing;     default:         + // At this point, the node should be empty or its children should not end with a         + // semicolon.         + Debug.Assert(!currentNode.ChildNodesAndTokens().Any()         +    !currentNode.ChildNodesAndTokens().Last().IsKind(SyntaxKind.SemicolonToken));         return false; }</pre>	
Model	Commit Message
Ground Truth	add an assertion that semi colon is missing explicitly lists all cases.
GPT-3.5	Add debug assertion to check if a node is empty or its children do not end with a semicolon.
Gemini	Add a check to see if a semicolon is missing in accessor declarations.
LLaMA-7B	Fixes issue where semicolon was missing in add accessor declaration syntax node.
LLaMA-13B	Fix: ensure semi-colon is present in add accessor declaration svntax nodes.

Fig. 7. A C# Example of Generated Commit Messages on Integrity.

by GPT-3.5 more clearly explained the reasons and motivations behind the code changes. Using the aforementioned C# language test question as an example, among the commit messages generated by the four models, GPT-3.5-generated commit messages contain more explicit “why” information, i.e., “check if a node is empty or its children do not end with a semicolon,” while the “why” information in the commit messages generated by the other LLMs are relatively vague (as shown in Fig. 7). This also indicates GPT-3.5’s significant advantage in understanding and expressing the motivation behind code changes, making practitioners collaborate with their team members more smoothly in OSS development.

It is noteworthy that all LLMs examined generally score higher in terms of the “what” aspect than the “Why” aspect when generating commit messages. This phenomenon may stem from the limitations of the code snippets themselves. Actually, in development practice, code snippets rely on each other closely and have their own specific functionalities, which can only be reflected when they are placed in the whole repository. Nonetheless, code *diff* samples in the MCMD dataset only comprise fragments. In this case, although LLMs can easily understand the specific content of code changes, they can hardly capture the motivation behind code *diff*s owing to the lack of adequate context information of the whole repository.

**Answer to RQ3:** GPT-3.5 excels in generating commit messages with Integrity, significantly outperforming other LLMs. It provides more detailed and comprehensive descriptions of code changes from both “What” and “Why” perspectives. All LLMs perform relatively weaker in expressing “Why” information in commit messages owing to the limited code context.

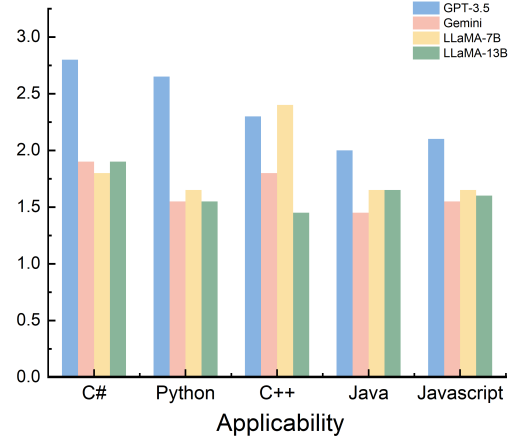


Fig. 8. Manual Assessment Results in Terms of Applicability

#### D. RQ4: How applicable is the commit message generated by LLMs?

Fig. 8 illustrates the manual assessment results concerning the Applicability of different LLMs by PLs, indicating that GPT-3.5 excels in Applicability compared to Gemini, LLaMA-7B, and LLaMA-13B. Among these models, the mean scores were 2.37 for GPT-3.5, 1.65 for Gemini, 1.83 for LLaMA7B, and 1.63 for LLaMA13B. We continue to take Python code *diff* as an example, as shown in Fig. 5. Compared to other LLMs, the commit messages generated by GPT provide clear, concise, professionally accurate descriptions that accurately convey the core content and purpose of the code changes. This clear and concise description helps evaluators swiftly grasp the purpose and impact of the code changes, leading to greater acceptability among them. In contrast, commit messages generated by other models tend to be more redundant and less straightforward in conveying information.

GPT-3.5 stands out for its Applicability, producing commit messages that align closely with programmers’ requirements and are more apt for real-world usage. This reflects its exceptional performance in comprehending and expressing language. This finding has practical implications for developing automated code commenting tools, aiding in enhancing development efficiency and alleviating the burden on programmers.

**Answer to RQ4:** Compared to other LLMs, GPT-3.5 demonstrates higher Applicability in generating commit messages. Its concise and accurate descriptions effectively convey the details of code changes, thereby gaining higher acceptance among evaluators.

#### E. RQ5: How readable is the commit message generated by LLMs?

Fig. 9 demonstrates the Readability assessment for diverse LLMs, where GPT-3.5, Gemini, LLaMA-7B, and LLaMA-13B have mean scores of 2.81, 2.49, 2.42, and 2.56, re-

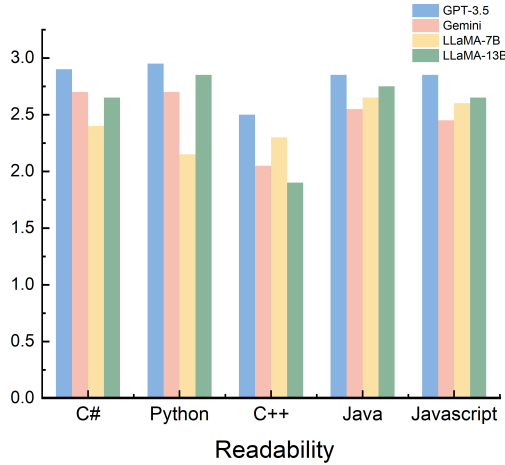


Fig. 9. Manual Assessment Results in Terms of Readability

spectively, showing that GPT-3.5 exhibits a certain degree of superiority in this aspect. Simultaneously, we observe that, among all test dimensions, Readability scores are generally higher and distributed relatively evenly compared to scores in other aspects mentioned before. GPT-3.5 has a slight advantage, while the other three LLMs perform comparably. Taking a Java code *diff* as an example (as shown in Fig. 10), one evaluator commented, “Overall, each commit message in this example is easy to understand, demonstrating good accuracy and readability. However, I particularly appreciate the commit messages generated by GPT-3.5, which succinctly and accurately address the specific issues being addressed. Specifically, GPT-3.5 articulates the generated text in a more natural manner, ensuring smooth expression. Moreover, from a structural perspective, commit messages generated by GPT exhibit a relatively clear structure, aiding in our understanding of the content and purpose of code changes. Additionally, GPT-3.5 demonstrates relatively good grammar, avoiding common syntactical errors. In contrast, compared to GPT-3.5, commit messages generated by Gemini lack accuracy, as they do not mention operations in the ‘onClose’ method. Commit messages generated by LLaMA-7B exhibit decent expression, yet they overlook crucial elements such as ‘error message’, resulting in expression deficiencies. LLaMA-13B-generated commit messages only mention ‘update’ regarding the change content, which is not precise enough and lacks specific descriptions of updates.”

This finding holds significant practical implications in OSS practice, as Readability directly affects programmers’ understanding and acceptance of code changes. Overall, although GPT-3.5 has a slight advantage in Readability, the differences among the four models are insignificant, indicating the powerful capabilities of recent LLMs in logical structure, grammar, and language expression when generating commit messages. This is crucial for the practical application of automatically generating commit messages and provides guidance for better integrating the application of LLMs into OSS practices in the future.

Code Diff	
<pre> Java public void onClose() {     @Override     public void onTimeout(TimeValue timeout) {         clusterService.remove(this);         - listener.onFailure(new MasterNotDiscoveredException());         + listener.onFailure(new MasterNotDiscoveredException(" waited for [" + timeout + "] "));     }     @Override </pre>	
Model	Commit Message
Groud Truth	better failure message when no master found ( how long we waited for it )
GPT-3.5	<b>Fixing the error message in the on close method of master not discovered exception.</b>
Gemini	Handle master not discovered exception with timeout details.
LLaMA-7B	Added timeout handling for on close method to prevent master not discovered exceptions.
LLaMA-13B	Update on close method to handle timeouts and provide more informative error messages.

Fig. 10. A Java Example of Generated Commit Messages on Readability.

**Answer to RQ5:** Manual assessments have found that LLMs perform well in terms of Readability. Among them, GPT-3.5 has a slight edge, reflecting its natural and fluent conveyance of code change information. This finding emphasizes the overall maturity and effectiveness of LLMs in generating readable commit messages.

## VII. ERICOMMITTER

In the previous six sections, we conduct an empirical study on the use of LLMs for the automatic generation of commit messages. Our evaluation, through both automated and manual analysis, highlighted the superiority of LLMs in producing accurate, relevant, and informative commit messages compared with state-of-the-art CMG approaches of diverse categories. In this section, considering the prominent efficacy of incorporating retrieved samples for references in CMG approaches (as mentioned in Section VI-A) and the extensive applicability of In-Context Learning (ICL) for LLMs [24], [65], [66], we carry an idea of retrieving similar samples towards the target code *diff* to construct ICL examples, thereby guiding LLMs’ generation and further improving their performance in the CMG task. However, the volume of the database for retrieval is exceptionally huge and full of noise. For example, each PL in the MCMD dataset contains over 200,000 samples, but not all samples are good candidates for retrieval, while the retrieval database in practice can be even larger. To accelerate the retrieval efficiency and make this approach practical, we propose a two-step filtering method to reduce the database volume before retrieving and ensure the CMG performance remains almost the same concurrently. To summarize, we introduce the Efficient Retrieval-based In-context Learning framework, named ERICommitter, suitable for different LLMs in the CMG field. Here’s how it works: (1) We employ a two-step filtering process to eliminate less informative and low-quality samples, constructing the reduced retrieval database. (2) Based on the reduced database, we retrieve similar samples

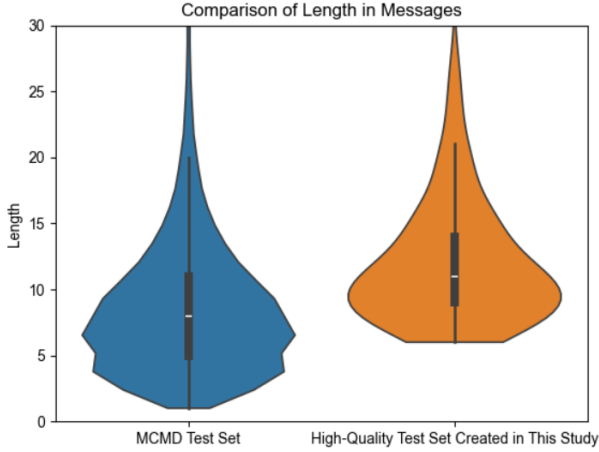


Fig. 11. Comparison of Length in Commit Messages.

to build ICL examples. (3) LLMs are guided to generate more accurate and informative commit messages via ICL.

#### A. Methodology

1) *Two-Step Filtering on the Retrieval Database*: Two-step filtering first harnesses the length of commit messages as a selection criterion to drop those less informative ones and then adopts Bi-LSTM to filter low-quality commit messages from a semantic perspective, thereby reducing the volume of the retrieval database.

**Length-based Filtering**: An intuitive hypothesis is high-quality commit messages are relatively longer (i.e., containing more tokens) because they include both “What” and “Why” information about code *diffs*, such as the specific functionalities, implementations, and underlying modification motivations. To substantiate this hypothesis, we plot violin diagrams to compare the distribution of token counts in the MCMD’s test set and our cleaned high-quality test set, as shown in Fig. 11. The distribution discrepancies in token counts between them are obvious; the cleaned high-quality test set carries more tokens for each commit message on average, showing that length-based filtering on the retrieval database is reasonable. In this experiment, we adopt the average length of the cleaned high-quality test set as the threshold to filter samples in the MCMD’s training set (i.e., retrieval database).

**Semantic-based Filtering**: Secondly, to further filter out high-quality commit messages, we again utilize the Bi-LSTM constructed in Section IV to capture and understand the critical elements of “What” and “Why” in commit messages, thereby effectively filtering out high-quality commit messages semantically.

These two filtering steps are applied to the original training set. The first step, based on the length of commit messages, eliminates overly short commit messages, and the second step uses the Bi-LSTM model to further filter out high-quality samples from a semantic perspective. Through these two filtering steps, we obtain a much smaller but relatively high-quality training set as the reduced retrieval database.

2) *Retrieval Method*: In this study, we investigate two commonly used retrieval methods in academia: lexical-based retrieval and semantic-based retrieval. These methods are used to select commit messages similar to the test samples from the training set.

**Lexical-based Retrieval**: uses the BM25 algorithm [67], a widely-used information retrieval function based on the bag-of-words model [68], evaluating the relevance between documents and queries by considering Term Frequency (TF) and Inverse Document Frequency (IDF). This method excels at keyword matching and is suitable for quickly retrieving documents related to specific query conditions (such as specific PLs or keywords) in extensive text collections.

**Semantic-based Retrieval** exerts a pre-trained code model, namely CodeReviewer, a Transformer-based encoder-decoder model designed with four pre-training tasks for the code review process. CodeReviewer takes code *diffs* as input and splits them into token sequences using the RoBERTa tokenizer. To enhance comprehension of the code *diff* format, we replace the special markers “-” and “+” in the code *diff* file, which represent line deletion and insertion, with the special tokens [DEL] and [ADD], respectively. Additionally, lines that remain unchanged, indicated by the absence of any marker, are denoted by the token [KEEP].

The output of the model includes token representations from the encoder and the generated token sequence from the decoder. In this study, we only use the pre-trained encoder and the last layer representation of the special token [CLS] to represent the vectorized code *diff*. We then measured the similarity between *diffs* by calculating the cosine angle between two [CLS] token representations, providing an effective method to assess the similarity between code changes. In experiments, we fix the number of retrieved examples to 1 by default, considering the efficiency issue.

#### B. Experimental Setting of ERICommitter

To investigate the effectiveness and efficiency of ERICommitter, we further propose RQ6-RQ8 and their evaluation procedures below.

**RQ6: How does ERICommitter perform against individual LLMs?** This RQ aims to examine the effectiveness of ERICommitter compared with individual LLMs tested. To make a fair comparison, we retrieve the most similar sample from the reduced retrieval database for ERICommitter while selecting one fixed example for individual LLMs. Due to computational overhead and resource constraints, we include two PLs (i.e., Java and Python) and two LLMs (i.e., GPT-3.5 and Gemini) for experiments, both semantic and lexical-based retrieval methods are assessed.

**RQ7: How efficient is ERICommitter?** This RQ aims to inspect the effectiveness of the two-step filtering in improving the efficiency of ERICommitter and its influence on ERICommitter’s performance. Following the settings of RQ6, we disassemble the two-step filtering module of ERICommitter, namely ERICommitter\*, for comparison.

**RQ8: How does the number of retrieved examples influence the performance of ERICommitter?** Based on the

experimental setting in RQ6, we further alter the number of provided examples  $N=\{1, 3, 5, 10\}$  to evaluate the influence of example numbers. Notably, due to the input token limit of GPT-3.5-turbo (4,096 tokens at most), we could only feed one example for it. However, to more comprehensively understand the impact of the number of examples on LLMs' generation performance, we introduce the version of GPT-3.5-16k as an additional experimental subject, which can accommodate 16,385 tokens for input. Although the GPT-3.5-16k may not perform as well as the GPT-3.5-turbo overall, the two LLMs are consistent in basic principles. Therefore, when evaluating the influence of the example number on LLM with ICL, we can replace GPT-3.5-turbo with GPT-3.5-16k to observe the performance variation for GPT-3.5 series LLMs.

TABLE VII  
PERFORMANCE COMPARISON OF ERICOMMITTER AGAINST VARIOUS UNDERLYING LLMs

Model	Java			Python		
	Met.	BLEU	Rou.	Met.	BLEU	Rou.
Lexical-based Retrieval						
GPT-3.5	20.40	28.27	18.50	17.02	28.96	18.33
ERICommitter(GPT-3.5)	<b>23.54</b>	<b>30.40</b>	<b>21.98</b>	<b>19.60</b>	<b>29.53</b>	<b>19.91</b>
Gemini	5.96	11.24	6.66	3.65	7.59	5.22
ERICommitter(Gemini)	<b>16.00</b>	<b>23.76</b>	<b>17.18</b>	<b>12.06</b>	<b>22.72</b>	<b>14.49</b>
Semantic-based Retrieval						
GPT-3.5	20.40	28.27	18.50	17.02	28.96	18.33
ERICommitter(GPT-3.5)	<b>21.81</b>	<b>29.89</b>	<b>20.13</b>	<b>19.03</b>	<b>29.05</b>	<b>19.55</b>
Gemini	5.96	11.24	6.66	3.65	7.59	5.22
ERICommitter(Gemini)	<b>15.54</b>	<b>24.19</b>	<b>17.44</b>	<b>11.20</b>	<b>20.86</b>	<b>13.17</b>

### C. Experimental Results of ERICommitter

**RQ6: How does ERICommitter perform against individual LLMs?** Table VII showcases the experimental results between ERICommitter and its underlying LLMs with lexical/semantic-based retrieval methods, respectively. Apparently, ERICommitter improves LLMs' CMG performance across different PLs consistently. Utilizing a lexical-based retrieval approach, GPT-3.5 achieves average enhancements of 15.26%, 4.75%, and 13.72% in the METEOR, BLEU, and ROUGE-L, respectively. Besides, Gemini exhibits substantial improvements of 199.43%, 155.36%, and 167.77% in terms of each evaluation metric on order. As for the setting of semantic-based retrieval, GPT-3.5 demonstrates average gains of 9.36%, 3.02%, and 7.73% in terms of METEOR, BLEU, and ROUGE-L, respectively. Meanwhile, Gemini shows significant enhancements of 183.79%, 145.02%, and 157.08% in terms of each metric in order. In summary, similar examples significantly improve the effectiveness of in-context learning for commit message generation with LLMs.

To better illustrate ERICommitter's performance compared to individual LLMs that only include fixed examples in the prompt, we further evaluated the performance of ERICommitter following the manual evaluation procedure described in Section V-B. We present radar charts in Fig. 12, showcasing the enhancement of commit message quality across five key aspects after employing ERICommitter on GPT-3.5 and Gemini

with lexical/semantic-based retrieval methods. For example,  $\text{ERICommitter}_L(\text{GPT-3.5})$  denotes the results of ERICommitter with GPT-3.5 leveraging the lexical retrieval method. To be specific, both GPT-3.5 and Gemini demonstrate significant improvements in terms of all aspects, especially Accuracy and Applicability, when generating commit messages. Judging from all the above, ERICommitter substantially enhances LLMs' CMG performance by retrieving high-quality examples from real-world projects with in-context learning. That's to say, the examples obtained through retrieval are similar to target code *diff*, whereas the fixed examples may be irrelevant. Therefore, the proposed framework, namely ERICommitter, can generate commit messages with LLMs that exhibit superior performance.

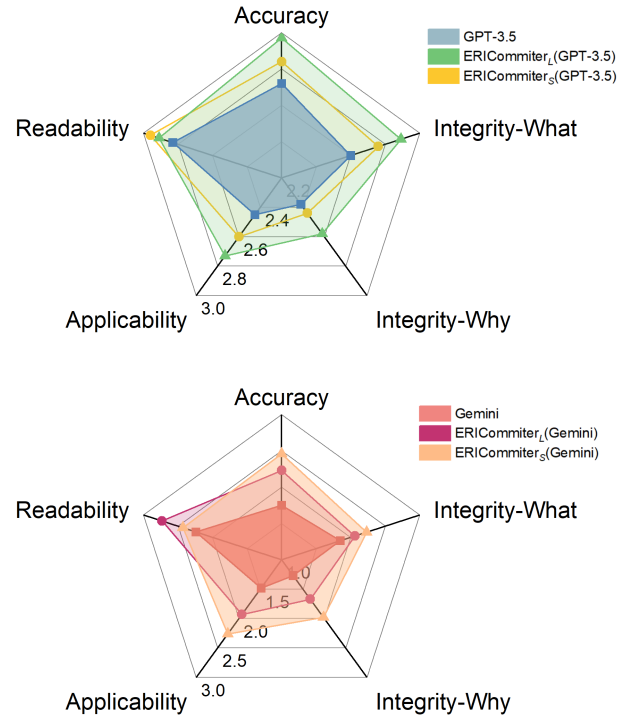


Fig. 12. An Overview of ERICommitter's Performance Among Different Aspects of Manual Assessment

**Answer to RQ6:** Our proposed framework, namely ERICommitter, consistently and substantially improves its underlying LLMs' performance when generating commit messages for code *diff*s in diverse PLs.


**RQ7: How efficient is ERICommitter?** Table VIII illustrates the efficacy and efficiency comparison between ERICommitter and ERICommitter\*, where the latter does not contain the two-step filtering module. As can be seen, ERICommitter performs neck-to-neck with ERICommitter\* overall, but ERICommitter reduces the retrieval time to 6.22% for lexical-based retrieval and 7.15% for semantic-based retrieval, which substantially improves the operating efficiency of the framework. Besides, the lexical-based retrieval method costs

TABLE VIII  
PERFORMANCE COMPARISON BETWEEN ERICOMMITTER WITH/WITHOUT  
TWO-STEP FILTERING MODULE

Model	Java			Python			AVG Time(s)
	Met.	BLEU	Rou.	Met.	BLEU	Rou.	
Lexical-based Retrieval							
ERICommitter(GPT-3.5)	23.54	30.40	21.98	<b>19.60</b>	29.53	<b>19.91</b>	<b>35.99</b>
ERICommitter(GPT-3.5)*	<b>24.35</b>	<b>32.11</b>	<b>23.73</b>	18.80	<b>30.21</b>	19.73	594.22
ERICommitter(Gemini)	16.00	23.76	17.18	12.06	22.72	14.49	<b>35.99</b>
ERICommitter(Gemini)*	<b>16.52</b>	<b>25.31</b>	<b>17.34</b>	<b>11.72</b>	<b>24.23</b>	<b>16.50</b>	594.22
Semantic-based Retrieval							
ERICommitter(GPT-3.5)	21.81	<b>30.89</b>	20.13	19.03	29.05	19.55	<b>0.81</b>
ERICommitter(GPT-3.5)*	<b>22.24</b>	30.86	<b>22.30</b>	<b>18.22</b>	<b>30.06</b>	<b>20.06</b>	11.31
ERICommitter(Gemini)	<b>15.54</b>	24.19	17.44	11.20	<b>20.86</b>	13.17	<b>0.81</b>
ERICommitter(Gemini)*	14.90	<b>23.85</b>	<b>18.48</b>	<b>10.01</b>	20.79	<b>11.31</b>	11.31

\* ERICommitter\* refers to ERICommitter without the two-step filtering module.

much more time than the semantic-based method. The reason is that BM25 relies on an inverted index for document retrieval. An inverted index is a data structure that maps each word to a list of documents containing that word. For each code *diff* retrieval, BM25 needs to traverse the inverted lists of every word in the entire dataset and calculate a score based on factors such as the frequency of words in the document, document length, and the inverse document frequency (IDF) of the word in the entire dataset, resulting in slow retrieval speeds. In contrast, semantic retrieval only needs to perform the operation of converting the code *diff* into a fixed-dimensional vector once and then use cosine similarity to measure the similarity, which has very low computational complexity. In certain scenarios where ERICommitter shows a slight underperformance compared to ERICommitter\*, we suspect that the two-step filtering process might exclude some samples that are highly similar to specific target code *diff*s. This could result in the retrieved examples misleading the LLMs into generating lower-quality commit messages. Nonetheless, the significantly reduced time overhead makes ERICommitter a practical framework in reality.


 **Answer to RQ7:** ERICommitter substantially reduces the retrieval time cost compared with ERICommitter\* and carries almost the same performance, showing the high practicality of ERICommitter in real software development and maintenance.

**RQ8: How does the number of retrieved examples influence the performance of ERICommitter?** Fig. 13 demonstrates the results of ERICommitter provided with various retrieved examples. As can be seen, with an increasing number of provided examples, the performance of the LLMs generally shows an upward trend in most situations. As more examples are available, they offer richer contextual information, aiding the model in accurately understanding and generating commit messages relevant to the target code *diff*.

However, this trend does not apply in all cases. In some LLMs, when the number increases excessively, their performance decreases. A potential explanation is more examples may include more noises, as samples that are similar to the target code *diff* in the retrieval database are limited. Therefore,

selecting an appropriate number of high-quality examples is crucial for optimizing model performance.

The relatively stable performance of METEOR in Fig. 13 could be attributed to the nature of the METEOR metric itself and the characteristics of LLMs. METEOR is designed to balance precision and recall while incorporating synonymy and stemming, which puts more emphasis on semantics rather than literal matching than other metrics. LLMs derive their understanding of semantics primarily from the code *diff* itself. While additional examples can help LLMs learn the structure and format of commit messages, they do not necessarily alter the underlying semantics.

 **Answer to RQ8:** The performance of ERICommitter, mainly in BLEU and ROUGE-L metrics, improves with more retrieved examples, but excessive examples can introduce noise and reduce effectiveness. Selecting an optimal number of high-quality examples is crucial for maximizing performance.

## VIII. THREATS TO VALIDITY

In this section, we carefully considered the following threats to the validity of our research.

### A. Internal Validity

One threat to the internal validity lies in the potential of data leakage that the cleaned dataset may have a certain overlap with the training samples of LLMs examined. As the MCMD dataset was crawled from Github, it is inevitable that LLMs might have seen some project code during their pre-training stage. However, samples in the MCMD dataset are all code *diff*s owing to the characteristic of the CMG task, which carries a totally different input format against the pre-training samples of LLMs [22], [69], [70]. In addition, we thoroughly reviewed all testing results and found that exactly no LLM-generated commit message is identical to the ground truth. Meanwhile, our proposed framework, namely ERICommitter, also achieved substantial improvement over its underlying LLMs. Considering all the factors above, we believe the threat of data leakage is limited.

### B. External Validity

One potential threat to external validity is that we did not exhaustively evaluate all existing LLMs, but selected four LLMs for evaluation. However, among the LLMs we selected, there are both open-source LLMs (Gemini, LLaMA) and closed-source LLMs (GPT-3.5), and we also chose LLMs with different parameter sizes (LLaMA-7B and LLaMA-13B). Besides, LLMs of diverse families are also included (GPT-3.5, LLaMA, and Gemini). Therefore, our evaluation of LLMs is relatively representative and can mitigate this threat. In the future, we can involve more LLMs for experiments.

Another threat to external validity concerns certain manual errors or inaccuracies during the data cleaning process for the MCMD dataset. To address this, we excluded non-target PL

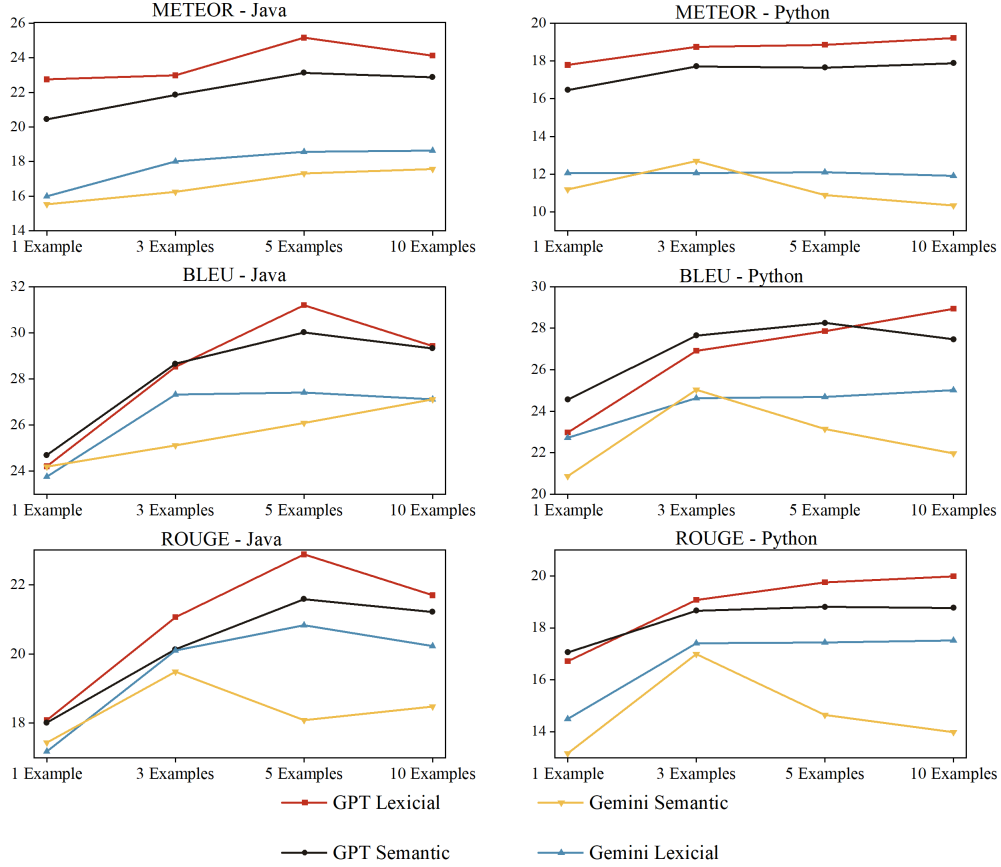


Fig. 13. The Influence of the Number of Retrieved Examples on ERICCommitter

samples by regular expressions and harnessed a deep learning-based model to conduct further filtering semantically to improve the density of high-quality samples, thereby making the following manual selection more concentrated on high-quality samples and improving the selection efficiency. In addition, we introduced a pairwise evaluation mechanism to manually select high-quality commit messages, where kappa consistency scores among the evaluators are all over the required criterion, demonstrating a rigorous and convincing procedure. Hence, we believe this threat can be minimal.

### C. Construct Validity

The lack of comprehensive evaluation metrics also poses a threat to validity. In this study, we select METEOR, BLEU, and ROUGE-L as the assessment metrics for CMG approaches and LLMs because they are widely used in the CMG domain. Additionally, aligning with the OSS practice, we also propose four manual evaluation aspects, covering Accuracy, Integrity, Applicability, and Readability, to systematically measure different kinds of approaches' commit message generation abilities. Therefore, the evaluation metrics adopted in this paper are representative and comprehensive.

## IX. CONCLUSION

This paper serves as the first systematic study investigating the capability of LLMs in generating commit messages based

on given code *diffs*. Specifically, motivated by our pilot analysis, we first screen and construct a high-quality test set through a rigorous three-step cleaning based on the most widely used Commit Message Generation (CMG) dataset, namely MCMD. Afterward, we assess the CMG performance of recent LLMs against state-of-the-art CMG approaches of diverse categories and carry out an in-depth manual analysis of LLM-generated commit messages from aspects of Accuracy, Integrity, Applicability, and Readability aligning with the Open-Source Software (OSS) practice. Results demonstrate the superiority of LLMs in the CMG task where GPT-3.5 performs the best. Finally, we further propose an Efficient Retrieval-based In-context learning framework, namely ERICCommitter, to improve LLMs' CMG performance. Comprehensive experiments prove the remarkable efficacy and efficiency of ERICCommitter, showing its practicality in OSS development and maintenance.

**Implications for Practitioners:** The research highlighted the significant benefits of using LLMs for Commit Message Generation (CMG) in the OSS practice. By re-evaluating state-of-the-art CMG approaches and LLMs, we presented a more objective and authentic result for practitioners to instruct their applications of CMG approaches in daily development and maintenance. Besides, our proposed ERICCommitter substantially enhances LLMs' CMG performance in a training-free manner and boasts a low time overhead, making it an effective and efficient LLM-based CMG approach that can be

extensively deployed in modern software engineering practice.

**Implications for Researchers:** This work carried out the first systematic empirical study on LLMs' performance in the CMG field. A series of automatic and manual assessments demonstrate their prospects and limitations, shedding light on the research of future alternative approaches. Besides, we constructed a high-quality test set cleaned from the most widely used CMG dataset, namely MCMD, facilitating researchers to evaluate future CMG approaches from a more practical perspective. Finally, our proposed framework, namely ERI-Committer, promoted the advancement of CMG approaches, especially with LLMs, proving the potential of in-context learning in generating commit messages.

## REFERENCES

- [1] M. Linares-Vásquez, L. F. Cortés-Coy, J. Aponte, and D. Poshyvanyk, "Changscribe: 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.
- [2] E. Shi, Y. Wang, W. Tao, L. Du, H. Zhang, S. Han, D. Zhang, and H. Sun, "Race: Retrieval-augmented commit message generation," 2022.
- [3] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang, "Neural-machine-translation-based commit message generation: how far are we?" in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 373–384.
- [4] S. Liu, C. Gao, S. Chen, L. Y. Nie, and Y. Liu, "Atom: Commit message generation based on abstract syntax tree and hybrid ranking," *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1800–1817, 2020.
- [5] W. Tao, Y. Wang, E. Shi, L. Du, S. Han, H. Zhang, D. Zhang, and W. Zhang, "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.
- [6] Q. Liu, Z. Liu, H. Zhu, H. Fan, B. Du, and Y. Qian, "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.
- [7] C. V. Lopes, V. I. Klotzman, I. Ma, and I. Ahmed, "Commit messages in the age of large language models," *arXiv preprint arXiv:2401.17622*, 2024.
- [8] K. Cai, J. Zhou, L. Kong, D. Liang, and X. Li, "Automated comment generation based on the large language model," in *International Conference on Computer Science and Education*. Springer, 2023, pp. 283–294.
- [9] Z. Yang, J. W. Keung, X. Yu, Y. Xiao, Z. Jin, and J. Zhang, "On the significance of category prediction for code-comment synchronization," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 2, pp. 1–41, 2023.
- [10] R. Pan, A. R. Ibrahimzada, R. Krishna, D. Sankar, L. P. Wassi, M. Merler, B. Sobolev, R. Pavuluri, S. Sinha, and R. Jabbarvand, "Understanding the effectiveness of large language models in code translation," *arXiv preprint arXiv:2308.03109*, 2023.
- [11] C. S. Xia, Y. Wei, and L. 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.
- [12] L. Zhang, J. Zhao, C. Wang, and P. Liang, "Using large language models for commit message generation: A preliminary study," *arXiv preprint arXiv:2401.05926*, 2024.
- [13] Y. Zhang, Z. Qiu, K.-J. Stol, W. Zhu, J. Zhu, Y. Tian, and H. Liu, "Automatic commit message generation: A critical review and directions for future work," *IEEE Transactions on Software Engineering*, 2024.
- [14] L. Wang, X. Tang, Y. He, C. Ren, S. Shi, C. Yan, and Z. Li, "Delving into commit-issue correlation to enhance commit message generation models," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 710–722.
- [15] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein, "The missing links: bugs and bug-fix commits," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, 2010, pp. 97–106.
- [16] P. R. Mazrae, M. Izadi, and A. Heydarnoori, "Automated recovery of issue-commit links leveraging both textual and non-textual data," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2021, pp. 263–273.
- [17] H. Ruan, B. Chen, X. Peng, and W. Zhao, "Deeplink: Recovering issue-commit links based on deep learning," *Journal of Systems and Software*, vol. 158, p. 110406, 2019.
- [18] J. Li and I. Ahmed, "Commit message matters: Investigating impact and evolution of commit message quality.(2023)," 2023.
- [19] Y. Tian, Y. Zhang, K.-J. Stol, L. Jiang, and H. Liu, "What makes a good commit message?" in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 2389–2401.
- [20] T.-H. Jung, "Commitbert: Commit message generation using pre-trained programming language model," *arXiv preprint arXiv:2105.14242*, 2021.
- [21] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," *arXiv preprint arXiv:2109.00859*, 2021.
- [22] S. Liu, J. Keung, Z. Yang, F. Liu, Q. Zhou, and Y. Liao, "Delving into parameter-efficient fine-tuning in code change learning: An empirical study," *arXiv preprint arXiv:2402.06247*, 2024.
- [23] H. Wang, X. Xia, D. Lo, Q. He, X. Wang, and J. Grundy, "Context-aware retrieval-based deep commit message generation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 4, pp. 1–30, 2021.
- [24] Q. Dong, L. Li, D. Dai, C. Zheng, Z. Wu, B. Chang, X. Sun, J. Xu, and Z. Sui, "A survey on in-context learning," *arXiv preprint arXiv:2301.00234*, 2022.
- [25] S. Bhattamishra, A. Patel, P. Blunsom, and V. Kanade, "Understanding in-context learning in transformers and llms by learning to learn discrete functions," *arXiv preprint arXiv:2310.03016*, 2023.
- [26] C. Wang, X. Liu, Y. Yue, X. Tang, T. Zhang, C. Jiayang, Y. Yao, W. Gao, X. Hu, Z. Qi *et al.*, "Survey on factuality in large language models: Knowledge, retrieval and domain-specificity," *arXiv preprint arXiv:2310.07521*, 2023.
- [27] O. Ovadia, M. Brief, M. Mishaeli, and O. Elisha, "Fine-tuning or retrieval? comparing knowledge injection in llms," *arXiv preprint arXiv:2312.05934*, 2023.
- [28] S. Liu, C. Gao, S. Chen, L. Y. Nie, and Y. Liu, "Atom: Commit message generation based on abstract syntax tree and hybrid ranking," *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1800–1817, 2022.
- [29] L. F. Cortés-Coy, M. Linares-Vásquez, J. Aponte, and D. Poshyvanyk, "On automatically generating commit messages via summarization of source code changes," in *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*, 2014, pp. 275–284.
- [30] J. Zhan, J. Mao, Y. Liu, J. Guo, M. Zhang, and S. Ma, "Optimizing dense retrieval model training with hard negatives," in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021, pp. 1503–1512.
- [31] J. Gao, J.-Y. Nie, G. Wu, and G. Cao, "Dependence language model for information retrieval," in *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, 2004, pp. 170–177.
- [32] A. Berger and J. Lafferty, "Information retrieval as statistical translation," in *ACM SIGIR Forum*, vol. 51, no. 2. ACM New York, NY, USA, 2017, pp. 219–226.
- [33] P. Loyola, E. Marrese-Taylor, and Y. Matsuo, "A neural architecture for generating natural language descriptions from source code changes," *arXiv preprint arXiv:1704.04856*, 2017.
- [34] S. Jiang, A. Armaly, and C. 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.
- [35] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.
- [36] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "Unixcoder: Unified cross-modal pre-training for code representation," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022, pp. 7212–7225.
- [37] I. Ma and C. V. Lopes, "Improving the quality of commit messages in students' projects," in *2023 IEEE/ACM 5th International Workshop on Software Engineering Education for the Next Generation (SEENG)*, 2023, pp. 1–4.
- [38] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, 2019, pp. 4171–4186.

- [39] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," *arXiv preprint arXiv:2003.10555*, 2020.
- [40] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 1536–1547. [Online]. Available: <https://aclanthology.org/2020.findings-emnlp.139>
- [41] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, L. Shujie, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu *et al.*, "Graphcodebert: Pre-training code representations with data flow," in *International Conference on Learning Representations*.
- [42] J. Li, Y. Zhao, Y. Li, G. Li, and Z. Jin, "Towards enhancing in-context learning for code generation," *arXiv preprint arXiv:2303.17780*, 2023.
- [43] J. Li, Y. Li, G. Li, Z. Jin, Y. Hao, and X. Hu, "Skocoder: A sketch-based approach for automatic code generation," *arXiv preprint arXiv:2302.06144*, 2023.
- [44] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.
- [45] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners."
- [46] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
- [47] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.
- [48] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.
- [49] GPT-Code-Clippy, <https://github.com/CodedotAI/gpt-code-clippy>, (Accessed on 04/12/2023).
- [50] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "Codegen: An open large language model for code with multi-turn program synthesis," *arXiv preprint arXiv:2203.13474*, 2022.
- [51] W. Ahmad, S. Chakraborty, B. Ray, and K. Chang, "Unified pre-training for program understanding and generation," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021.
- [52] S. Banerjee and A. 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.
- [53] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "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.
- [54] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.
- [55] Y. Chang, X. Wang, J. Wang, Y. Wu, K. Zhu, H. Chen, L. Yang, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *arXiv preprint arXiv:2307.03109*, 2023.
- [56] A. Bryman, *Social research methods*. Oxford university press, 2016.
- [57] H. Chmura Kraemer, V. S. Periyakoil, and A. Noda, "Kappa coefficients in medical research," *Statistics in medicine*, vol. 21, no. 14, pp. 2109–2129, 2002.
- [58] S. Zhang, D. Zheng, X. Hu, and M. Yang, "Bidirectional long short-term memory networks for relation classification," in *Proceedings of the 29th Pacific Asia conference on language, information and computation*, 2015, pp. 73–78.
- [59] G. Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth *et al.*, "Gemini: a family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2023.
- [60] S. Kaur and S. Singh, "Improving the quality of open source software," *Agile Software Development: Trends, Challenges and Applications*, pp. 309–323, 2023.
- [61] N. Yilmaz and A. Kolukisa Tarhan, "Quality evaluation models or frameworks for open source software: A systematic literature review," *Journal of Software: Evolution and Process*, vol. 34, no. 6, p. e2458, 2022.
- [62] D. Bertram, "Likert scales," *Retrieved November*, vol. 2, no. 10, pp. 1–10, 2007.
- [63] R. Liu, M. Hannum, and C. T. Simons, "Using immersive technologies to explore the effects of congruent and incongruent contextual cues on context recall, product evaluation time, and preference and liking during consumer hedonic testing," *Food Research International*, vol. 117, pp. 19–29, 2019.
- [64] S. N. Akter, Z. Yu, A. Muhamed, T. Ou, A. Bäuerle, Á. A. Cabrera, K. Dholakia, C. Xiong, and G. Neubig, "An in-depth look at gemini's language abilities," *arXiv preprint arXiv:2312.11444*, 2023.
- [65] G. Lu, X. Ju, X. Chen, W. Pei, and Z. Cai, "Grace: Empowering llm-based software vulnerability detection with graph structure and in-context learning," *Journal of Systems and Software*, p. 112031, 2024.
- [66] J. Yang, S. Ma, and F. Wei, "Auto-icl: In-context learning without human supervision," *arXiv preprint arXiv:2311.09263*, 2023.
- [67] S. Robertson, H. Zaragoza *et al.*, "The probabilistic relevance framework: Bm25 and beyond," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009.
- [68] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: a statistical framework," *International journal of machine learning and cybernetics*, vol. 1, pp. 43–52, 2010.
- [69] B. Lin, S. Wang, Z. Liu, Y. Liu, X. Xia, and X. Mao, "Cct5: A code-change-oriented pre-trained model," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 1509–1521.
- [70] C. Wang, Y. Yang, C. Gao, Y. Peng, H. Zhang, and M. R. Lyu, "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.