RAG-Enhanced Commit Message Generation

LINGHAO ZHANG, School of Computer Science, Wuhan University, China
HONGYI ZHANG, School of Computer Science, Wuhan University, China
CHONG WANG, School of Computer Science, Wuhan University, China
PENG LIANG, School of Computer Science, Wuhan University, China

Commit message is one of the most important textual information in software development and maintenance. However, it is time-consuming to write commit messages manually. Commit Message Generation (CMG) has become a research hotspot. Recently, several pre-trained language models (PLMs) and large language models (LLMs) with code capabilities have been introduced, demonstrating impressive performance on code-related tasks. Meanwhile, prior studies have explored the utilization of retrieval techniques for CMG, but it is still unclear what effects would emerge from combining advanced retrieval techniques with various generation models. This paper proposed **REACT**, a **RE**trieval-**A**ugmented framework for **C**ommi**T** message generation. It integrates advanced retrieval techniques with different PLMs and LLMs, to enhance the performance of these models on the CMG task. Specifically, a hybrid retriever is designed and used to retrieve the most relevant code diff and commit message pair as an exemplar. Then, the retrieved pair is utilized to guide and enhance the CMG task by PLMs and LLMs through fine-tuning and in-context learning. The experimental results show that REACT significantly enhances these models' performance on the CMG task, improving the BLEU score of CodeT5 by up to 55%, boosting Llama 3's BLEU score by 102%, and substantially surpassing all baselines.

CCS Concepts: • **Software and its engineering** → **Software maintenance tools**.

Additional Key Words and Phrases: Commit Message Generation, Pre-trained Language Model, Large Language Model, Retrieval-Augmented Generation

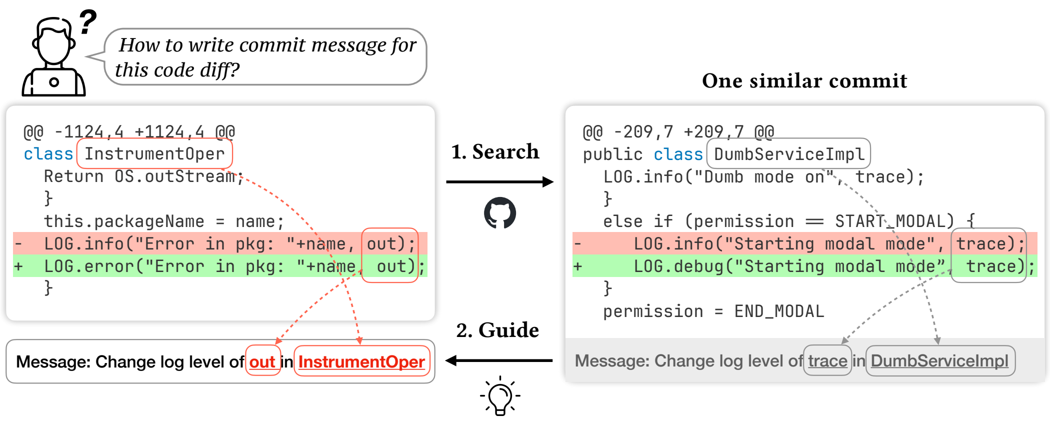
ACM Reference Format:
Linghao Zhang, Hongyi Zhang, Chong Wang, and Peng Liang. 2024. RAG-Enhanced Commit Message Generation. 1, 1 (October 2024), 22 pages. https://doi.org/10.1145/xxxxxxx.xxxxxxx

1 INTRODUCTION

In software development and maintenance, the Git version control system has been widely used to store and share code. Within Git, commit messages describe and document the code changes made in a commit. These messages record alterations to the source code, help developers understand the changes in the code, and promote efficient collaboration. Therefore, the commit message serves as one of the critical pieces of textual information in the software engineering life-cycle. However, writing commit messages is time-consuming and laborious for developers. Many developers find them tedious and are not motivated to write [29]. Additionally, due to the subjectivity of developers, the quality of commit messages in existing code repositories is inconsistent. Some messages can be non-informative (e.g., "initial commit", "last commit today") or even empty [19]. As a recent work

Authors' addresses: Linghao Zhang, School of Computer Science, Wuhan University, Wuhan, China, starryzhang@whu.edu.cn; Hongyi Zhang, School of Computer Science, Wuhan University, Wuhan, China, harryzhang@whu.edu.cn; Chong Wang, School of Computer Science, Wuhan University, Wuhan, China, cwang@whu.edu.cn; Peng Liang, School of Computer Science, Wuhan University, Wuhan, China, liangp@whu.edu.cn.

ACM XXXX-XXXX/2024/10-ART
https://doi.org/10.1145/xxxxxxx.xxxxxxx



```
@@ -1124,4 +1124,4 @@
class InstrumentOper
  Return OS.outStream;
  }
  this.packageName = name;
- LOG.info("Error in pkg: "+name, out);
+ LOG.error("Error in pkg: "+name, out);
  }
```

Message: Change log level of out in InstrumentOper

```
@@ -209,7 +209,7 @@
public class DumbServiceImpl
  LOG.info("Dumb mode on", trace);
  }
  else if (permission == START_MODAL) {
-    LOG.info("Starting modal mode", trace);
+    LOG.debug("Starting modal mode", trace);
  }
  permission = END_MODAL
```

Message: Change log level of trace in DumbServiceImpl

Fig. 1. A motivating scenario of how developers write a commit message by referring to a similar exemplar.

reported [38], on average 44% of commit messages did not reach the desired quality, indicating a lack of essential information and the struggle to convey critical details about what the commit did and why.

In this context, Commit Message Generation (CMG) has become a hot topic in automated software engineering and has garnered attention from this research community. CMG task aims to take the differences between two versions of code as input, typically in the form of a code diff file `.diff` generated by Git, aka code change, and then generates the corresponding commit message. Initially, rule-based approaches were used in CMG [9, 19, 35], where predefined rules or templates were utilized for generation. Some retrieval-based approaches leverage information retrieval (IR) techniques to suggest commit messages from similar code diff [14, 24]. With the rapid development of deep learning, many learning-based methods have recently emerged [10, 16, 21, 26, 46]. These methods treat CMG as a neural machine translation task, in which code diff is the input of neural network model to generate commit messages as output. In addition, there are some hybrid approaches [23, 36, 40] that leverage both IR and deep learning techniques.

More recently, various language models possessing code capabilities have been proposed, including code-specific pre-trained language models such as CodeT5 [43] and general large language models, notably ChatGPT[1], which have attracted the vast attention of the research community. In this paper, we refer to these code-capable language models as Code Language Models (CLMs). Different from traditional models, CLMs can be adapted for downstream code-related tasks by merely fine-tuning or prompting [44], rather than training a model from initial weights. Such approaches not only conserve resources and training time but also can achieve better performance. Researchers have explored applying them to the CMG task [49, 50] and got some promising prospects.

Motivation. Consider a scenario in Figure 1, where if a developer wants to write a good commit message, one of the efficient ways is to search and imitate the exemplar. That is, exemplary commit messages for similar code diff may be valuable and provide additional information to improve the performance of commit message generation. In essence, this scenario embodies the concept of one paradigm of Retrieval-Augmented Generation (RAG) [17]. Prior studies [36, 40] applied retrieval-augmented approaches to the CMG task and conducted preliminary investigations on the effectiveness of retrieved exemplars in improving generation. However, they opted to train a

[1]https://chatgpt.com/

model from scratch instead of leveraging the prior knowledge embedded in pre-trained CLMs and only focused on specific models and trivial retrieval techniques which limited the performance. Moreover, it still remains unclear how to effectively integrate retrieval techniques with different models and how much improvement such integration could bring.

Our approach. In this paper, we propose **REACT** as a **RE**trieval-**A**ugmented framework for **C**ommi**T** message generation. REACT comprises three phases, i.e., *Retrieve*, *Augment*, and *Generate*, to generate commit messages for given code diff. **Retrieve:** We design a hybrid retriever to retrieve the most relevant diff and its corresponding commit message from a comprehensive and high-quality source database. The retrieved diff-message pair will be used to guide the subsequent generation of commit message. **Augment:** The query diff and retrieved diff-message pair need to be combined as the input of the models, that is, input augmentation. The augmenter concatenates them with pre-defined special tokens or fills them into a prompt template before passing the result to the generator for the next phase. **Generate:** The generator receives augmented inputs and generates a commit message corresponding to the query diff under the guidance of the retrieved pair. We use various CLMs as the backbone of the generator instead of training a model from scratch. This can fully leverage the prior knowledge in CLMs and generate messages more effectively. Some generators with PLM backbones require training to learn how to leverage the additional information provided by the retrieved pair to enhance commit message generation. If LLM is used as the generator, the enhancement can be achieved through in-context learning [11] by embedding the retrieved pairs into the input prompt, eliminating the need for training. The final commit message generated through three stages is enhanced by the additional information or similar patterns provided by the exemplar.

Results. We designed and conducted comprehensive experiments to evaluate REACT on a widely-used CMG dataset. Before assessing the enhancement effects of REACT, we first reported the performance of directly using CLMs to generate commit messages. The experimental results showed that without integrating the REACT framework, directly generating commit messages with PLMs yielded remarkable results, surpassing all baselines comprehensively. Specifically, CodeT5's BLEU score was 21% higher than the best baseline method, demonstrating the superiority of PLMs. Although LLMs showed mediocre scores on some metrics, Llama 3's METEOR score still surpassed all baselines without additional training. When integrating CLMs into REACT, the model's performance had broad and significant improvements. CodeT5's BLEU score increased by 55% compared to direct application, and Llama 3's BLEU score increased by 102%. The results established the effectiveness and wide applicability of the REACT framework, setting a new SOTA for the CMG task. Furthermore, we conducted a within-project case study on Electron, a popular open-source project from GitHub, where the exemplars were retrieved solely from the project's historical commits. The results indicated that guided by exemplars, the generation process adheres well to the project's commit message writing conventions, thereby significantly improving metric scores. This finding provides valuable insights into the application of REACT in real-world within-project CMG scenarios.

Contributions. The main contributions of our paper can be summarized as follows:

- **An effective retrieval-augmented framework for CMG.** We propose REACT, a retrieval-augmented framework for enhancing commit message generation. REACT can effectively integrate advanced retrieval techniques with various models for better CMG.
- **Extensive experiments are designed to evaluate the performance of various CLMs on the CMG task.** To the best of our knowledge, this work is the first to comprehensively evaluate the performance of various CLMs on the CMG task, including PLMs (e.g., CodeT5,

UniXCoder, etc.) and LLMs (e.g., ChatGPT, Llama 3, Gemma). Experimental results show that, benefiting from the code understanding ability acquired during pre-training, CLMs achieved impressive performance on CMG with simple fine-tuning or prompting with basic instruction, surpassing the baselines by a large margin. Specifically, CodeT5's BLEU score was 21% higher than the best baseline method RACE.
- **Extensive experiments are designed to evaluate the effectiveness of our REACT framework.** We conduct a comprehensive evaluation of REACT, and the results show that REACT can generally enhance the performance of CLMs compared to directly using them for generation. In terms of BLEU scores, REACT improves CodeT5 by up to 55% and Llama 3 by up to 102%, respectively.

The remainder of this paper is structured as follows: Section 2 covers related work on CMG and CLMs. Section 3 introduces our REACT framework, detailing its three phases. Section 4 describes the experimental setup, including the dataset and model selection, while Section 5 presents results and analysis. Finally, Section 6 and Section 7 discuss the implications and threats to validity respectively, followed by the conclusions in Section 8. The replication package is available online [48]

2 RELATED WORK

2.1 Commit Message Generation

Researchers have proposed several approaches for commit message generation. According to different generating mechanisms, they can be categorized into retrieval-based, learning-based, and hybrid methods.

2.1.1 Retrieval-based methods. NNGen [24] leverages information retrieval techniques to suggest commit messages from similar code diffs. To generate a commit message, NNGen calculates the cosine similarity between the target code diff and each code diff in the collected corpus. Then, the top-k diff-message pairs are selected to compute the BLEU scores, the one with the highest score is regarded as the most similar code diff, and its commit message will be used as the target one. CC2Vec [14] learns a representation of code diff guided by their accompanying commit messages. Similar to the nearest neighbors approach, it computes the distance between code diff vectors and directly outputs the commit message of the closest CC2Vec vector. Retrieval-based methods have significant challenges as the corpus is limited and cannot cover all code diffs.

2.1.2 Learning-based methods. Learning-based methods [10, 16, 21, 26, 46] typically employ deep learning techniques, treating CMG as a Neural Machine Translation (NMT) task. These methods learn how to generate commit messages by training deep neural network models on massive diff-message datasets collected from GitHub projects. CommitGen [16] is an early attempt to use NMT in CMG, it trained a recurrent neural network (RNN) encoder-decoder model using a corpus of diffs and human-written commit messages from the top 1k GitHub projects. CoDiSum [46] uses a multi-layer bidirectional gated recurrent unit (GRU) as its encoder part, which can better learn the representations of code changes. Moreover, the copying mechanism is used in the decoder part to mitigate the out-of-vocabulary (OOV) issue. PtrGNCMsg [21] is another NMT approach based on an improved sequence-to-sequence model with the pointer-generator network which is an adapted version of the attention RNN encoder-decoder model. The most recently proposed learning-based approach is FIRA [10]. It first represents the code diffs with fine-grained graphs, which explicitly describe the code edit operations between the old version and the new version, and code tokens at different granularities. The hybrid architecture of transformer and GNN is adopted as the backbone of the model. FIRA outperforms other learning-based methods and can be considered the current state-of-the-art (SOTA) approach of a single model.

2.1.3 *Hybrid methods.*

ATOM [23] is a hybrid method containing three modules, a generation module encoding the structure of code diff using Abstract Syntax Tree (AST), a retrieval module retrieving the most similar message based on the text-similarity, and a hybrid ranking module selecting the best commit message from the ones generated by generation and retrieval modules. CoRec [40] takes advantage of both IR and NMT, addressing the low-frequency word and exposure bias issue. It trained a context-aware encoder-decoder model. Given a diff for testing, the method retrieves the most similar diff from the training set and then uses it to guide the probability distribution for the final generated vocabulary. RACE [36] combines retrieval and generation techniques in a more integrated way but employs a trivial retriever and opts for training a specific model from scratch. Whereas, our proposed framework REACT incorporates a hybrid advanced retriever, and experimental results demonstrate its effectiveness. Furthermore, REACT is not limited to a specific model, but rather widely adopts various successful models as the generator, enabling more effective generation of commit messages and providing generality insights.

2.2 Code Language Models

Code Language Models (CLMs) refer to language models trained on datasets containing code, equipping them with the capability to generate and understand code. This makes CLMs applicable for downstream code tasks, including but not limited to code completion, program repair, code summarization, and others. In this work, we present a comprehensive evaluation of CLMs on the CMG task, and CLMs act as the generator in our proposed framework. Based on their scale and the methods for transferring to downstream tasks, they can be categorized into the following two types:

2.2.1 *Pre-trained Language Models (PLMs) for Code.*

PLMs are the language models that have been pre-trained on massive training datasets but require fine-tuning when transferred to downstream tasks. Researchers have recently introduced numerous impressive PLMs for code [6, 12, 13, 42, 43]. For instance, CodeT5 [42, 43] is a unified pre-trained encoder-decoder transformer [39] model with the identifier-aware pre-training objective on large-scale program language and natural language datasets, which has impressive code understanding and generation capabilities. There are some empirical studies of PLMs for various tasks in the software engineering field [15, 20, 28, 45], showcasing their remarkable success in code-related tasks.

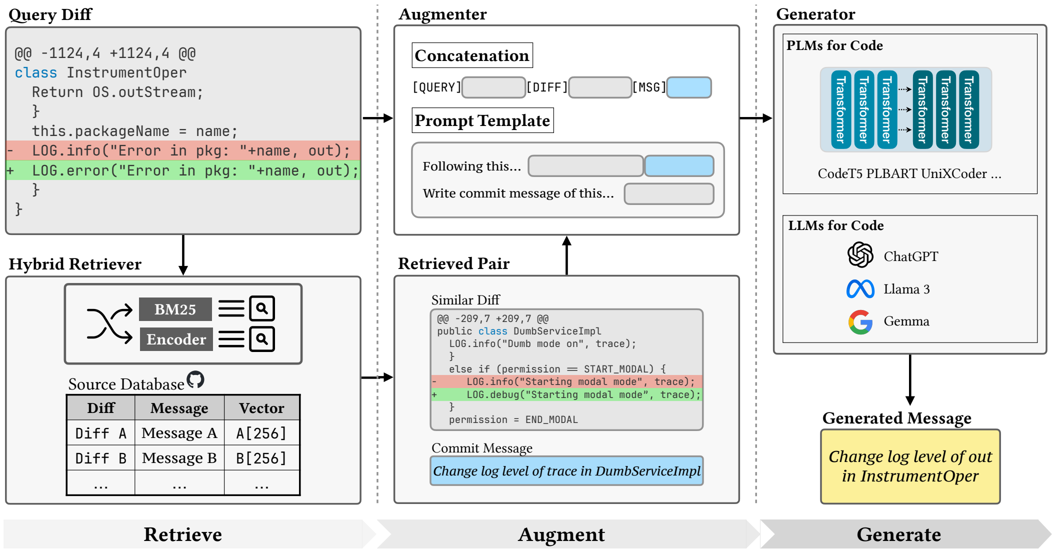
2.2.2 *Large Language Models (LLMs).*

The emergence of ChatGPT marks the beginning of the LLMs era. LLMs possess massive numbers of parameters (exceeding billions), which are typically instruction-tuned [44], enabling them to be directly applied to downstream tasks without training by simply designing the instruct prompt. LLMs generally have code capabilities, and code-related tasks are important parts of LLM benchmarks. Meta recently released its latest series of open-source LLMs, Llama 3[2]. Google has also released its open-source LLM, Gemma[3], which is featured with lightweight. These LLMs have contributed a lot to the open-source community, benefiting countless research activities.

2.3 Retrieval-Augmented Generation

Retrieval-augmented generation [17] involves enhancing the generation of existing models by incorporating knowledge from external databases. A common paradigm includes three steps: information retrieval, data augmentation, and final generation. It has been widely applied to various tasks in NLP, particularly benefiting knowledge-intensive tasks by allowing for the integration

[2]https://llama.meta.com/llama3/
[3]https://ai.google.dev/gemma



```
@@ -1124,4 +1124,4 @@
class InstrumentOper
   Return OS.outStream;
   }
   this.packageName = name;
-  LOG.info("Error in pkg: "+name, out);
+  LOG.error("Error in pkg: "+name, out);
   }
}
```

Diff	Message	Vector
Diff A	Message A	A[256]
Diff B	Message B	B[256]
...	...	...

```
@@ -209,7 +209,7 @@
public class DumbServiceImpl
   LOG.info("Dumb mode on", trace);
   }
   else if (permission == START_MODAL) {
-      LOG.info("Starting modal mode", trace);
+      LOG.debug("Starting modal mode", trace);
   }
   permission = END_MODAL
```

Change log level of trace in DumbServiceImpl

Change log level of out in InstrumentOper

Fig. 2. Overview of our REACT framework. Regarding a code diff (query diff) awaiting the generation of its corresponding commit message, we first retrieve a relevant diff-message pair from the source database using the hybrid retriever. After that, we put query diff and retrieved pair together into the augmenter for concatenation or filling the prompt template. The alterable generator receives formatted inputs and generates a commit message under the guidance of the retrieved pair.

of domain-specific information, and has achieved impressive performance. Recent works have also applied this paradigm of RAG to code-related tasks, including code summarization [22, 31], program repair [41], and code completion [27].

3 METHODOLOGY

This section presents the methodology of our proposed three-phrase framework, i.e., REACT. We first provide a brief overview of REACT. Then, the three phases of REACT are addressed in detail.

3.1 Overview

As shown in Fig. 2, REACT comprises three phases, i.e., *Retrieve*, *Augment*, and *Generate*, to generate the commit message for a specified code diff. In this paper, we refer to the code diff that awaits the generation of its corresponding commit message as the "query diff". Firstly, REACT retrieves the most relevant diff-message pair from the source database using a hybrid retriever. Then, the query diff and the retrieved diff-message pair are sent together to the augmenter for augmentation. Finally, the augmented input is fed into the generator to generate a commit message under the guidance of the retrieved pair.

3.2 Phase I: Retrieve

Phase I intends to retrieve the most relevant diff-message pair for guidance. which is accomplished through the following two components:

3.2.1 Hybrid Retriever. To more accurately assess the similarity between two code diffs and retrieve the most relevant diff-message pair based on that similarity, we designed an advanced hybrid retriever that involves combining two similarity scores with weighted fusion. **BM25** [32] is a relevance scoring algorithm commonly used in information retrieval and search engines. It measures the relevance of a document to a given query by considering factors such as term frequency, document length, and term saturation. BM25 treats query diff as a bag-of-words representation and computes lexical similarity scores between query diff and each of the candidates. We also employ a transformer **encoder**-based similarity scoring method. As an improved version of CodeT5 [43], CodeT5+ [42] is a cutting-edge PLM for code with the architecture of transformer encoder-decoder. Through its pre-training process, it learned rich representations from code data. We extract the encoder component of CodeT5+ and then utilize it to transform the code diff text into a dense 256-dimensional vector embedding, which is generally considered to encapsulate the semantic information of the code diff. By calculating the cosine similarity score between two vectors, we can obtain the similarity score between two code diffs. Following the normalization of the scores to a common scale, we combine the scores obtained from these two methods with equal weights (1:1) in our experiment, and we refer to it as the *hybrid score*. After calculating the hybrid scores between the query diff and all diffs in the source database, the diff-message pair with the highest score is retrieved as the most relevant pair. The hybrid retriever is expected to be more robust and effective compared to single retrievers that rely on only one similarity scoring method.

Data leakage issue: It should be pointed out that due to the intersection between the test set and the source database in the actual experiment, the retrieved diff-message pair may be exactly the same as that of the test set, resulting in data leakage issues. We append a simple mechanism to avoid this issue: when the retrieved diff is detected to be the same as the query diff, the pair with the second-highest hybrid score is selected to replace it.

3.2.2 Source Database. To ensure that each query diff can retrieve a relevant diff-message pair as an exemplar and that the retrieved exemplar is of high quality, containing sufficient information to guide generation, a comprehensive, high-quality source database is crucial. To avoid reinventing the wheel, we reuse the data from CommitBench [34], a recently proposed dataset, to construct the source database used in our work. CommitBench collects commits from 72,000 publicly available GitHub repositories, which is the largest collection scope among existing datasets, ensuring the comprehensiveness of the dataset. Regarding the quality of this dataset, the repositories collected by CommitBench are non-fork repositories, which have been used by at least one other project and have a reasonable number of stars. Additionally, they performed an overall filtering process, designing ten filtering rules to exclude worthless messages, including bot messages, too-short or too-long messages, etc. The resulting source database contains over 1.6 million diff-message pairs. During the pre-processing stage, we persisted the 256-dimensional vector embedding of code diffs, obtained through the encoder in the hybrid retriever, as a new column in the source database to avoid additional computational overhead.

3.3 Phase II: Augment

This phase aims at combining the query diff and the retrieved pair into specific formats to augment the input context. For ease of reference, we denote `<query diff>`, `<retrieved diff>`, `<retrieved msg>` as the query diff for generation, the retrieved code diff, and the corresponding retrieved commit message, respectively. Considering the differences between PLMs and LLMs, we design the following two input augmentation methods for both:

3.3.1 For PLMs. PLMs require a specific input format to serve as the generator in REACT. To explicitly distinguish between different input components, we define three special tokens, i.e., `[QUERY]`, `[DIFF]`, and `[MSG]`. The augmenter concatenates these special tokens with query diff and retrieved pair into the following form:

`[QUERY]<query diff>[DIFF]<retrieved diff>[MSG]<retrieved msg>`

These three special tokens will be added to tokenizers of PLMs and will be fully trained during the fine-tuning stage.

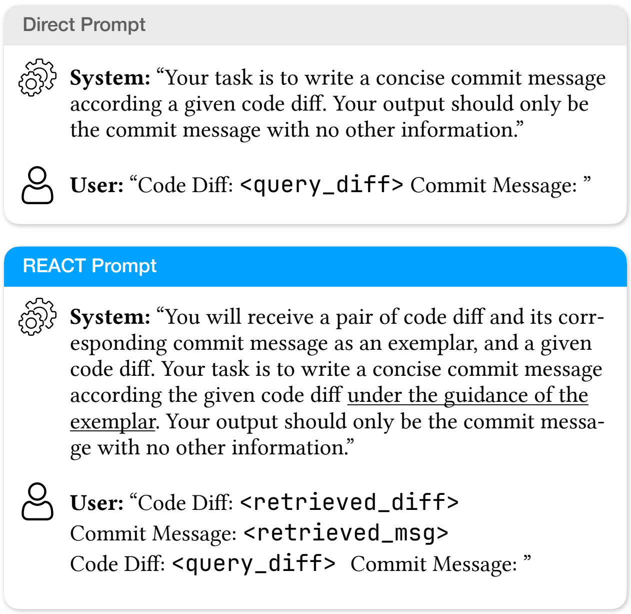
3.3.2 For LLMs. We manually construct a prompt template for LLMs, as Figure 3 shows, to guide commit message generation by referring to retrieved exemplar. The augmenter fills in the corresponding content into the prompt template and then inputs it into LLMs for generation.

3.4 Phase III: Generate

The generator receives the augmented input from Phase II and generates the final commit message under the guidance of the retrieved pair inserted in the input. In this paper, we adopt various models, including PLMs and LLMs, as the generators and conduct a comprehensive study. The specific model selection will be detailed in Section 4.3. In this subsection, we discuss the two types of generators used in REACT and what adjustments they need to make to perform generating.

3.4.1 PLMs. PLMs for code have impressive code understanding capabilities and are well suited for the CMG task. As pre-trained models, PLMs require fine-tuning to fit our generating needs. Fine-tuning is essentially a model training process where PLMs are expected to learn how to generate better commit messages based on the additional information provided by the exemplar.

We denote the augmented input to the PLMs generator as X: `[QUERY]<query diff>[DIFF]<retrieved diff>[MSG]<retrieved msg>`, and the expected output (ground truth) as $\hat{Y}$, that is, commit messages written by the developer. The objective of fine-tuning is to minimize the cross entropy [37] loss $\mathcal{L}(X, \hat{Y})$ of all instances of the training set defined as:



Direct Prompt

System: "Your task is to write a concise commit message according a given code diff. Your output should only be the commit message with no other information."

User: "Code Diff: `<query_diff>` Commit Message: "

REACT Prompt

System: "You will receive a pair of code diff and its corr-esponding commit message as an exemplar, and a given code diff. Your task is to write a concise commit message according the given code diff under the guidance of the exemplar. Your output should only be the commit messa-ge with no other information."

User: "Code Diff: `<retrieved_diff>`
Commit Message: `<retrieved_msg>`
Code Diff: `<query_diff>` Commit Message: "

Fig. 3. Direct prompt template and REACT prompt template for LLMs.

$$\mathcal{L}(X, \hat{Y}) = -\sum_{t=1}^{T} \log P_{\theta}(\hat{Y}_t|X)$$

where $P_{\theta}(\hat{Y}_t|X)$ represents the probability assigned by the PLMs to the correct token $\hat{Y}_t$ at time step t. After adequate training, the overall parameters of PLMs are updated, and they can be used as the generator in REACT.

3.4.2 LLMs. In contrast to PLMs, LLMs can serve as the generator to generate commit messages without training. It benefits from the powerful generalization ability of LLMs, which can complete the specified tasks with only prompting [44]. Meanwhile, through the well-designed instructions as shown in Figure 3, LLMs can perform generating under the guidance of exemplars with in-context learning [11]. By providing a relevant exemplar in the input, that is, one-shot prompting [8], LLMs can perform the CMG task with reference to the provided exemplar. The more relevant the provided exemplar is to the query diff, the more valuable it is as a reference, and LLMs are expected to generate better commit messages.

4 EXPERIMENTAL SETUP

In this section, we first propose four research questions that reflect the basis of our experimental path. Then, we introduce the dataset used in the experiments, the selected models, the baselines for comparison, the metrics used for evaluating results, as well as the implementation details for reference and reproduction.

4.1 Research Questions

The overall goal of this research is to validate the effectiveness of REACT. To achieve this goal, we conducted experiments aimed at investigating the following five research questions (RQs):

RQ1: What is the efficacy of *directly* applying CLMs to the CMG task? How do CLMs perform on the CMG task when directly fine-tuned or prompted with basic instructions, and how do they compare to existing baselines? This RQ is designed to explore the applicability of chosen CLMs on the CMG task.

RQ2: Does REACT enhance the performance of PLMs on the CMG task? When PLMs are integrated into the REACT framework as the generator, to what extent does REACT enhance the performance of PLMs on the CMG task? This RQ is designed to investigate the effectiveness of REACT when integrating with PLMs.

RQ3: Does REACT enhance the performance of LLMs on the CMG task? To what extent does REACT enhance the performance of LLMs on CMG when LLMs are integrated into the framework as the generator and augmented with a distinct in-context learning paradigm? This RQ is designed to investigate the effectiveness of REACT when integrated with LLMs, providing generality insights.

RQ4: Can our hybrid retriever find relevant diff-message pairs to guide generation? Compared to using single retrieval techniques, can the hybrid retriever better retrieve relevant pairs to guide generation? This RQ aims to investigate the ablation study of the hybrid retriever.

RQ5: Can REACT perform well in the within-project scenario? Considering a within-project scenario where commit messages typically follow a consistent style, can REACT adapt its generated commit messages to align with the conventions of the specific project, enhancing both the relevance and usability of the output? This RQ is designed to investigate REACT's ability to generate project-specific commit messages by leveraging historical records from the same project.

Table 1. Statistics of Original and Augmented Datasets.

Dataset	Origin. (RQ1)	Augmen. (RQ2–4)
Count in Training Set	75,000	75,000
Count in Validation Set	8,000	8,000
Count in Testing Set	7,661	7,661
Avg. tokens of Message	9.1	28.4
Avg. tokens of Diff	338.8	634.5

Table 2. Selected Models.

	Model	Parameters	Architecture
PLMs	CodeT5 [43]	220M	Encoder-Decoder
	CodeT5+ [42]	220M	
	PLBART [6]	140M	
	UniXCoder [13]	126M	
LLMs	ChatGPT [2]	N/A	Decoder-only
	Llama 3 [5]	70B	
	Gemma [4]	7B	

4.2 Dataset

In this work, we employ a well-established dataset [46] that has been widely used in previous CMG works [10, 14, 16, 24, 40, 46]. The dataset contains a total of 90,661 pairs of code diff and corresponding commit messages collected from the top 1,000 popular Java repositories in GitHub. Following the previous work [46], the dataset is divided into training, validation, and testing sets in the ratio, as shown in Table 1. In addition to using the original dataset to evaluate the performance of directly applying CLMs to the CMG task in RQ1, we also construct an augmented dataset. The augmented dataset is derived from the original dataset through a retrieval augmentation process described in Section 3.2 and Section 3.3. Thus, for each query diff in the original dataset, a relevant diff-message pair is retrieved and saved. The augmented dataset is used for assessing the effectiveness of REACT in RQ2–RQ4. Table 1 presents the statistics of both the original and augmented datasets, in which “Count” refers to the number of entries in each dataset split. We find that since each entry of the augmented dataset includes two diff-message pairs (query and retrieved pair), the number of tokens nearly doubles. All the models in the experiments are evaluated on the testing set. The training and validation sets are used in the fine-tuning stage of PLMs.

4.3 Model Selection

Table 2 shows the selected CLMs in this paper. Specifically, in the experiment, we adopt the bimodal version for PLMs to acquire the capacities of both natural language and code. For ChatGPT, we use the GPT-4o API version. For Llama 3 and Gemma, we adopt the instruct version so that LLMs can follow the prompt instructions.

In our experiments, in addition to serving these models as the generator in REACT, we also evaluate the performance of these models directly applied to the CMG task for comparison, in which “directly applying” means using the original dataset without augmentation to generate commit messages with no guidance of exemplars.

4.4 Compared Baselines

We choose the following six existing CMG methods as baselines:

Retrieval-based methods: We consider the representative retrieval based method **NNGen** [24], which is the first work to apply IR techniques to the CMG task.

Learning-based methods: We consider three learning-based methods leveraging deep learning models for CMG, i.e., **CommitGen** [16], **PtrGNCMsg** [21], and **FIRA** [10].

Hybrid methods: We consider two hybrid methods that take advantage of both IR and NMT techniques. They are **CoRec** [40] and **RACE** [36].

4.5 Metrics

Following previous works on CMG [10, 14, 46], we employ three widely used metrics, **BLEU**, **Rouge-L**, and **METEOR** to measure the similarity. The evaluation metrics compute similarity scores between the generated text and the ground truth, higher is better.

- **BLEU** [30] is a metric originally used in machine translation evaluation. In this task, it is computed based on the similarity of n-gram between the generated commit message and ground truth (developer-written message). The n-gram precision refers to the ratio of the number of matched n-grams to the number of all the n-grams in the generated text. We use BLEU-4 with 4-grams in the experiments.
- **Rouge-L** [18] stands for Recall-Oriented Understudy for Gisting Evaluation. The measures count the number of overlapping units such as n-grams, word sequences, and word pairs. Rouge-L regards the longest common subsequence (LCS) between the two sentences.
- **METEOR** [7] is calculated based on the harmonic mean of precision and recall, with recall weighted more than precision. It is based on a generalized concept of unigram matching between the generated text and ground truth. It is also widely used in machine translation evaluation.

For simplicity and consistency, all the metrics use the implementation provided by HuggingFace[4].

4.6 Implementation Details

The PLMs used in this study are implemented from the official HuggingFace repository[5][6][7][8], Gemma and Llama 3 are based on the official released model weights and use FP16 precision. ChatGPT used the GPT-4o API as of the experiment date, Aug 1. The fine-tuning of the PLMs was completed on an NVIDIA RTX 4090, utilizing the Adam optimizer to minimize the cross-entropy loss function. The models were trained for 20 epochs using a batch size of 32 and a learning rate of 5e-5, and the same hyperparameters were applied to all four PLMs. Due to the vast size of the source database, which contains millions of entries, we employed the industrial-grade search engine Apache PyLucene [1] to calculate BM25 scores. This ensures that the retrieval average time for one query remains acceptably short.

5 RESULTS AND ANALYSIS

In this section, we present and analysis the experimental results to answer the proposed four RQs.

[4]https://huggingface.co/evaluate-metric
[5]https://huggingface.co/Salesforce/codet5-base
[6]https://huggingface.co/Salesforce/codet5p-220m-bimodal
[7]https://huggingface.co/uclanlp/plbart-base
[8]https://huggingface.co/microsoft/unixcoder-base

5.1 RQ1: Efficacy of Directly Applying CLMs to the CMG Task

At the beginning, we evaluated the CLMs selected for the experiments by directly applying them to the CMG task to assess their feasibility. These CLMs have already been widely applied to various code-related tasks, such as code completion, code summarization, and automated program repair. Additionally, some recent works [25, 47, 49] have explored the performance of some CLMs, e.g. CodeBERT, Llama, and ChatGPT, on the CMG task. However, the evaluations of these studies were conducted on small sampled datasets, and their model selections were neither comprehensive coverage nor cutting-edge. Therefore, it is necessary to comprehensively evaluate and compare the performance of selected CLMs directly applied to the CMG task before investigating the remaining RQs.

To answer RQ1, we selected seven models, covering both PLMs and LLMs and including the latest models (e.g., Llama 3 released in April 2024). Moreover, the evaluations of these seven models are conducted on the whole dataset and the results are compared with existing SOTA baselines.

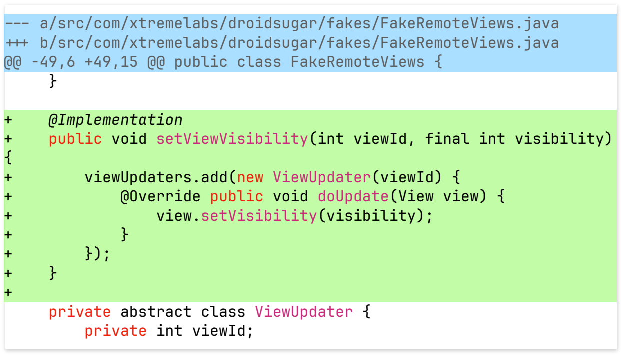
5.1.1 Results. Table 3 lists the scores of CLMs and baselines on the testing set across three metrics (higher is better). The overall results show that CodeT5 performs the best, achieving the highest scores in both BLEU and Rouge-L metrics and significantly surpassing the baselines. Among the four PLMs, three models ranked in the top-3, demonstrating a clear advantage. LLMs obtain decent scores, with METEOR scores matching or exceeding the baselines. However, for the other two metrics, they fall behind FIRA and RACE. Figure 4 shows an example of code diff and its corresponding commit messages generated by selected CLMs.

5.1.2 Analysis. For PLMs, it is impressive that they achieved a significant lead with only simple fine-tuning. Specifically, all three metric scores of CodeT5 are higher than that of RACE, which performs the best in the baselines. Moreover, to achieve such performance, CodeT5 required only 8 hours of training on a single RTX 4090 GPU, whereas RACE took 14 hours in the same environment. The fine-tuning paradigm of PLMs for code not only achieves better performance but also requires relatively fewer resources. In fact, this is the superiority of PLMs. They have

Table 3. Results of directly using CLMs on the CMG task.

	Model	**Metric Scores (%)**		
		BLEU	Rouge-L	METEOR
Baselines	CommitGen [16]	1.36	10.59	9.17
	NNGen [24]	3.09	10.54	10.11
	CoRec [40]	3.03	12.28	10.75
	PtrGNCMsg [21]	0.99	16.06	11.89
	FIRA [10]	2.80	21.56	14.75
	RACE [36]	5.16	24.72	18.19
LLMs	Gemma [4]	1.54	13.50	15.04
	Llama 3 [5]	2.40	17.91	18.65
	GPT-4o [2]	2.57	20.25	19.03
PLMs	UniXCoder [13]	5.24	23.88	17.95
	PLBART [6]	5.17	23.30	19.90
	CodeT5+ [42]	6.00	25.66	**21.75**
	CodeT5 [43]	**6.24**	**25.85**	21.71

Code Diff:



```
--- a/src/com/xtremelabs/droidsugar/fakes/FakeRemoteViews.java
+++ b/src/com/xtremelabs/droidsugar/fakes/FakeRemoteViews.java
@@ -49,6 +49,15 @@ public class FakeRemoteViews {
     }

+    @Implementation
+    public void setViewVisibility(int viewId, final int visibility)
{
+        viewUpdaters.add(new ViewUpdater(viewId) {
+            @Override public void doUpdate(View view) {
+                view.setVisibility(visibility);
+            }
+        });
+    }
+
     private abstract class ViewUpdater {
         private int viewId;
```

Developer : Added setViewVisibility() to RemoteViews

UniXCoder : Add FakeRemoteViews.setViewVisibility
PLBART : Added new setViewVisibility to FakeRemoteViews.
CodeT5 : Add setViewVisibility to FakeRemoteViews
CodeT5+ : Add setViewVisibility method to FakeRemoteViews.

Gemma : Added setViewVisibility method to update views visibility.
ChatGPT : Add implementation for setViewVisibility in FakeRemoteViews
Llama 3 : Add setViewVisibility method to FakeRemoteViews

Fig. 4. An example of CLMs generating commit message according to a code diff.

undergone extensive pre-training on massive datasets, acquiring sufficient prior knowledge. This allows them can be easily transferred to downstream tasks including CMG through fine-tuning. On the other hand, existing CMG methods generally choose to train a model from scratch, which limits their performance. Compared to teaching a child who is just learning to speak (an initial weight model) how to write commit messages, it is easier and more effective to teach a programmer who already knows code and language (a pre-trained model like CodeT5) to do so.

For LLMs, although they do not perform as well as RACE in terms of BLEU and Rouge-L scores, they are equivalent to other baselines. In particular, Llama 3 surpasses all the baselines in the METEOR score. It is worth noting that the LLMs were directly applied to the CMG task without any training or fine-tuning, achieving such results through simple prompting in an out-of-the-box manner. This performance is quite remarkable and also proves that LLMs have sufficient usability for the CMG task.

Overall, the results of RQ1 demonstrate the strong capabilities of CLMs and their suitability for the CMG task. This is owing to the rich prior knowledge embedded in CLMs, which establishes a foundation for their subsequent use as the generator in the REACT framework.

Key Findings of RQ1:

- When directly applying to the CMG task, PLMs significantly outperformed the baselines across the board.
- While LLMs lagged behind the best baseline model in BLEU and Rouge-L metrics, Llama 3 surpassed all baselines in the METEOR score.
- These results demonstrate the superiority of CLMs, attributed to their prior knowledge, which enables them to be effectively transferred to the CMG task through simple fine-tuning or prompting.

5.2 RQ2: Effectiveness of REACT When Integrating with PLMs

For clarity, RQ2 and RQ3 are separated into distinct two research questions because integrating PLMs or LLMs into the REACT framework corresponds to the two different paradigms: fine-tuning or in-context learning. Therefore, we investigate them separately in the two RQs. This RQ primarily explores the effectiveness and extent to which the REACT framework can enhance the CMG performance of PLMs. Before integrating PLMs into the REACT framework as the generator, they must undergo fine-tuning as described in section 3.4.1. The experiment involved evaluating four PLMs across the entire testing set.

5.2.1 Results. The first four rows of Table 4 show the comparison results between directly applying PLMs and integrating PLMs into the REACT framework. The results indicate that REACT enhances the performance of all four PLMs evaluated by a large margin. Specifically, when CodeT5 is integrated as the generator within REACT, it achieves the highest BLEU score of 9.68, representing a 55% improvement compared to using CodeT5 directly. Its Rouge-L and METEOR scores also increase by 5.2% and 8.9%, respectively. More importantly, integrating CodeT5 into the REACT framework results in metric scores that surpass all baselines, establishing a new state-of-the-art (SOTA). For the other three PLMs, REACT also provides varying degrees of improvement. For UniXCoder, the BLEU score shows a maximum percentage increase of up to 76%. Overall, integrating PLMs into REACT can significantly increase BLEU scores, the percentage increases in Rouge-L and METEOR scores are more modest but still notable.

5.2.2 Analysis. The substantial enhancement in metric scores suggests that REACT effectively leverages the prior knowledge embedded in PLMs, allowing them to generate more accurate and relevant commit messages under the guidance of retrieved exemplars. The results also reveal that the integration of retrieval-augmented techniques is particularly beneficial for PLMs with strong

Table 4. Results of REACT integrating with CLMs. "↑X / ↓X": relative changes over directly applying.

Approach		**Metric Scores (%)**		
		BLEU	Rouge-L	METEOR
UniXCoder	*directly*	5.24	23.88	17.95
	REACT	**9.25 ↑76%**	**25.66 ↑7.5%**	**20.16 ↑12%**
PLBART	*directly*	5.17	23.30	19.90
	REACT	**6.95 ↑34%**	**24.15 ↑3.6%**	19.25 ↓3.3%
CodeT5	*directly*	6.24	25.85	21.71
	REACT	**9.68 ↑55%**	**27.20 ↑5.2%**	**23.65 ↑8.9%**
CodeT5+	*directly*	6.00	25.66	21.75
	REACT	**9.59 ↑60%**	**26.30 ↑2.5%**	**22.88 ↑5.2%**
Gemma	*directly*	1.54	13.50	15.04
	REACT	**2.32 ↑51%**	9.69 ↓28%	11.64 ↓23%
GPT-4o	*directly*	2.57	20.25	19.03
	REACT	**3.89 ↑51%**	20.45 ~0%	**19.78 ↑3.9%**
Llama 3	*directly*	2.40	17.91	18.65
	REACT	**4.84 ↑102%**	**19.88 ↑11%**	**20.58 ↑10%**

pre-existing capabilities, like CodeT5 and UniXCoder. These models showed the most substantial gains. Whereas the improvement of PLBART is relatively small.

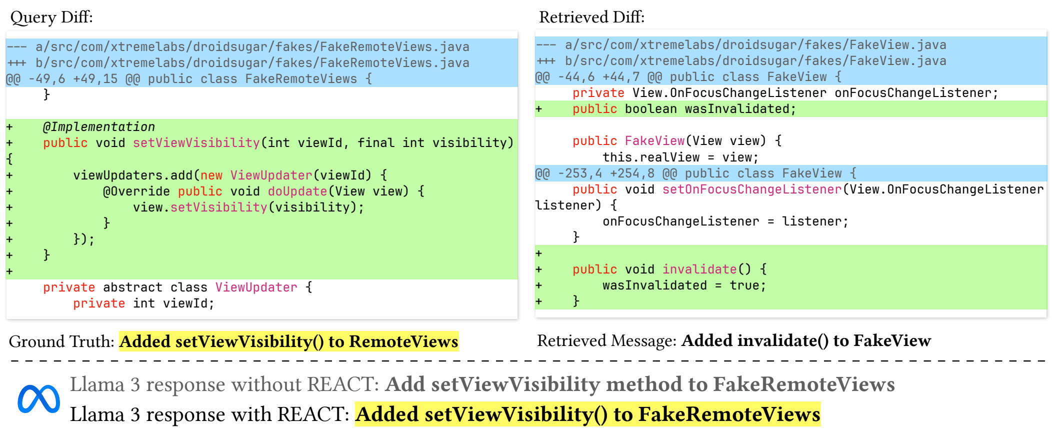
Key Findings of RQ2:

- Integrating PLMs into the REACT framework significantly enhance their CMG performance.
- With REACT, UniXCoder achieves the highest BLEU percentage increase of 76%, while CodeT5 attains the highest BLEU score of 9.68, establishing a new SOTA.

5.3 RQ3: Effectiveness of REACT When Integrating with LLMs

Integrating LLMs into the REACT framework requires no additional training. Compared to direct application, the prompt input to LLMs in REACT includes an example diff-message pair as shown in Figure 3. This essentially transforms a zero-shot scenario into a one-shot scenario by incorporating an exemplar into the context of the LLMs' input, with the expectation that this will enable the LLMs to generate better commit messages.

5.3.1 Results. The last three rows of Table 4 show the comparison results between directly applying LLMs and integrating LLMs into the REACT framework. From the perspective of BLEU scores, REACT significantly and consistently improves LLM performance. Specifically, Llama 3's BLEU score increased from 2.40 to 4.84, representing a 102% improvement. However, for the Rouge-L and METEOR scores, while both ChatGPT and Llama 3 show varying degrees of improvement, Gemma exhibits a noticeable decrease and its scores are comparatively lower than the other two LLMs. Figure 5 shows a case using REACT, the query diff is the same as the one of Figure 4, with the relevant exemplar shown on the right. We can see that both commits originate from different files within the same repository, exhibiting similar commit message writing styles. By referencing the exemplar, Llama 3 with REACT can effectively adopt this writing style, resulting in commit messages that closely align with those written by developers, thereby improving the metric scores.



Query Diff:

```
--- a/src/com/xtremelabs/droidsugar/fakes/FakeRemoteViews.java
+++ b/src/com/xtremelabs/droidsugar/fakes/FakeRemoteViews.java
@@ -49,6 +49,15 @@ public class FakeRemoteViews {
     }

+    @Implementation
+    public void setViewVisibility(int viewId, final int visibility)
{
+        viewUpdaters.add(new ViewUpdater(viewId) {
+            @Override public void doUpdate(View view) {
+                view.setVisibility(visibility);
+            }
+        });
+    }
+
     private abstract class ViewUpdater {
         private int viewId;
```

Ground Truth: **Added setViewVisibility() to RemoteViews**

Retrieved Diff:

```
--- a/src/com/xtremelabs/droidsugar/fakes/FakeView.java
+++ b/src/com/xtremelabs/droidsugar/fakes/FakeView.java
@@ -44,6 +44,7 @@ public class FakeView {
     private View.OnFocusChangeListener onFocusChangeListener;
+    public boolean wasInvalidated;

     public FakeView(View view) {
         this.realView = view;
@@ -253,4 +254,8 @@ public class FakeView {
     public void setOnFocusChangeListener(View.OnFocusChangeListener
listener) {
         onFocusChangeListener = listener;
     }
+
+    public void invalidate() {
+        wasInvalidated = true;
+    }
```

Retrieved Message: **Added invalidate() to FakeView**

Llama 3 response without REACT: **Add setViewVisibility method to FakeRemoteViews**
Llama 3 response with REACT: **Added setViewVisibility() to FakeRemoteViews**

Fig. 5. A case of generating commit message using Llama 3 integrated with REACT.

Table 5. Efficacy of different retrieval methods in REACT (Generator: CodeT5)

Approach	BLEU	Rouge-L	METEOR
No retrieval	6.24	25.85	21.71
Random	6.05	26.05	21.62
Only BM25	9.53	27.11	23.42
Only Encoder	9.53	27.13	23.44
Hybrid	9.68	27.20	23.65

5.3.2 Analysis. REACT can enhance the performance of ChatGPT and Llama 3 in CMG. However, the results for Gemma are somewhat anomalous. Although integrating Gemma into REACT increases its BLEU score by 51%, it leads to a decline in the scores for the other two metrics. This is partly because the focus of evaluation differs among the three metrics. BLEU primarily measures the overlap between generated and reference texts, emphasizing precision. The other two metrics might assess aspects like relevance, fluency, or diversity, which could be adversely affected by the changes that improve BLEU scores. On the other hand, Gemma has the smallest parameter size among the three and has relatively weaker capabilities (according to the EvalPlus Leaderboard [3]). The enhancement brought by REACT relies on the LLM's in-context learning ability, and Gemma's limited capabilities might prevent it from effectively following the instructions and also reliably benefiting from the guidance of the exemplar.

Overall, REACT still effectively enhances the performance of the other two LLMs across all three metrics. It even doubles Llama 3's BLEU score, strongly demonstrating the effectiveness of REACT, proving that including an exemplar in the input for LLMs can guide better generation.

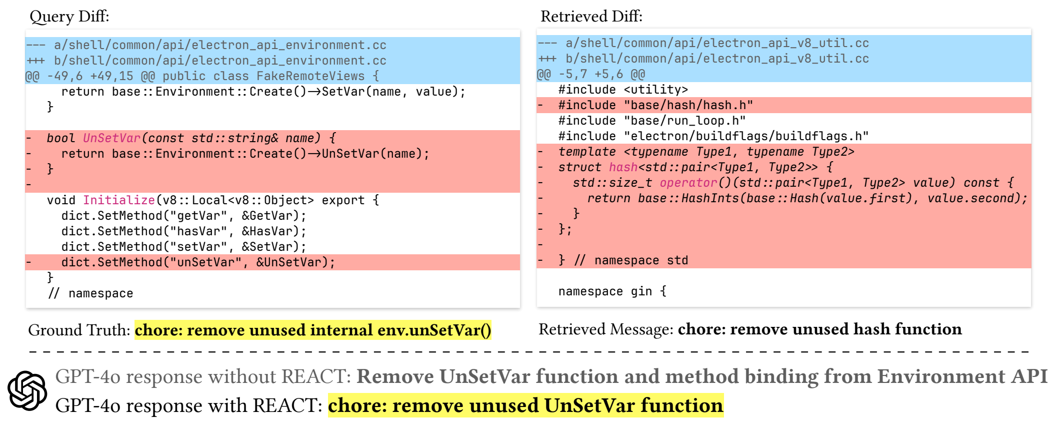
Key Findings of RQ3:

- Integrating LLMs into the REACT framework significantly enhances their CMG performance, especially in terms of BLEU scores.
- With REACT, Llama 3 achieves a 102% improvement in BLEU score solely through prompting, without the need for training.

5.4 RQ4: Ablation Study of the Hybrid Retriever

This RQ intends to validate the effectiveness and necessity of the hybrid retriever within the REACT framework. We aim to determine whether the hybrid retriever can retrieve helpful exemplars. To achieve this, we conducted an ablation study by comparing the effects of different retrieval methods: random retrieval, single retrieval (using either BM25 or encoder), and hybrid retrieval. The generator for these experiments was the best-performing model identified in previous sections, CodeT5. By analyzing the results of these retrieval methods, we can assess whether the hybrid retriever truly works.

5.4.1 Results. Table 5 presents the results of using different retrieval methods within the REACT framework, with CodeT5 as the generator. The table includes the metric scores for BLEU, Rouge-L, and METEOR. "No retrieval" means not providing any exemplar, that is, direct application. "Random retrieval" means randomly selecting a diff-message pair from the source database to serve as an exemplar. The results indicate that using no retrieval achieves the lowest scores. Random retrieval does not aid the generation process and even slightly decreases performance compared to no retrieval. Both single retrieval methods (BM25 and encoder) significantly improve performance,



Query Diff:

```
--- a/shell/common/api/electron_api_environment.cc
+++ b/shell/common/api/electron_api_environment.cc
@@ -49,6 +49,15 @@ public class FakeRemoteViews {
     return base::Environment::Create()->SetVar(name, value);
   }

-  bool UnSetVar(const std::string& name) {
-    return base::Environment::Create()->UnSetVar(name);
-  }
-
   void Initialize(v8::Local<v8::Object> export {
     dict.SetMethod("getVar", &GetVar);
     dict.SetMethod("hasVar", &HasVar);
     dict.SetMethod("setVar", &SetVar);
-    dict.SetMethod("unSetVar", &UnSetVar);
   }
   // namespace
```

Ground Truth: **chore: remove unused internal env.unSetVar()**

Retrieved Diff:

```
--- a/shell/common/api/electron_api_v8_util.cc
+++ b/shell/common/api/electron_api_v8_util.cc
@@ -5,7 +5,6 @@
   #include <utility>
-  #include "base/hash/hash.h"
   #include "base/run_loop.h"
   #include "electron/buildflags/buildflags.h"
-  template <typename Type1, typename Type2>
-  struct hash<std::pair<Type1, Type2>> {
-    std::size_t operator()(std::pair<Type1, Type2> value) const {
-      return base::HashInts(base::Hash(value.first), value.second);
-    }
-  };
-
-  } // namespace std

   namespace gin {
```

Retrieved Message: **chore: remove unused hash function**

GPT-4o response without REACT: **Remove UnSetVar function and method binding from Environment API**

GPT-4o response with REACT: **chore: remove unused UnSetVar function**

Fig. 6. A case of with-in project study using GPT-4o integrated with REACT. Under the guidance of the retrieved exemplar, GPT-4o effectively adhered to the commit message writing conventions of the project, achieving a high degree of similarity with the ground truth.

achieving similar scores. However, the hybrid retrieval approach yields the highest scores across all metrics, indicating its superiority.

5.4.2 Analysis. The substantial improvement in performance when using the hybrid retriever demonstrates its effectiveness in retrieving similar exemplars. Specifically, the BLEU score increases by 60% when comparing the hybrid retriever to random retrieval, the Rouge-L and METEOR scores also show noticeable improvements. The hybrid retriever effectively combines the strengths of both BM25 and encoder-based methods, retrieving more relevant diff-message pairs. This, in turn, provides better guidance for the generator, enabling it to produce more accurate commit messages. The results also reveal that while single retrieval methods are beneficial, the hybrid approach offers additional advantages, further justifying its inclusion in the REACT framework.

Key Findings of RQ4:

- The hybrid retriever can effectively retrieve helpful exemplars to guide generation, with a 60% improvement in BLEU score compared to random retrieval.
- Single retrieval methods (BM25 and encoder) provide substantial improvements, but the hybrid approach offers the highest performance across all metrics.

5.5 RQ5: Within Project Case Study

The dataset used in this study was collected from multiple repositories. In fact, most of the existing widely-used CMG datasets are cross-project that do not differentiate between projects, meaning that the diff-message pairs retrieved in REACT may come from other projects. However, in general development practices, commit messages within a specific repository often follow similar conventions. When a developer writes a commit message for a project, they typically refer to the historical commit log to adopt a consistent style. In some cases, the repository's code of conduct explicitly specifies the commit message conventions[9] that should be followed. The REACT method proposed in this paper, a retrieval-augmented generation approach, intuitively proves to be more practical in

[9]Conventional Commits, https://www.conventionalcommits.org/

this scenario since it can adapt the consistent writing style by following the guidance of retrieved exemplar. In this section, we selected an open-source project for a within-project case study, using experiments to further demonstrate the superiority of REACT in this scenario.

5.5.1 Project Selection. We selected an open-source project from GitHub as the case, **Electron**[10], which meets the following criteria: it is highly popular, with 114k stars and 15.3k forks; it has a rich history of commits, exceeding 28,000 records; and it follows modern project organization and open-source contribution processes, with a comprehensive contribution guide and code of conduct. As a result, the quality of commit message writing is notably high.

5.5.2 Dataset Collecting. We scraped the historical commit records from the main branch as of January 1, 2020, and then filtered out commits with overly long code diffs and commit messages (>1000 tokens). Additionally, we excluded commits authored by bots, such as those by dependabot[11], which typically involve bumping a dependency package to a newer version. In the end, we obtained a dataset containing 3,604 diff-message pairs for the single project Electron.

Table 6. Results of the with-in project experiments for Electron

Approach	BLEU	Rouge-L	METEOR
GPT4o	3.41	23.79	13.03
REACT(GPT4o)	**10.46**	**30.42**	**30.57**

5.5.3 Results & Analysis. We conducted a within-project experiment using GPT-4o. During the retrieval phase of the REACT method, we restricted the retrieval to similar examples only from the current project. As shown in Table 6, the experimental results indicate that in the within-project application scenario, the enhancement to the three metric scores provided by REACT is highly significant. It is able to generate commit messages that closely align with the project's writing style, gaining higher metric scores than directly generating messages without REACT. Figure 6 shows that GPT-4o, based on the retrieved exemplar, learned the commit message writing convention of this project as "[action]: [description]". The commit message generated by REACT successfully emulated this style. This further highlights the practical significance of REACT in real-world within-project scenarios, demonstrating that it not only significantly improves metric scores of CMG but also produces more usable commit messages that can be effortlessly adopted by developers.

Key Findings of RQ5:

- REACT enhances commit message generation in within-project scenarios by adapting to project-specific conventions through examples retrieved from the current project, making the messages more usable and aligned with project standards.

6 IMPLICATIONS

According to the findings of research questions and analysis of experimental results, we discuss the implications for the community:

[10]https://github.com/electron/electron, visiting date: Sep 1, 2024.

[11]https://github.com/dependabot

Enhanced Efficiency and Quality in CMG. One of the most direct implications is the potential for enhanced efficiency and quality in CMG within software development processes. Writing commit messages is a crucial but often tedious task for developers. By automating this process with higher accuracy and relevance, CMG techniques can save developers considerable time and effort. Based on the results of this paper, our proposed approach achieved the highest metric scores compared to all baselines for the selected dataset. This achievement further elevates the technique of automatic CMG to a level of practical value.

Broader Applicability of Retrieval-Augmented Generation (RAG). The experimental results indicate that REACT can broadly and significantly enhance CMG, further validating the effectiveness of the RAG paradigm across various tasks. In fact, RAG has already become one of the best practices for text generation tasks in multiple domains [17], proving to be a simple yet effective concept. Our work further demonstrates the effectiveness of RAG in the CMG task and provides insights for future research.

Advantages and Superiority of CLMs. The results of Section 5.1 indicate that by applying CLMs directly to the CMG task through fine-tuning or prompting, they have already surpassed all existing baselines. This raises a question: Do we really need to train a specialized model from scratch? The demonstrated advantages and superiority of CLMs in the CMG task highlight their potential for other code-related tasks in software engineering. Researchers can investigate the integration of CLMs with various other software engineering tools, extending their utility beyond CMG.

7 THREATS TO VALIDITY

We discuss the potential threats to the validity of this study according to the guidelines proposed by [33], and the impact that these threats may have on our study.

Internal Validity. A primary threat to internal validity is the inherent randomness of LLMs. We keep all parameters, including temperature, at their default settings when using them. However, due to the LLMs' inherent uncertainty, the generated results may vary with each run. To mitigate this threat, future work could involve repeating the experiments multiple times and averaging the results to reduce the impact of randomness. Despite this variability, the significant improvements observed in the experiments with LLMs in our study are sufficient to draw reliable conclusions.

External Validity. A potential threat to validity arises from generalizability. Although our chosen dataset is widely used [10, 14, 16, 24, 40, 46], it only includes Java language and does not investigate the model's generalizability to other languages. We conducted our study on four PLMs and three LLMs with varying parameter sizes and generation capabilities, but we cannot confirm that the results generalize to other models. Future work could involve broader experiments to validate our approach.

Construct Validity. A potential threat to construct validity in this research arises from the evaluation metrics used to assess the quality of the generated commit messages. The three metrics employed primarily measure text similarity between the generated messages and the reference messages. While these metrics are widely used in NLP tasks, they may not fully capture the essential qualities that constitute a high-quality commit message. Future work could consider incorporating additional evaluation methods, such as human evaluations.

8 CONCLUSIONS

This paper proposed REACT, a retrieval-augmented framework designed to enhance commit message generation by effectively integrating advanced retrieval techniques with various language models with code capabilities. Through comprehensive evaluations, we demonstrated that REACT significantly improves the performance of both pre-trained language models and large language models on the commit message generation task. Experimental results indicate that when CodeT5 is integrated into REACT, its metric scores outperform all baselines and achieve the new SOTA. These findings underscore the efficacy of leveraging retrieval-augmented generation in conjunction with the rich prior knowledge embedded in CLMs, providing a robust and efficient solution for automated commit message generation.

For future work, we will consider incorporating additional information beyond retrieving similar exemplars to enhance the input. For instance, for issue-related commits, we can include the issue's description and discussion information to aid in generating commit messages. Meanwhile, the RAG paradigm can be extended to more software engineering tasks.

ACKNOWLEDGMENTS

This work has been partially supported by the National Natural Science Foundation of China (NSFC) with Grant Nos. 62032016 and 62172311.

REFERENCES

[1] 1999. Apache Lucene. https://lucene.apache.org/. Last accessed: May 2024.
[2] 2022. ChatGPT, OpenAI. https://chatgpt.com/. Last accessed: May 2024.
[3] 2023. Evalplus Leaderboard. https://evalplus.github.io/leaderboard.html.
[4] 2024. Gemma, Google. https://ai.google.dev/gemma. Last accessed: May 2024.
[5] 2024. Llama 3, Meta. https://llama.meta.com/llama3/. Last accessed: May 2024.
[6] Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. 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 (NAACL-HLT)*. ACL, 2655–2668.
[7] Satanjeev Banerjee and Alon Lavie. 2005. 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*. ACL, 65–72.
[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems* 33 (2020), 1877–1901.
[9] Raymond P.L. Buse and Westley R. Weimer. 2010. Automatically documenting program changes. In *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 33–42.
[10] Jinhao Dong, Yiling Lou, Qihao Zhu, Zeyu Sun, Zhilin Li, Wenjie Zhang, and Dan Hao. 2022. FIRA: fine-grained graph-based code change representation for automated commit message generation. In *Proceedings of the 44th International Conference on Software Engineering (ICSE)*. ACM, 970–981.
[11] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. 2023. A Survey on In-context Learning. arXiv:2301.00234
[12] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP): Findings*. ACL, 1536–1547.
[13] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified Cross-Modal Pre-training for Code Representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*. ACM, 7212–7225.
[14] Thong Hoang, Hong Jin Kang, David Lo, and Julia Lawall. 2020. CC2Vec: distributed representations of code changes. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE)*. ACM, 518–529.
[15] Kai Huang, Xiangxin Meng, Jian Zhang, Yang Liu, Wenjie Wang, Shuhao Li, and Yuqing Zhang. 2023. An Empirical Study on Fine-Tuning Large Language Models of Code for Automated Program Repair. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1162–1174.

[16] Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Automatically generating commit messages from diffs using neural machine translation. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 135–146.

[17] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, Vol. 33. 9459–9474.

[18] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*. ACL, 74–81.

[19] Mario Linares-Vásquez, Luis Fernando Cortés-Coy, Jairo Aponte, and Denys Poshyvanyk. 2015. ChangeScribe: A Tool for Automatically Generating Commit Messages. In *Proceedings of the 37th IEEE/ACM IEEE International Conference on Software Engineering (ICSE)*. IEEE, 709–712.

[20] Fang Liu, Ge Li, Yunfei Zhao, and Zhi Jin. 2020. Multi-task learning based pre-trained language model for code completion. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 473–485.

[21] Qin Liu, Zihe Liu, Hongming Zhu, Hongfei Fan, Bowen Du, and Yu Qian. 2019. Generating Commit Messages from Diffs using Pointer-Generator Network. In *Proceedings of the 16th IEEE/ACM International Conference on Mining Software Repositories (MSR)*. IEEE, 299–309.

[22] Shangqing Liu, Yu Chen, Xiaofei Xie, Jingkai Siow, and Yang Liu. 2021. Retrieval-Augmented Generation for Code Summarization via Hybrid GNN. arXiv:2006.05405

[23] Shangqing Liu, Cuiyun Gao, Sen Chen, Lun Yiu Nie, and Yang Liu. 2022. ATOM: Commit Message Generation Based on Abstract Syntax Tree and Hybrid Ranking. *IEEE Transactions on Software Engineering* 48, 5 (2022), 1800–1817.

[24] Zhongxin Liu, Xin Xia, Ahmed E. Hassan, David Lo, Zhenchang Xing, and Xinyu Wang. 2018. Neural-machine-translation-based commit message generation: how far are we?. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)*. ACM, 373–384.

[25] Cristina V. Lopes, Vanessa I. Klotzman, Iris Ma, and Iftekar Ahmed. 2024. Commit Messages in the Age of Large Language Models. arXiv:2401.17622

[26] Pablo Loyola, Edison Marrese-Taylor, and Yutaka Matsuo. 2017. A Neural Architecture for Generating Natural Language Descriptions from Source Code Changes. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*. ACL, 287–292.

[27] Shuai Lu, Nan Duan, Hojae Han, Daya Guo, Seung-won Hwang, and Alexey Svyatkovskiy. 2022. ReACC: A Retrieval-Augmented Code Completion Framework. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*. ACL, 6227–6240.

[28] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. arXiv:2102.04664

[29] Walid Maalej and Hans-Jörg Happel. 2010. Can development work describe itself?. In *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 191–200.

[30] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*. ACL, 311–318.

[31] Md Rizwan Parvez, Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Retrieval Augmented Code Generation and Summarization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP): Findings*. ACL, 2719–2734.

[32] Stephen Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends® in Information Retrieval* 3, 4 (2009), 333–389.

[33] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* 14 (2009), 131–164.

[34] Maximilian Schall, Tamara Czinczoll, and Gerard de Melo. 2024. CommitBench. https://doi.org/10.5281/zenodo.10497442

[35] Jinfeng Shen, Xiaobing Sun, Bin Li, Hui Yang, and Jiajun Hu. 2016. On Automatic Summarization of What and Why Information in Source Code Changes. In *Proceedings of the 40th IEEE Annual Computer Software and Applications Conference (COMPSAC)*. IEEE, 103–112.

[36] Ensheng Shi, Yanlin Wang, Wei Tao, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2022. RACE: Retrieval-augmented Commit Message Generation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. ACL, 5520–5530.

[37] John Shore and Rodney Johnson. 1981. Properties of cross-entropy minimization. *IEEE Transactions on Information Theory* 27, 4 (1981), 472–482.

[38] Yingchen Tian, Yuxia Zhang, Klaas-Jan Stol, Lin Jiang, and Hui Liu. 2022. What makes a good commit message?. In *Proceedings of the 44th International Conference on Software Engineering (ICSE).* ACM, 2389–2401.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* 30 (2017).

[40] Haoye Wang, Xin Xia, David Lo, Qiang He, Xinyu Wang, and John Grundy. 2022. Context-aware Retrieval-based Deep Commit Message Generation. *ACM Transactions on Software Engineering and Methodology* 30, 4 (2022), 1–30.

[41] Weishi Wang, Yue Wang, Shafiq Joty, and Steven C.H. Hoi. 2023. RAP-Gen: Retrieval-Augmented Patch Generation with CodeT5 for Automatic Program Repair. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE).* ACM, 146—-158.

[42] Yue Wang, Hung Le, Akhilesh Gotmare, Nghi Bui, Junnan Li, and Steven Hoi. 2023. CodeT5+: Open Code Large Language Models for Code Understanding and Generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP).* ACL, 1069–1088.

[43] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP).* ACL, 8696–8708.

[44] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. arXiv:2109.01652

[45] Chunqiu Steven Xia and Lingming Zhang. 2022. Less training, more repairing please: revisiting automated program repair via zero-shot learning. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE).* ACM, 959–971.

[46] Shengbin Xu, Yuan Yao, Feng Xu, Tianxiao Gu, Hanghang Tong, and Jian Lu. 2019. Commit message generation for source code changes. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI).* IJCAI, 3975–3981.

[47] Pengyu Xue, Linhao Wu, Zhongxing Yu, Zhi Jin, Zhen Yang, Xinyi Li, Zhenyu Yang, and Yue Tan. 2024. Automated Commit Message Generation with Large Language Models: An Empirical Study and Beyond. arXiv:2404.14824

[48] Linghao Zhang, Hongyi Zhang, Chong Wang, and Peng Liang. 2024. Replication Package of the Paper: "RAG-Enhanced Commit Message Generation". https://doi.org/10.5281/zenodo.11505992

[49] Linghao Zhang, Jingshu Zhao, Chong Wang, and Peng Liang. 2024. Using Large Language Models for Commit Message Generation: A Preliminary Study. In *Proceedings of the 31st IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER).* IEEE.

[50] Yuxia Zhang, Zhiqing Qiu, Klaas-Jan Stol, Wenhui Zhu, Jiaxin Zhu, Yingchen Tian, and Hui Liu. 2024. Automatic Commit Message Generation: A Critical Review and Directions for Future Work. *IEEE Transactions on Software Engineering* 50, 4 (2024), 816–835.