Delving into History: Retrieve Less but Augment More on Commit Message Generation

PENG LAN, Central South University, China
JIAKUN LIU*, Harbin Institute of Technology, Faculty of Computing, China
XINYU ZHONG, Central South University, China
NING GUI, Central South University, China
LINGFENG BAO, Zhejiang University, China
DAVID LO, Singapore Management University, Singapore
ZHIFANG LIAO*, Central South University, China

Commit messages are crucial to software development and maintenance, allowing developers to track code changes and collaborate effectively. Automating commit message generation (CMG) reduces developers' manual effort and facilitates program comprehension and software maintenance. Previous approaches typically generate commit messages based solely on code changes, overlooking the valuable context provided by commit history. Moreover, different developers have varying background knowledge and work on different tasks, leading to diverse commit message styles. The commit history provides important context when developers handle different software development tasks across different repositories, yet its potential in CMG remains underexplored. To fill this gap, we propose a novel paradigm named HisRag, which retrieves relevant commit messages from commit history and uses them to enhance existing CMG approaches, improving the quality and relevance of generated commit messages. Extensive experiments showed that HisRag significantly enhances the performance of various CMG approaches. The average improvements on the B-NORM, BLEU, ROUGE-L, METEOR, Log-MNEXT, and BRSA are 65%, 78%, 38%, 42%, 37%, and 12%, and the human evaluation results indicate that HisRag can improve CMG approaches in terms of informativeness, conciseness, expressiveness, and acceptance rate, providing meaningful insights for future research on history retrieval-augmented commit message generation and practical application.

CCS Concepts: • **Software and its engineering → Search-based software engineering**; *Software development techniques*; **Software maintenance tools**.

Additional Key Words and Phrases: Commit Message Generation, Retrieval-Augmented Generation, LLMs, Commit Message History, Code Change Representation Learning

1 Introduction

When developers submit code change in version control systems, they summarize their code change with a simple comment, known as a *commit message*. Commit messages serve as critical documentation for software

*Corresponding authors

Authors' Contact Information: Peng Lan, rambohhh@csu.edu.cn, Central South University, Changsha, China; Jiakun Liu, Harbin Institute of Technology, Faculty of Computing, Harbin, China, jiakunliu@hit.edu.cn; Xinyu Zhong, Central South University, Changsha, China, 8209230211@csu.edu.cn; Ning Gui, Central South University, Changsha, China, ninggui@gmail.com; Lingfeng Bao, Zhejiang University, Hangzhou, China, lingfengbao@zju.edu.cn; David Lo, Singapore Management University, Hekla, Singapore, davidlo@smu.edu.sg; Zhifang Liao, Central South University, Changsha, China, zfliao@csu.edu.cn.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 1557-7392/2026/2-ART
https://doi.org/10.1145/3794857

Just Accepted

maintenance and evolution, because they can help developers understand the intent behind code changes without delving into the source code [50]. However, writing commit messages is a time-consuming and tedious task for developers. Existing works show that there are a significant number of commit messages lacking essential information [59] or even being empty [19] in open-source projects. These low-quality or empty commit messages can negatively impact software development and maintenance [34].

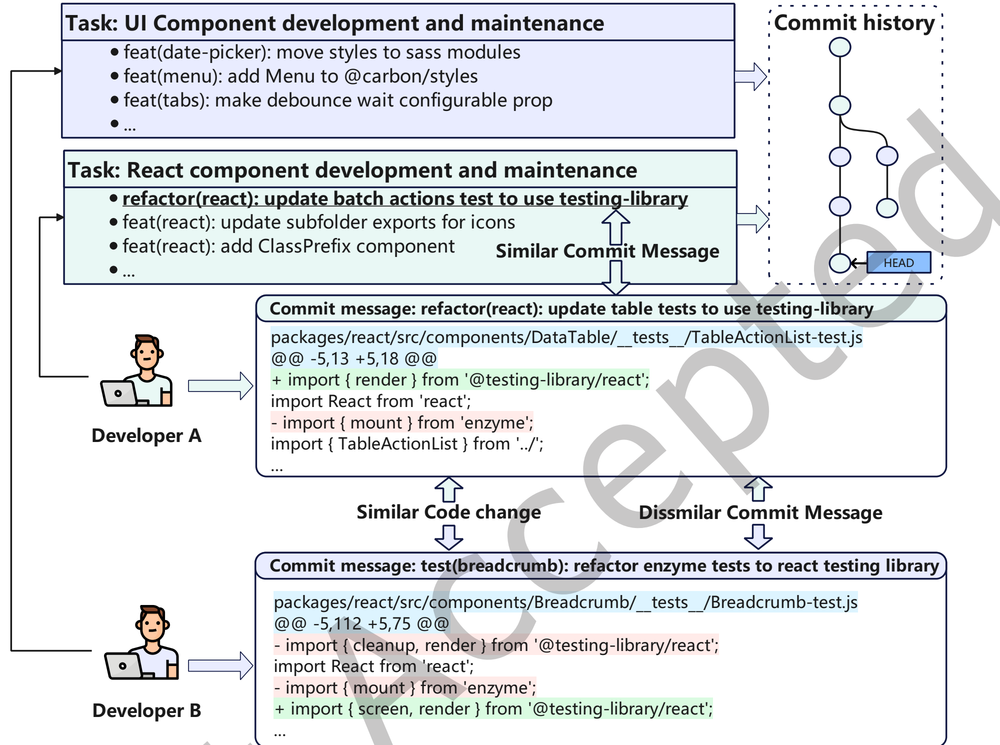


Fig. 1. A motivating scenario in GitHub repository *carbon* [1]. This highlights the variability in commit message styles based on developers' roles and tasks.

Just Accepted

To reduce the manual effort required for writing commit messages, researchers have proposed various approaches for automatically generating appropriate commit messages. Given the fundamental differences between code changes (input) and natural language (output), existing approaches primarily treat CMG as a machine translation task. Mainstream commit message generation approaches can be categorized into five types: template-based approaches [9, 39], retrieval-based approaches [26, 46], representation learning-based approaches [16, 30, 36, 42, 45, 70], hybrid approaches [24, 43, 55, 62, 73] and LLM-based approaches [20, 29, 35, 61, 73]. Despite achieving promising performance, these methods face significant practical adoption barriers [20]. A key limitation lies in the tendency of existing approaches to generate commit messages that are often monotonous, meaningless, or irrelevant [30, 46, 60]. Dong et al. [15] have found that the majority (~90%) of commit messages generated by representation learning-based approaches follow simple patterns (i.e., addition, removal, fix, or avoidance patterns).

[1]https://github.com/carbon-design-system/carbon

Recently, Retrieval-augmented generation (RAG) has become a popular paradigm in natural language processing and is gradually being used in CMG tasks to improve generation performance [20, 73]. Although these CMG approaches achieved significant improvements, they retrieved extensive knowledge from the overwhelming dataset, which could potentially make it difficult for the model to capture the semantics of the original code changes, causing it to get lost in the retrieved information, and decreasing the quality of the generated message [75]. Existing study [75] has also highlighted that the commit message style and background knowledge vary by project. This motivates us to investigate: **Can we retrieve less but augment more to boost the efficacy of CMG models?**

Figure 1 shows a motivating scenario in the GitHub repository *carbon*. Developer A and Developer B work on different tasks (Developer A primarily focuses on React component development and maintenance, and Developer B handles UI Component development and maintenance), which leads to significant differences in their historical commit message styles. At the same time, for a similar code change they made, they still have a different writing style (*refactor(react): update table tests to use testing-library* **vs.** *test(breadcrumb): refactor enzyme tests to react testing library*). Developer B's commit history is not as useful as Developer A's commit history. Existing study [61] also found that for different domain repos, the writing style of commit messages may be different.

To address the limitations above, we first conduct a preliminary study to verify the importance of the commit history from different retrieval sources. We define three retrieval sources: (1) Developer-level: retrieve from the commit history of the developer who wrote the code change, (2) Repository-level: retrieve from the commit history of the repository that the code change belongs to, and (3) Dataset-level: retrieve from all collected commits. We use the retrieved commit messages as generated commit messages to evaluate the retrieval quality. The experiment results show that the BRSA metric score can be improved by 37% on average at the developer-level and 32% on average at the repository-level when compared to the dataset-level retrieval. Furthermore, the number of retrieved instances is reduced by 144 times on average at the developer-level and 49 times on average at the repository-level. These results indicate finer-grained retrieval sources (developer-level and repository-level) are more valuable and effective than coarse-grained retrieval source (dataset-level).

Based on the preliminary study results, we propose a novel **His**tory **R**etrieval-**A**ugmented **G**eneration paradigm called HisRag for the CMG task. HisRag comprises three stages: *history retrieval stage*, *augmentation stage*, and *generation stage* to **enhance the generation performance of various existing state-of-the-art CMG models**. In the *history retrieval stage*, the primary objective is to retrieve Top-k similar code changes from the finer-grained commit history for a given code change, which involves the lexical-based and semantic-based retrieval methods. In the *augmentation stage*, the given code change and the commit messages of retrieved code changes construct the augmented input to augment the generator, which typically are small pre-trained language models (SPLMs) or large language models (LLMs). For SPLMs, the augmented inputs are input embeddings constructed by code change tokens and Top-k commit messages from the commit history. For LLMs, the augmented inputs are constructed by the prompt template, which contains the given code change and Top-k commit messages from the commit history. Finally, in the *generation stage*, the generator uses the augmented input to generate the final commit message. For SPLMs, the generator will be fine-tuned based on the augmented inputs. For LLMs, the generator directly generates the commit message through a prompt without training.

We evaluate HisRag on a multi-language dataset from CommitChronicle [20]. The results show that HisRag can significantly enhance the performance of existing SOTA CMG approaches. The average improvements on the B-NORM, BLEU, ROUGE-L, METEOR, Log-MNEXT, and BRSA are 65%, 78%, 38%, 42%, 37%, and 12%. Compared with the dataset-level retrieval source, HisRag achieved 92× and 75× query efficiency improvements by retrieving from developer- and repository-level histories, respectively. Moreover, HisRag enables existing CMG approaches to generate more personalized commit messages by incorporating developer- and project-specific historical patterns. Further, the results of ablation experiments can demonstrate the effectiveness of each stage of HisRag. We also delve into the impact of retrieval sources, retrieval methods, and retrieval numbers on generation

Just Accepted

performance, providing meaningful insights for future research on history retrieval-augmented commit message generation and practical application. In summary, the main contributions of this paper are as follows:

- We proposed HisRag, a novel history retrieval-augmented generation paradigm for CMG, which consists of a history-retrieval stage, an augmentation stage, and a generation stage. It is compatible with both SPLMs and LLMs, which can enhance the performance of various existing CMG approaches and generate personalized commit messages with historical preference.
- We conducted a systematic exploration of how different levels of retrieval granularity (developer, repository, and dataset) affect CMG, which has not been explored in previous work. Our findings reveal that retrieving from finer-grained commit history (developer-level and repository-level) consistently outperforms retrieval from coarse-grained source (dataset-level), while also significantly reducing retrieval overhead. This investigation provides a strong empirical foundation for practical CMG tools deployments.
- We conducted extensive experiments to verify the effectiveness of HisRag from automatic and human evaluation. The results show that HisRag can enhance 11 CMG approaches across multiple automatic metrics (B-Norm, BLEU, ROUGE-L, METEOR, Log-MNEXT, and BRSA). And the human evaluation results indicate that HisRag can improve existing CMG methods in terms of informativeness, conciseness, and expressiveness. Moreover, the open-source model enhanced by HisRag is comparable to the proprietary CMG tool (GitHub Copilot) on the acceptance rate, underscoring its potential for practical adoption.

2 Background

2.1 Diverse Commit Message Styles Across Developers and Projects

In collaborative software projects, commit history serves as the fundamental record of incremental changes made to the code base over time. It provides critical context for code evolution, enabling developers to track changes, identify the origins of particular functionalities, and understand the progression of software projects from conception to deployment [25]. However, the variety of tasks and contexts developers encounter creates an inherent diversity in commit messages [75]. Depending on the developer's background, the nature of the project, and the specific task being performed, commit messages may differ significantly in form, content, and level of detail [8].

Existing research has explored that software projects have contributors with diverse skill sets, and the style of commit messages is different [10, 34, 61]. This diversity, while reflective of the specific needs of each task, poses a challenge for maintaining a coherent and usable commit history.

2.2 Commit message generation with commit history

To further enhance the performance of CMG approaches, there are two approaches (HACMG [20] and REACT [73]) to use commit history to add to the input of the model recently. Eliseeva et al.[20] proposed a history-aware commit message generation method (HACMG) that uses all the commit message history from a developer as an additional input to enhance the performance of CMG models. Zhang et al. proposed REACT to use the most similar commit from a database to enhance the performance of CMG models. The database can be regarded as the other repositories' commit history. Although the two approaches achieved significant improvements in the CMG task, there are some limitations. For HACMG, it indiscriminately utilizes all commit messages from the developer's history. Due to the long commit history of developers in the actual development process, it is unrealistic to use the commit message history of all developers as input to the model. In addition, a long commit message history can make it difficult for the generated model to capture the semantics of the original code changes in the model input, causing it to get lost in the long commit message history and decreasing the quality of the generated messages. For REACT, the commit history considered comes entirely from other repositories, which can result in generated commit messages lacking historical preferences.

Just Accepted

Considering the difference in developers' domain-specific expertise and the divergence of commit message styles across developers (as shown in Figure 1), we are motivated to study the impact of the retrieval sources and explore methods to efficiently use commit history to enhance the performance of commit message generation.

3 Preliminary Study

The purpose of our preliminary study is to investigate the importance of the historical information from the commits of developers and repositories.

3.1 Motivation

As we introduced in Section 2.2, there are two ways to use commit history in the CMG task. One way is to use all commit message history from the developer (HACMG [20]), and the other is to retrieve the most similar commit from a database (REACT [73]). Naturally, we wonder if retrieving from finer-grained commit history would also improve the efficacy of CMG tasks. Therefore, we want to explore the quality of commit messages retrieved using different retrieval sources through a preliminary study.

3.2 Approach

Based on the idea of NNGen [46] (a classic retrieval-based approach, and even outperforms some learning-based approaches): *Similar code changes are more likely to have similar commit messages*, we use the retrieved commit messages as generated commit messages to evaluate the retrieval quality. For a given code change that requires generating a commit message, it must belong to a certain developer or repository. We define three retrieval sources: (1) Developer-level: retrieve from the commit history of the developer who wrote the code change, (2) Repository-level: retrieve from the commit history of the repository to which the code change belongs, and (3) Dataset-level: retrieve from all collected commits. For retrieval methods, we leverage both lexical-based and semantic-based retrieval methods. Previous work [46] has demonstrated the effectiveness of bag-of-words-based retrieval methods for the CMG task. By measuring the similarity of code change based on lexical, we can retrieve textually similar code changes. We employ the BM25 [51] algorithm, which uses a sparse vector representation for lexical matching, which has demonstrated effectiveness in multiple downstream tasks in software engineering [5, 65]. BM25 converts each code change as bag-of-words representation and computes a lexical similarity score $BM25(\mathcal{C}_i, \mathcal{C}_j)$ between the given code change $\mathcal{C}_i$ and a candidate code change $\mathcal{C}_j$, where $\mathcal{C}_j$ is from the commit history. Since the lexical-based retrieval method relies on lexical matching, the semantic-based retrieval methods are better suited for capturing the deeper semantic information of code changes. We use embedding models to embed code changes and then calculate the cosine similarity between given code changes and code changes in the retrieval commits. Specifically, we compute the semantic similarity score $Cosine(u_i, u_j)$, where u_i and u_j are the embeddings of the given code change $\mathcal{C}_i$ and a candidate code change $\mathcal{C}_j$. We use all-MiniLM-L6-v2 [1] as the embedding model, which is efficient and provides competitive performance [66].

3.3 Dataset

CommitChronicle [20] is a large-scale multilingual CMG dataset. This dataset comprises 10.7 million commits in 20 programming languages and preserves necessary information for utilizing commit history, which is better adapted to real-world software development scenarios. To the best of our knowledge, CommitChronicle is the only dataset that contains the commit history while preserving the chronological order of information when we write this paper. Due to the large total number of commits in the dataset, training and evaluating the performance of models requires an implausible time overhead.

To mitigate this issue, we would like to collect samples from the test set of CommitChronicle, which is not used when training HACMG [20]. Specifically, following prior studies [22, 64] to keep a comparable scale of

Just Accepted

Table 1. The statistical characteristics of the dataset.

Split	Number	Min	Max	Average
train	Tokens in code change	1	3290	271.06
	Tokens in message	1	59	8.65
	Retrived instances from developer-level	0	2385	276.38
	Retrived instances from repository-level	0	3188	809.34
	Retrived instances from dataset-level	86042	86042	86042
validation	Tokens in code change	1	3132	315.3
	Tokens in message	2	51	8.19
	Retrived instances from developer-level	0	2534	423.67
	Retrived instances from repository-level	90	3263	1285.23
	Retrived instances from dataset-level	86043	86043	86043
test	Tokens in code change	5	2970	272.74
	Tokens in message	2	56	8.48
	Retrived instances from developer-level	0	3104	611.05
	Retrived instances from repository-level	718	3423	1756.57
	Retrived instances from dataset-level	86043	86043	86043

datasets and ensure the representativeness of real-world settings, where they consider repositories with more than 1,000 commits as active/popular repositories, we select the repositories with more than 1,000 commits. Finally, we obtained 107,554 commits scattered in 65 repositories with 4,336 developers across 17 programming languages. This sample size is similar to prior studies [31, 43, 58, 70], e.g., CoDiSum [70], where they collected 90,661 commits from the top 1,000 popular Java projects on GitHub. We split these commits according to the commit timeline and finally obtained the training set, validation set, and test set by 80%/10%/10%. The test set contains 10,756 commits, and all of them have repository history. The number of commits with developer history is 10,216, and for a minority of cases when there is no developer history, we will fall back to considering the whole repository history. Besides, this sample size supports an extensive evaluation to explore the effectiveness of LLMs (the test set covers approximately 10,000 instances). For the retrieval source of dataset-level, we use the training set as the source database.

Table 1 summarizes the statistical characteristics of the dataset. For each split, we report the number of tokens (split by space) in the code change and commit message, as well as the number of retrieved instances from three retrieval sources: developer-level, repository-level, and dataset-level. As shown, the average number of tokens in code change ranges from 271 to 315 across splits, while messages are concise with around 8 tokens on average. The number of retrieved commits varies significantly across different retrieval sources, with the developer level requiring the least retrieval, followed by the repository level. Notably, retrieving from developer and repository history requires calculating a smaller and more variable number of commits, whereas retrieving from a database always involves a fixed and relatively large number of commits. For lexical-based retrieval, the number of retrieved instances is reduced by 144 times on average at the developer-level and 49 times on average at the repository-level. This makes retrieval from the developer and repository history more efficient than retrieving from the database. The dataset-level retrieval yields a constant number of instances due to global retrieval over the entire training corpus. Notably, the dataset-level retrieval count for the training split is slightly smaller (86,042) than that of validation and test (86,043). Each training example is retrieved from all other training instances, excluding itself, whereas validation and test examples are retrieved from the entire training set without such constraints.

Just Accepted

Figure 2 shows the distributions of code change token count and commit message token count across all samples in the dataset. The average code change token count is 275.65, and the average commit message token count is 8.58. The token count of code change and message in the dataset is appropriate for evaluating the most CMG approaches [20, 24, 36, 55, 67, 73]. These methods typically adopt the CodeT5 architecture, which commonly uses an input token length of 512 and an output token length of 50.

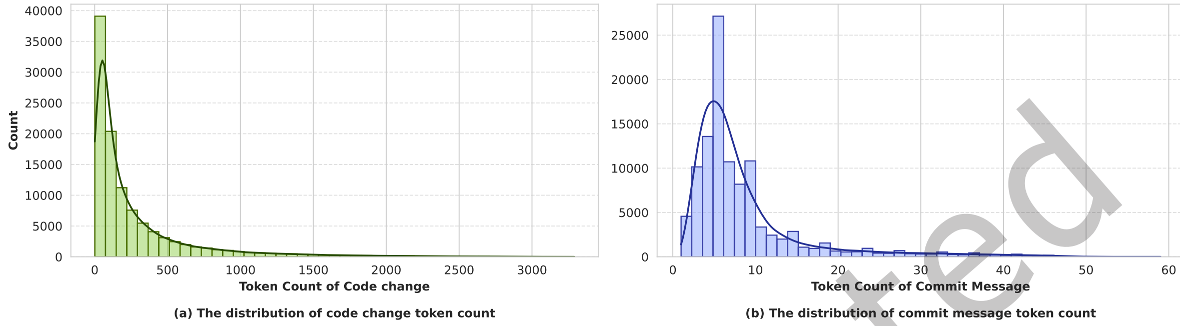


Fig. 2. The token count of the code change and commit message(split by space). The curves show the kernel density estimate (KDE).

Just Accepted

3.4 Evaluation Metrics

We employ six automatic metrics commonly used to evaluate the quality of retrieved commit messages:

• **BLEU** [49] measures the precision of the generated sequence, which is the average of the modified n-gram precision. The modified n-gram precision refers to the ratio of matched n-grams to the n-grams in the generated sequence. In this paper, we used the BLEU-4 metric with a smoothing function.

• **B-Norm** is a version of BLEU that was shown to be the most in line with human judgment on the quality of commit messages by Tao et al [58].

• **Log-MNEXT** is a novel metric on the commit message generation task, and it is based on METEOR-NEXT [13]. In this paper, we used the implementation in Dey's work [14].

• **ROUGE-L** [37] calculates the F-score of precision and recall based on the longest common sub-sequences (LCS) between the generated sequence and the ground truth [38]. A longer LCS indicates a higher similarity between two sentences.

• **METEOR** [7] calculates the harmonic mean of 1-gram precision and 1-gram recall of the generated sequence against the ground truth [7]. It takes into account not only exact word matches but also partial matches, which makes it a more robust evaluation metric compared to BLEU or ROUGE.

• **BRSA** [33] utilizes a semantic model to measure semantic similarities between the generated sequence and the ground truth. Specifically, it utilizes a sentence transformer (MPNet [56]) to embed commit messages into fixed-width semantic vectors and calculate their cosine similarity as SEMSIM scores. Then, the BRSA metric score is calculated by combining ROUGE, BLEU, and SEMSIM (weighed at 0.25, 0.25, and 0.5, respectively) [33].

3.5 Preliminary Study Results

The results of the commit messages retrieved using different retrieval sources and retrieval methods are shown in Table 2. Firstly, using the same retrieval method, retrieving using a dataset-level source performs worse than retrieving using a developer-level source or repository-level source. For lexical-based retrieval, the results show

Table 2. Performance Comparison of Retrieval Strategies (Retrieval methods and Retrieval sources) and CMG approaches.

Approach		B-NORM	BLEU	ROUGE-L	METEOR	Log-MNEXT	BRSA
Lexical-based Retrieval	w/ Developer	13.56	4.24	17.41	17.67	12.81	21.33
	w/ Repository	12.96	3.80	17.19	16.77	12.41	22.18
	w/ Dataset	7.63	1.46	10.58	10.67	7.64	16.16
Semantic-based Retrieval	w/ Developer	12.70	3.76	15.93	16.48	11.61	20.08
	w/ Repository	11.80	3.28	15.32	15.13	10.87	20.51
	w/ Dataset	7.42	1.34	10.00	10.07	7.09	15.42
NNGen [46]		7.50	1.50	10.43	10.18	7.47	15.42
HACMG [20]		9.44	1.71	14.79	8.64	7.96	20.73
REACT [73]		15.70	7.84	21.94	19.57	16.28	27.61
CodeT5 [67]		19.00	8.81	27.41	22.03	18.45	33.73

Just Accepted

that the BRSA metric score can be improved by 37% on average at the developer-level and 32% on average at the repository-level when compared to the dataset-level retrieval. This indicates the potential of finer-grained sources that the history of the developer and repository contains richer information, and a finer-grained retrieval can obtain more useful information. Moreover, using the same retrieval sources, the lexical-based retrieval method outperforms the semantic-based method across six metrics (i.e., B-NORM, BLEU, ROUGE-L, METEOR, Log-MNEXT, and BRSA). One possible reason is that current embedding models have not captured deeper associations of code changes. Using lexical-based methods such as BM25 is still a more effective choice.

We also present the results of four CMG approaches (NNGen [46], HACMG [20], REACT [73], and CodeT5 [67]). NNGen [46] is a classic retrieval-based approach that only uses the retrieved commit message as generated results. Specifically, it retrieves the top-k code changes with the highest lexical similarity from the training set, and its retrieval strategy can be regarded as from the dataset source with a lexical-based retrieval method. The results indicate that the performance of NNGen is comparable to that of dataset-level retrieval, but inferior to fine-grained retrieval.

HACMG [20] and REACT [73] are typical CMG approaches that utilize commit history from different perspectives. HACMG [20] utilizes all the commit message history of the developer. REACT [73] utilizes the retrieved diff and commit message at the dataset-level. We directly apply the original training and evaluation settings from their replication packages. And we set the training set as the retrieval corpus of REACT [73] for fair comparison. CodeT5 [67] is a pre-trained programming language model that shows comparable performance on the CMG task [24, 36, 55], and it only utilizes code change as the input of the model. For their implementations, we apply the same settings according to the public code sources.

Surprisingly, HACMG [20] and REACT [73] perform worse than CodeT5. This illustrates that the use of commit history in HACMG [20] and REACT [73] does not enhance the performance of the model. In addition, retrieving using developer-level sources and retrieving using repository-level sources outperform two CMG approaches: NNGen [46] and HACMG [20]. This motivates us to explore how to effectively incorporate the finer-grained commit history.

> The retrieval sources are important, and different retrieval sources will bring different retrieval results. Finer-grained sources (developer-level and repository-level) are more valuable and effective than coarse-grained retrieval source (dataset-level). Our findings inspire us to leverage finer-grained history information to enhance the efficiency of existing CMG approaches.

4 Proposed Approach

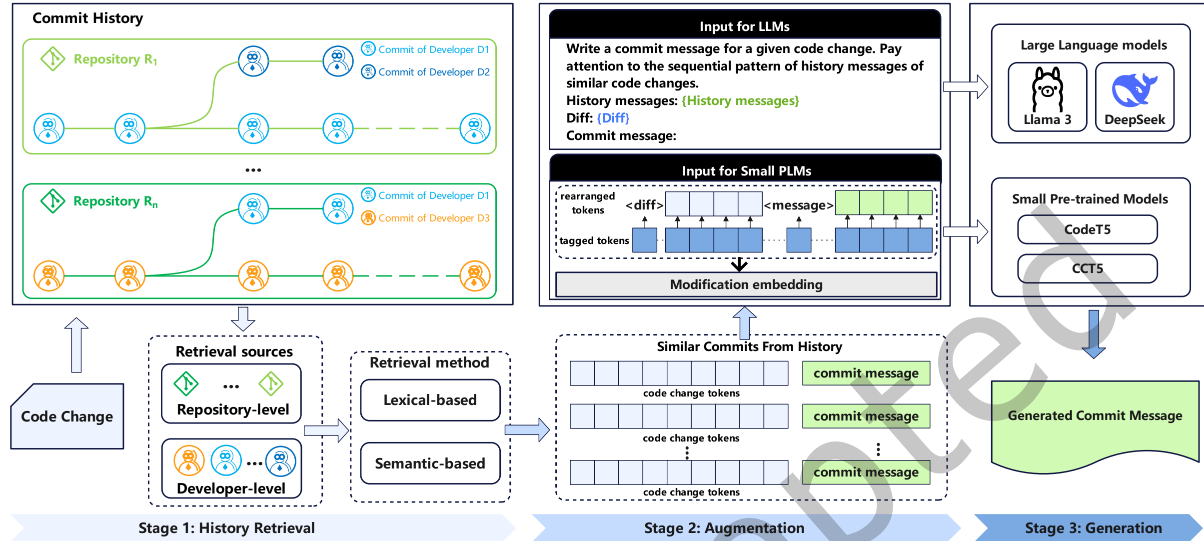


Fig. 3. Overview of HisRag paradigm.

Based on the results of the preliminary study, we introduce a novel paradigm called HisRag (**His**tory **R**etrieval-**A**ugmented **G**eneration), which retrieves less but augments more on commit message generation. Compared with the existing RAG approach [73] on the CMG task, the novelty of HisRag lies in three aspects: **1) Personalized preferences.** Unlike existing coarse-grained retrieval methods [55, 73], HisRag mainly focuses on developer-level and repository-level historical information, which makes generating commit messages more in line with developer preferences. **2) Fast Retrieval.** Different from existing studies [46, 55, 73] that use a large corpus, HisRag retrieve less but augment the performance of CMG more. **3) Broader applicability.** HisRag can support both SPLMs and LLMs, which can adapt to various existing CMG methods. The overview of HisRag is illustrated in Figure 3. It is noteworthy that HisRag is not a simple model that takes code changes as input and generates commit messages as output. Instead, it is a pipeline that integrates the relevant historical information from commit history into existing CMG models (SPLMs or LLMs) to generate commit messages with historical preference. The entire paradigm consists of three stages: history retrieval stage, augmentation stage, and generation stage.

During the first stage, we retrieve the most relevant commits based on lexical or semantic information from the developer's or repository's commit history. Then, during the augmentation stage, we utilize the commit messages of retrieved commits as additional input to provide the retrieved content to the language model. For SPLMs, we employ an edit distance algorithm to construct a specific input format that contains code change tokens and relevant commit message tokens. We employ a sequence-to-sequence task to fine-tune the existing CMG model F. Notably, F can be an existing learning-based or hybrid CMG model that uses code change sequences as input. For LLMs, we combine the given code change and the relevant commit messages into the specific input prompt. Finally, during the generation stage, the augmented input is fed into SPLMs or LLMs, which generate a commit message under the guidance of the retrieved commit messages from the commit history.

Just Accepted

4.1 History Retrieval Stage

The history retrieval stage in HisRag aims to retrieve Top-k similar code changes from commit history and then use the commit messages corresponding to these code changes to guide the generation process. The setting of the retrieval source and retrieval method is consistent with the preliminary study. The retrieval source has two settings: repository-level and developer-level. And the retrieval method has two settings: lexical-based and semantic-based. We use the BM25 [51] algorithm as the lexical-based retrieval method for its effectiveness in multiple software engineering tasks [5, 65]. For the semantic-based retrieval method, due to limitations in computing resources, we choose all-MiniLM-L6-v2 [1] as the embedding model, which is fast and provides competitive performance [66].

4.2 Augmentation Stage

In this stage, the commit messages of retrieved code changes from history will construct the specific input with the given code change. We consider SPLMs and LLMs, respectively, in this paper.

4.2.1 The input augmentation for SPLMs. SPLMs require a specific input format to serve as the generator in HisRag. Following an existing study [24], modification embedding that rearranges the added and deleted code according to the edit distance algorithm can highlight edit operations and represent code changes concisely. Therefore, we reused the edit distance algorithm in [24] to transform code changes into rearranged tokens and tagged tokens. The tag of the unchanged token is set to 0, the deleted token is set to 1, and the added token is set to 2. For the retrieved commit message history, we use the same tokenizer to get rearranged tokens and set the tagged tokens to 3. We add two special tokens (<diff> and <msg>) to the code change and the retrieved commit message history, respectively, and concatenate them. Then, rearranged tokens and tagged tokens will be embedded through the rearranged token embedding layer and tagged token embedding layer, respectively, to obtain rearranged token embedding $E_{rearranged}$ and tagged token embedding E_{tag}. Finally, we get the input embedding E_{input}, which is the sum of $E_{rearranged}$ and E_{tag}. Figure 4 indicates the construct method of input embedding of HisRag for SPLMs. Specifically, we represent each rearranged token r and tagged token t as one-hot vectors, then the encoder of input embedding E_{input} can be obtained:

$$E_{rearranged} = rW_r \tag{1}$$

$$E_{tag} = tW_t \tag{2}$$

$$E_{input} = E_{rearranged} + E_{tag} \tag{3}$$

where $W_r \in \mathbb{R}^{v_1 \times d}$ and $W_t \in \mathbb{R}^{v_2 \times d}$ are two trainable matrices, v_1 denotes to the vocabulary size and v_2 denotes to the number of different tag, and d denotes to the dimension of the input embedding.

Note that modification embedding is an optional component of the HisRag paradigm. It is enabled only for models with compatible architectures that allow tagged token embeddings (e.g., COME [24]). For CMG approaches that do not contain this component, HisRag augment them according to their original model architecture designs. For example, for models that contain retrieval modules (e.g., HACMG [20], RACE [55], and REACT [73]), the history retrieval modules will substitute their retrieval modules; for models that do not contain retrieval modules (e.g., CodeT5 [67] and CCT5 [36]), their original model inputs are replaced with the combination of code change and commit message history through special tokens (i.e., <diff> and <msg>). Section 5.2 explains the detailed settings of each enhanced approach.

4.2.2 The input augmentation for LLMs. For LLMs, we combine the given code change and the retrieved commit message history into specific formats to augment the inputs. We manually construct a prompt template for LLMs, as Figure 3 shows, enabling them to generate commit messages by referencing the writing styles of similar commit history. The design of the prompt template strictly follows the design specifications of existing research

Just Accepted

[6, 20], which indicates that the prompt can include four types of elements: instruction, context, input data, and output indicator. The given code change and retrieved commit message history will be filled in the prompt template and then input into the LLMs for generation, as shown in Table 4.

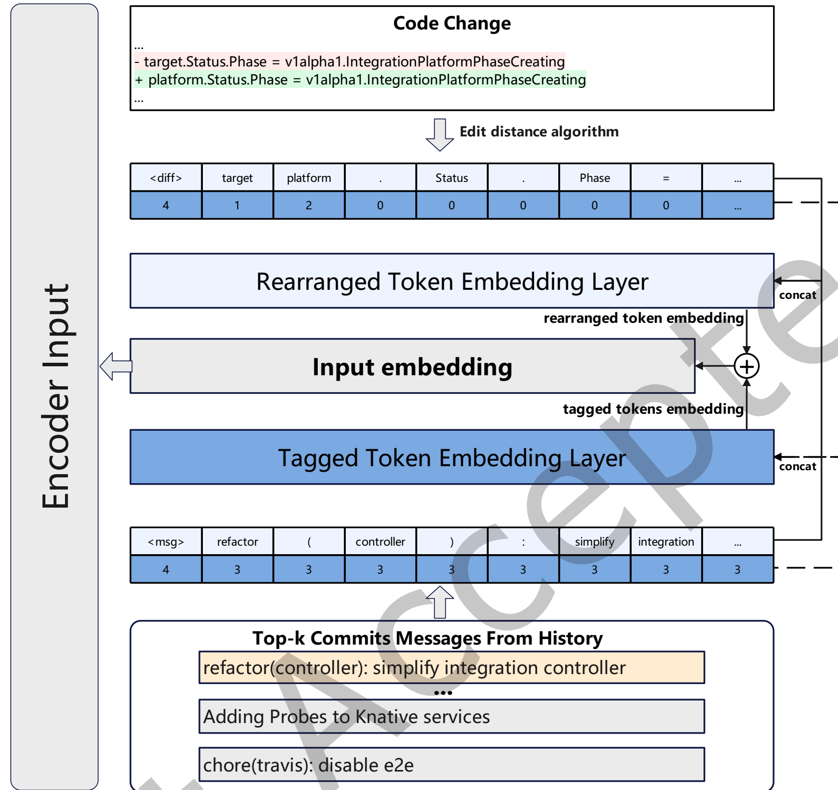


Fig. 4. The input format for SPLMs in HisRag

Just Accepted

4.3 Generation Stage

In the generation stage, we use the augmented input from the augmentation stage and generate the final commit message under the guidance of the retrieved commit messages from the commit history. In our framework, both SPLMs and LLMs can be used as the generators.

4.3.1 SPLMs. SPLMs (i.e., CodeT5 [67]) were pre-trained on a large amount of source code with source code-specific pertaining objectives and have shown impressive code understanding capabilities, and were used in prior studies related to CMG. To incorporate SPLMs in HisRag, we fine-tuned a small pre-trained model to learn how to generate better commit messages with the help of the historical information provided by the retrieval stage. Specifically, the model is fine-tuned to predict the commit message sequences $\hat{Y}$ by the input sequences $C + T$. The log-likelihood function is used as the objective function:

$$\mathcal{L} = -\frac{1}{|\mathbb{X}|} \sum_{i=1}^{|\mathbb{X}|} \sum_{j} \log \mathcal{P}\left(Y_{ij} \mid C_i + T_i; \theta\right) \tag{4}$$

where C_i is concatenated by rearranged code token sequences and similar message tokens, T_i denotes the tagged token sequences of C_i, Y_{ij} denotes the jth token of the ith commit message to be generated in the dataset $\mathbb{X}$, and θ denotes the model parameters.

4.3.2 LLMs. Given the plug-and-play nature, LLMs can serve as the generator without the need for additional training, leveraging their powerful generalization capabilities. The input from the augmentation stage is directly used to generate the final commit message, guided by the retrieved commit message history.

5 Experimental Setup

5.1 Research Questions

Our experiments aim to address the following research questions.
RQ1: To what extent can HisRag enhance the performance of existing CMG approaches?
RQ2: How do different retrieval sources and retrieval methods affect the performance of the HisRag?
RQ3: How do the number of commit messages from history and modification embedding affect the performance of the HisRag?
RQ4: How do human evaluation and manual case analysis reveal the quality and characteristics of the commit messages generated by HisRag?
RQ5: Is HisRag computationally efficient in real-world usage settings?

5.2 Enhanced Approaches

Because our experimental dataset covers various programming languages instead of focusing on a single one, we didn't consider the methods relying on code structure [16, 43]. In addition, we didn't compare HisRag with recent CMG approaches like OMG [35] and OMEGA [29], which utilize multiple additional context information (i.e., PR title, issue title, and commit type). The primary reason is that their datasets are significantly smaller in scale and include types of contextual metadata that are not considered in existing CMG methods like RACE [55], COME [24], and CCT5 [36]. Due to the disparity in dataset scale and contextual information, a direct comparison would require extensive adaptation for all existing CMG methods, which is beyond the scope of this paper. Since our work focuses on modeling commit message history without relying on extra contextual information, we leave a thorough comparison with such context-aware methods to future work. We use HisRag to enhance a total 11 state-of-the-art CMG approaches across 4 types as follows:

- **Retrieval-based approach**
  (1) **NNGen** [46] is the representative retrieval-based method, which is the first work to apply IR techniques to the CMG task. For applying HisRag, we change its retrieval sources to the developer's commit history.
- **Learning-based approach**
  (2) **CodeT5** [67] is a variation of the sequence-to-sequence language model T5 that was pre-trained on a large amount of source code with source code-specific pertaining objectives.
  (3) **CCT5** [36] is a code-change-oriented pre-trained models. It is pre-trained with the code data from CodeChangeNet [36].
- **Hybrid approach**
  (4) **RACE** [55] is a hybrid CMG approach that utilizes similar commit messages and code changes to improve the performance of CMG models.

Just Accepted

(5) **REACT** [73] is a RAG-enhanced framework for CMG.
(6) **HACMG** [20] is the only state-of-the-art method using commit history when we write this paper (March 15, 2025). It uses the commit message history of the developer as additional information to construct the input with the code change.
(7) **COME** [24] is the state-of-the-art representation-based method. It first proposes modification embeddings to represent code changes in a fine-grained way.

- **LLM-based approach**

We mainly consider four LLMs for comparison: **CodeLlama-7B** [52], **Llama-3-8B** [2], **Deepseek V3** [40] and **DeepSeek-R1-Distill-Llama-8B** [23].

For original implementation, we implement it strictly following the open-source code or paper. Due to the different design of existing CMG methods, we use different settings when adapting HisRag to enhance them. There are four main settings that need to be set up: 1) Whether to use the retrieved code change. 2) Whether to use the retrieved message. 3) Retrieved number. 4) Whether to use modification embedding.

Table 3 shows the specific settings of different CMG approaches enhanced by HisRag. The motivation of HisRag is to learn the style of commit messages rather than code semantics from the retrieved code change. However, some approaches (i.e., NNGen [46], RACE [55], and REACT [73]) are originally designed to require both the retrieved code change and message as the input of the model. To preserve the original design of these approaches and make a fair comparison, we use the retrieved code change and message from the history retrieval stage. For approaches that didn't consider the retrieved code change in their original implementations, we keep the same setting of only retrieved commit messages, just like the existing approach (HACMG [20]) did.

For the retrieved number, we set its default value to three for both SPLMs and LLMs. This setting is based on the observation that each retrieved message contains a maximum of approximately 60 tokens, and 3 is a relatively appropriate value based on the input limit of the model. There are some special cases for some CMG approaches. Specifically, NNGen [46] is a representative retrieval-based approach that does not require training, and we set the retrieved number to 5 according to its original implementation. For RACE [55] and REACT [73], due to the limit of model input (The average retrieved code change contains about 270 tokens, making it impractical to include multiple retrieved instances without exceeding the input budget), we set the retrieved number to 1 for these two approaches.

For modification embedding, we apply it only to the COME [24], as its model architecture is explicitly designed to contain this component. COME utilizes edit distance algorithm that distinguishes added, deleted, and unchanged code tokens, allowing it to capture fine-grained information through tagged representations. In contrast, other CMG approaches (i.e. CCT5 [36], RACE [55], and REACT [73]) considered in this paper do not include components that support modification embedding. Therefore, to maintain architectural consistency and ensure a fair comparison, we don't apply modification embedding to these CMG approaches except COME [24].

Just Accepted

5.3 Experimental Settings

We employ BLEU, B-Norm, Log-MNEXT, ROUGE-L, METEOR, and BRSA mentioned in Section 3.4 as automatic evaluation metrics.

Experiments are conducted using different sets of hyperparameters to optimize HisRag's performance on the dataset. We use lexical-based retrieval at the developer level as the default evaluation setting. We use the weight of the CodeT5-base [67] to initialize the CMG model. The original vocabulary size of CodeT5 is 32,100. We adopt the AdamW optimizer [32] with a 5e-5 learning rate. The batch size is 12, and the max epoch is 10. All experiments are performed on a single 24G GPU of NVIDIA RTX 4090. To ensure a consistent and fair comparison, all CMG approaches and their variants enhanced by HisRag were evaluated under the same data splits.

Table 3. The settings of CMG approaches enhanced by HisRag. We use BM25 as the default retrieval method.

Approach	Retrieved code change	Retrieved message	Retrieved number	Modification
NNGen [46]	✓	✓	5	×
CodeT5 [67]	×	✓	3	×
CCT5 [36]	×	✓	3	×
RACE [55]	✓	✓	1	×
REACT [73]	✓	✓	1	×
HACMG [20]	×	✓	3	×
COME [24]	×	✓	3	✓
CodeLlama-7B [52]	×	✓	3	×
LLama3-8B [2]	×	✓	3	×
Deepseek V3 [40]	×	✓	3	×
DeepSeek-R1-Distill-Llama-8B [23]	×	✓	3	×

Table 4. The prompt template of LLM-based approaches and HisRag-enhanced approaches.

Approach	Input design
LLM-based	Write a commit message for a given diff. Only output commit message. Diff: {Diff} Commit message:
HisRag	Write a commit message for a given code change. Pay attention to the sequential pattern of history messages of similar code changes. History messages: {History messages} Diff: {Diff} Commit message:

6 Results

In this section, we present the overall effectiveness of HisRag-enhanced CMG approaches (RQ1) , the performance of the history retrieval stage (RQ2) , the contribution of the augmentation stage (RQ3) , the human evaluation results (RQ4), and the computational efficiency (RQ5).

6.1 RQ1: To what extent can HisRag enhance the performance of existing CMG approaches?

To verify the overall effectiveness of HisRag, considering the length of the commit message may impact the evaluation results, we investigate the effectiveness of HisRag under scenarios of short and long commit messages. Since the lengths of commit messages vary in our dataset, we partition the test set into six quantile-based (approximately equal frequency) groups according to the number of tokens in the ground truth message. We index these groups from Group 1 to 6, where Group 1 contains the shortest messages, and Group 6 contains the longest ones. This stratification allows us to verify the effectiveness of HisRag under scenarios of short and long messages.

The overall effectiveness of HisRag is presented in Table 5. The experimental results show that HisRag enhances the performance of different existing approaches significantly across evaluation metrics (p-value < 0.05). Among them, the retrieval-based method NNGen [46] achieved significant improvements in all evaluation metrics, such as B-Norm increasing from 7.5 to 13.56 (81% ↑), BLEU increasing from 1.5 to 4.24 (183% ↑), and ROUGE-L increasing

Just Accepted

Table 5. Generation performance for HisRag-enhanced commit message generation models. “Ori.” means Original; “Enh. ” is Enhanced; “Imp.” indicates Improvement.

Type	Approach	B-Norm			BLEU			ROUGE-L			METEOR			Log-MNEXT			BRSA		
		Ori.	**Enh.**	**Imp.**	**Ori.**	**Enh.**	**Imp.**	**Ori.**	**Enh.**	**Imp.**	**Ori.**	**Enh.**	**Imp.**	**Ori.**	**Enh.**	**Imp.**	**Ori.**	**Enh.**	**Imp.**
Retrieval-based	NNGen [46]	7.50	13.56	81%	1.50	4.24	183%	10.43	17.41	67%	10.18	17.67	74%	7.47	12.81	71%	15.42	21.33	38%
Learning-based	CodeT5 [67]	19.00	20.52	8%	8.81	11.09	26%	27.41	28.90	5%	22.03	26.31	19%	18.45	21.61	17%	33.73	35.81	6%
	CCT5 [36]	19.29	22.43	16%	8.83	10.47	19%	27.84	30.69	10%	22.51	26.73	19%	18.99	21.94	16%	33.28	35.47	7%
Hybrid	RACE [55]	18.90	20.24	7%	8.62	9.32	8%	27.07	27.86	3%	21.63	24.2	12%	18.85	20.09	7%	32.45	32.97	2%
	REACT [73]	15.70	20.86	33%	7.84	10.34	32%	21.94	29.70	35%	19.57	25.41	30%	16.28	21.66	33%	27.61	35.40	28%
	HACMG [20]	9.44	10.79	14%	1.71	2.28	33%	14.79	15.82	7%	8.64	9.56	11%	7.96	8.94	12%	20.73	21.31	3%
	COME [24]	19.39	22.93	18%	8.73	11.00	26%	28.08	30.82	10%	22.66	26.69	18%	19.29	21.95	14%	33.76	35.83	6%
LLM-based	CodeLlama-7B [52]	7.00	10.21	46%	1.52	1.88	24%	14.36	16.38	14%	15.60	18.12	16%	13.64	15.26	12%	24.16	26.83	11%
	LLama3-8B [2]	12.13	18.82	55%	5.31	9.26	74%	20.77	26.56	28%	16.50	24.40	48%	16.06	20.46	27%	29.32	32.43	11%
	Deepseek V3 [40]	5.73	15.43	169%	1.37	3.65	166%	15.05	25.73	71%	18.63	29.74	60%	16.22	24.32	50%	29.61	35.09	19%
	Deepseek-R1 [23]	5.55	13.03	269%	1.04	2.30	266%	13.08	21.64	171%	15.59	24.03	160%	13.53	19.39	150%	26.37	30.54	16%

from 10.43 to 17.41 (67% ↑). This indicates that the performance of the retrieval-based approach can be effectively improved by considering the commit message history. In addition, hybrid methods such as RACE [55], REACT [73], HACMG [20], and COME [24] also showed good improvements in most evaluation metrics, with REACT [73] showing particularly outstanding improvements in B-Norm and BLEU, reaching 33% and 32% respectively, while COME showed improvements of over 10% in key metrics such as ROUGE-L, METEOR, and Log-MNEXT.

In terms of LLMs, CodeLlama-7B, Llama-3-8B, Deepseek V3, and Deepseek-R1 all exhibit varying degrees of performance enhancement. Especially the Deepseeker series models have achieved significant improvements in all metrics, such as the B-Norm of DeepSeek-R1-Distill-Llama-8B increasing from 5.55 to 13.03 (269% ↑), BLEU increasing from 1.04 to 2.3 (266% ↑), ROUGE-L increasing from 13.08 to 21.64 (171% ↑), METEOR increasing from 15.59 to 24.03 (160% ↑), Log-MNEXT increasing from 13.53 to 19.39 (150% ↑), and BRSA increasing from 26.37 to 30.54 (16% ↑). This result indicates that LLM-based models have strong modeling capabilities in commit message generation tasks, especially when applied HisRag, the quality of the generated commit messages is significantly improved.

Figure 5 shows the relative improvements in commit message generation performance for HisRag-enhanced CMG approaches across the six groups, where a darker color indicates a larger improvement. To better illustrate the improvement of our approach, we add one additional row, Average, that shows the average improvements for all 11 CMG approaches. HisRag improves their performance across metrics in all groups, which demonstrates the effectiveness of HisRag. The results also show that the average improvements tend to be larger for shorter commit messages than for longer ones. From Group 1 (shortest messages) to Group 6 (longest messages), the average improvements in ROUGE-L, METEOR, Log-MNEXT, and BRSA gradually decrease. One possible reason is that generating longer commit messages is more challenging than generating shorter commit messages, which is consistent with the existing study [61].

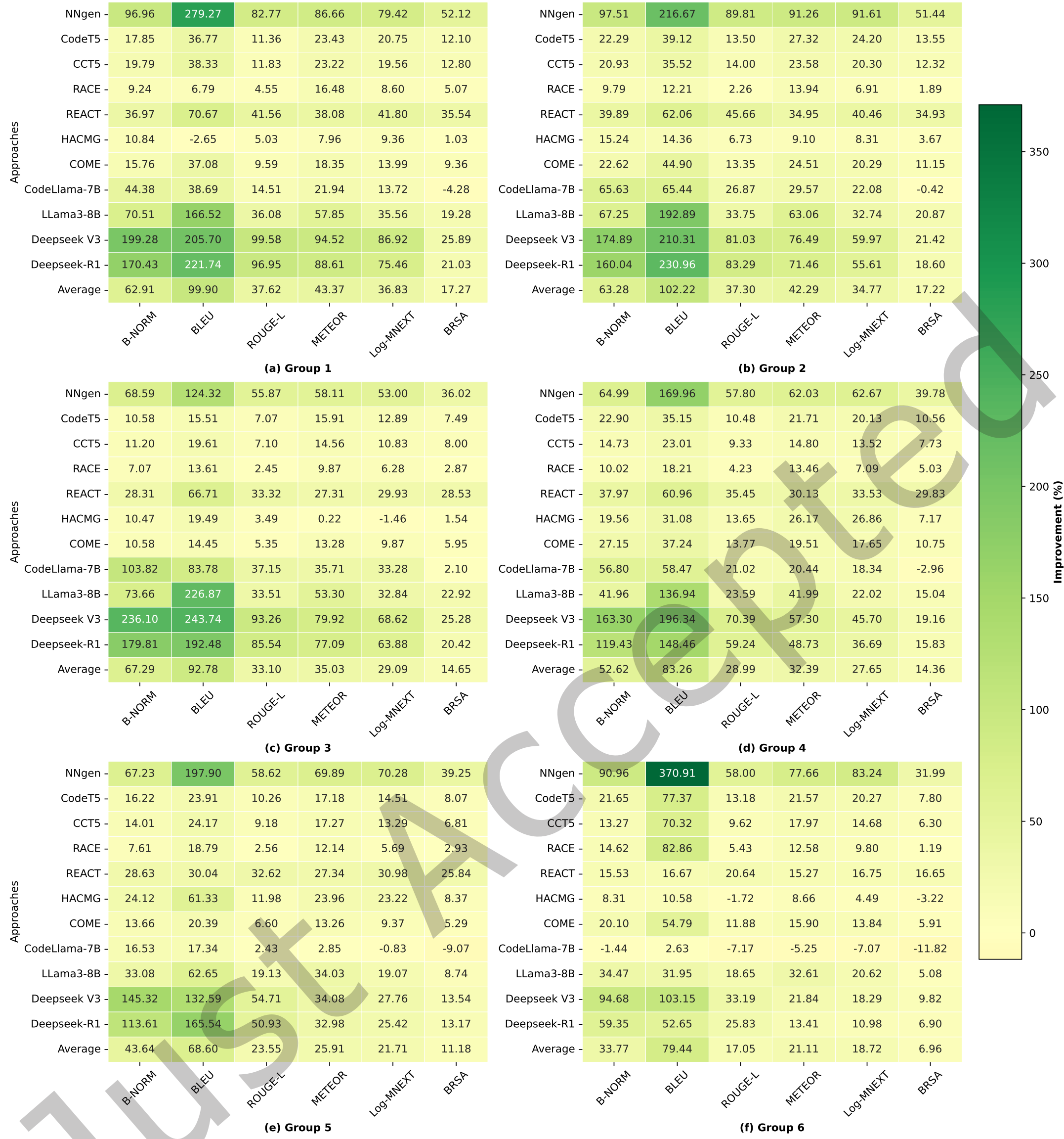
Overall, the experimental results indicate that existing state-of-the-art CMG approaches have better performance when applied to the HisRag paradigm, providing further optimization of this task in the future.

> The results show that HisRag improves the performance of existing SOTA CMG approaches. The average improvements on the B-NORM, BLEU, ROUGE-L, METEOR, Log-MNEXT, and BRSA are 65%, 78%, 38%, 42%, 37%, and 12%. This highlights its effectiveness in enhancing commit message generation.

6.2 RQ2: How do different retrieval sources and retrieval methods affect the performance of the HisRag?

As shown in Table 6, we examine the effectiveness of different retrieval sources and methods in enhancing the performance of HisRag for commit message generation. The results of our experiments show that retrieval

Just Accepted



Just Accepted

Fig. 5. Improvements (%) for HisRag-enhanced CMG approaches in six groups, where the group is split by the token count of the ground truth message according to the frequency.

sources and retrieval methods both play significant roles in determining the quality of generated commit messages. Specifically, when comparing developer-level versus repository-level retrieval, the former consistently outperforms the latter across all evaluation metrics, including B-NORM, BLEU, ROUGE-L, METEOR, Log-MNEXT, and BRSA. This suggests that a finer granularity level of history retrieval, which takes into account individual

Table 6. The generation performance of the HisRag under different retrieval sources and retrieval methods.

Source	Method	**Metric Score(%)**					
		B-NORM	BLEU	ROUGE-L	METEOR	Log-MNEXT	BRSA
Developer	Lexical	22.93	11.00	30.82	26.69	21.95	35.83
Developer	Semantic	22.31	10.54	30.11	26.04	21.27	35.32
Repository	Lexical	21.42	10.35	29.39	25.22	20.69	34.78
Repository	Semantic	21.07	10.00	28.76	24.31	19.78	34.14

developer contributions, leads to more relevant and precise commit message suggestions, ultimately improving the quality of the generated messages.

Further analysis of retrieval methods reveals a noticeable performance gap between lexical-based and semantic-based retrieval methods. Lexical-based retrieval generally provides superior results across all metrics, with higher B-NORM, BLEU, and ROUGE-L scores compared to semantic-based retrieval. The experimental results are consistent with the preliminary experimental results in Section 3.5. This means that the retrieved results and generated results maintain a consistent trend, which suggests whether we can further enhance the quality of generated results by improving the quality of retrieval. Due to limited space, this will leave to future work.

Overall, combining developer-level sources with lexical-based retrieval methods can achieve the best performance on all evaluation metrics. The retrieved results and generated results maintain a consistent trend, and the results emphasize the importance of selecting appropriate retrieval sources and retrieval methods when optimizing HisRag.

6.3 RQ3: How do the number of commit messages from history and modification embedding affect the performance of the HisRag?

We conducted experiments to compare the enhancement effects of retrieved Top-k ($k = 1, 3, 5, 7, 9, 11$) similar history commits to augment the generation performance of SPLMs and LLMs. For SPLMs, we use HisRag-enhanced COME. For LLMs, we use Llama-3-8B as the foundation model. The experimental results of HisRag$_{COME}$ and HisRag$_{Llama3-8B}$ are shown in Figure 6, and it can be seen that the performance increases slightly when k increases. This suggests that retrieving more similar history commits can help the model generate better results. But the count of similar messages from history is not necessarily the bigger the better. For HisRag$_{COME}$, the performance begins to decline when k is greater than 5. For HisRag$_{Llama3-8B}$, there is a slow decline in performance when k is greater than 7.

Figure 6 also shows the impact of using modification embedding. HisRag$_{COME}$ w/o modification means removing the tagged token embedding layer and only using the rearranged token embedding layer for HisRag$_{COME}$. The results show that not using modification embedding will lead to a decrease in performance at different k values, which indicates the effectiveness of modification embedding. This improvement can be attributed to the modification embedding explicitly encodes the structural semantics of code changes and natural language patterns in retrieved historical messages.

HisRag_*Random*$_{Llama3-8B}$ retrieves random historical messages from the same developer's commit history. Compared with HisRag$_{Llama3-8B}$, the results show that using random history performs worse at different numbers of retrieved instances, which indicates the effectiveness of the retrieved history.

Just Accepted

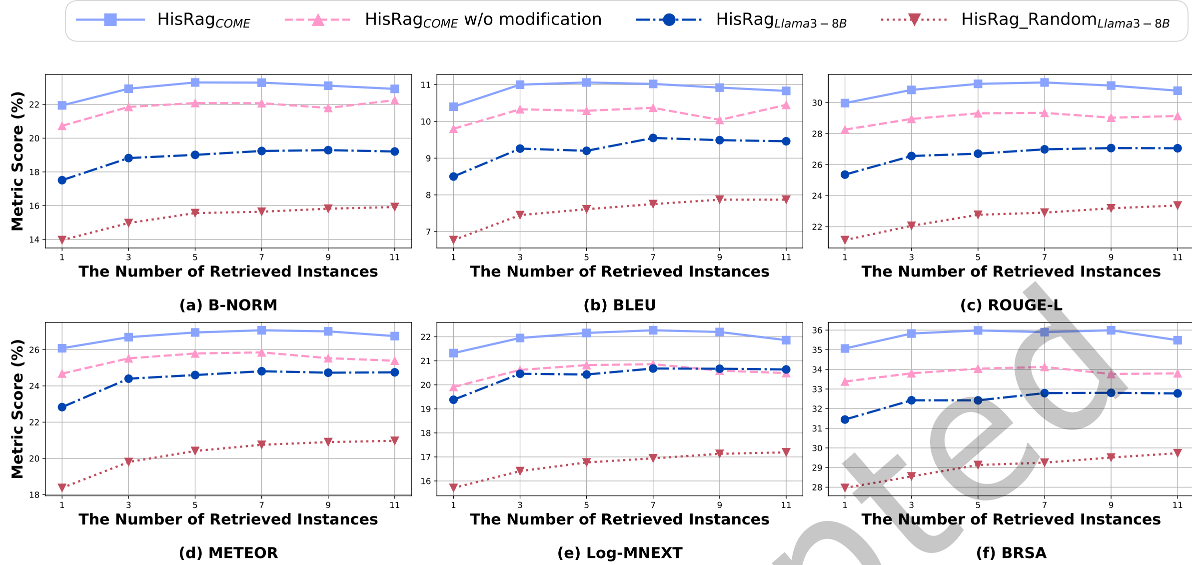


Fig. 6. Performances of HisRag$_{COME}$ and HisRag$_{Llama-3-8B}$ augmented with k retrieved relevant commits in history.

> The results indicate that the number of commit messages can enhance the performance, but not necessarily the larger the better. This suggests that a few highly relevant history commits are more effective than a large number.

6.4 RQ4: How do the human evaluation and manual case analysis reveal the quality and characteristics of the commit messages generated by HisRag?

Automatic evaluation metrics calculate text similarity between the generated text and the reference texts. However, they do not consider the syntax, grammar, and sentence structures of the generated text. To better understand the usefulness of our approach in real-world scenarios, we conduct a human evaluation to further evaluate the usefulness of HisRag in practice.

We invited six experienced developers who were not authors but had a major in computer science and industrial experience in programming (1-4 years) to participate in our survey. Following the existing studies [24, 55], we created a questionnaire containing 50 questions, as shown in Figure 7 (full details are provided in our replication package [3]). We randomly sampled 50 commits from the test set to form the questions. The sampled commit messages range from 1 to 40 tokens (mean: 7.76, median: 7), while the corresponding code changes range from 5 to 1,637 tokens (mean: 283.62, median: 98.5). The observed token length distribution of sampled commits is broadly consistent with that of commits in the full dataset. In the questionnaire, each question contained a code change for a commit, a reference message (Ground Truth), and the messages generated by different CMG approaches. Each participant is required to evaluate the quality of the generated commit messages compared to the reference message. To ensure fair and unbiased evaluations, we shuffle the commit messages generated by different approaches so that the participants do not know which approach each commit message comes from. Following a prior study [55], each participant is asked to score the commit messages considering three aspects: *Informativeness*, *Conciseness*, and *Expressiveness*. *Informativeness* refers to the amount of important information

Just Accepted

about the code change reflected in the commit message. *Conciseness* refers to the extent of extraneous information included in the commit message. *Expressiveness* refers to grammaticality and fluency. Scores range from 0 to 4, with higher scores suggesting better informativeness, conciseness, and expressiveness.

Questionnaire

For each Question, it contains a code change, ground truth message, and several commit messages are generated by different CMG approaches.

Please rate the each commit message on the following three aspects using 0-4 scale. Higher scores suggesting better performance on each aspect. Note that the value of the score can only be taken as 0, 1, 2, 3, 4.

Informativeness refers to the amount of important information about the code change reflected in the commit message.

Conciseness refers to the extent of extraneous information included in the commit message.

Expressiveness refers to grammaticality and fluency.

In addition, please indicate whether you would accept each generated commit message as the final commit message. And **Accpetance**=1 indicates accepted, 0 indicates not accepted.

Question 1

Code change:

```
@@ -1,10 +1,8 @@
## Using Helm Registry Plug-in with a self-signed certificate
+  This document assumes you have deployed [Quay Enterprise with a self-signed certificate.][self-signed]
-  This document assumes you have deployed Quay Enterprise with a self-signed certificate.
-   [Appr](https://github.com/app-registry/appr) makes use of the python requests library, which by default only trusts a standard set of certificates. This prevents helm from interacting with a Quay Enterprise instance that is using a self-signed certificate.
+   [Appr](https://github.com/app-registry/appr) uses a [Python Requests library][python-requests], which trusts only a standard set of certificates by default. This prevents Helm from interacting with a Quay Enterprise instance that is using a self-signed certificate.
...
```

Ground Truth Message: quay-enterprise: fixes for custom-cert-kube-apps.md

Approach	Generated Commit message	Informativeness(0,1,2,3,4)	Conciseness(0,1,2,3,4)	Expressiveness(0,1,2,3,4)	Acceptance(0/1)
Approach 1	quay-enterprise: update custom-cert-kube-apps.md				
Approach 2	Quay Enterprise: update custom-cert-kube-apps.md				
Approach 3	Update custom-cert-kube-apps.md to include instructions for using Helm with				
Approach 4	Quay Enterprise: update custom-cert-kube-apps.md				
Approach 5	qe: add doc on using helm registry plug-in with a self-signed certificate				

Question 2

...

Question 50

...

Fig. 7. The questionnaire of human evaluation.

Furthermore, we use the *Acceptance rate* to measure whether a commit message is accepted by the developer in practice. Specifically, a commit message that is accepted means it is committed without any revision. It is worth noting that the commit messages generated by different approaches may be the same, but as long as the participants believe that the generated message can be used as the final commit message, then it is accepted. The final acceptance rate of different approaches is the proportion of accepted commit messages among all evaluated commits. The participants are required to evaluate whether commit messages generated by different approaches can be accepted (0/1).

We selected the generated results of the four CMG approaches and an industrial CMG tool for human evaluation. For CMG approaches, we choose two representative approaches (Llama-3-8B and COME) and their enhanced versions (HisRag$_{Llama-3-8B}$ and HisRag$_{COME}$) by HisRag for the following reasons. First, as shown in Table 5, these two approaches have better performance across all CMG approaches. Second, including all 11 baselines in human evaluation would introduce significant overhead and participant fatigue, leading to potential biases or reduced evaluation quality. Therefore, we only select these four approaches for human evaluation.

Just Accepted

For the industrial CMG tool, we selected GitHub Copilot [4] (version 1.388.0), a VSCode extension that supports commit message generation, to compare our approach with the State-of-Practice commercial tool. To do so, we implemented a script to generate a commit message given a specific commit by using GitHub Copilot. Figure 8 shows the process of generating a commit message for using the GitHub Copilot extension in our experiments. For each sampled commit, we first clone its corresponding repository locally. Then we restore the code changes of this commit with the command "git restore". Finally, we open the repository folder with VSCode and use the GitHub Copilot extension to generate the commit message. The generated commit message is recorded for human evaluation.

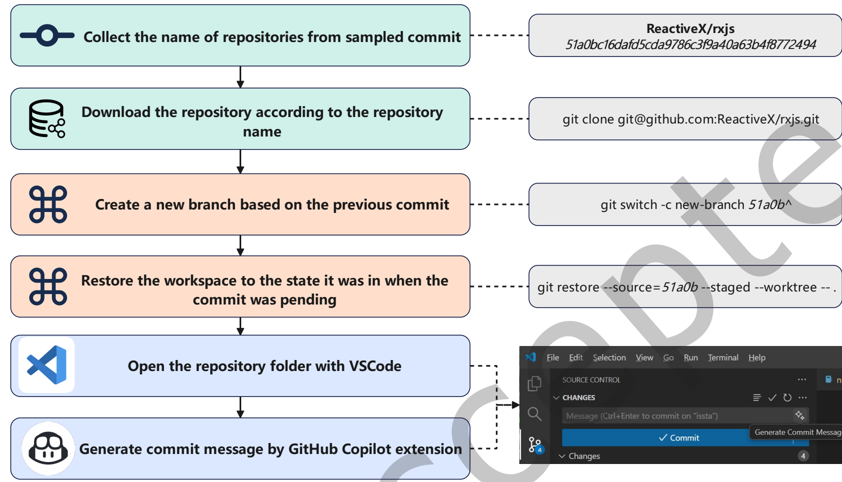


Fig. 8. The process of generating a commit message for using the GitHub Copilot extension

Just Accepted

Table 7 shows the result of the human evaluation. The performance of each approach is measured by the average score of all its generated commit messages from four aspects: *Informativeness*, *Conciseness*, *Expressiveness*, and *Acceptance rate*. We also reported inter-annotator agreement using Kendall's coefficient of concordance. The agreement is weak to moderate across dimensions, suggesting that the subjective ratings should be interpreted with caution. We also apply the Friedman test [21] across all approaches. The results indicate statistically significant overall differences for all five evaluation dimensions.

Overall, HisRag consistently enhances all three dimensions with a higher acceptance rate over its paired baselines (HisRag_{COME} vs COME and $\text{HisRag}_{Llama-3-8B}$ vs Llama-3-8B). To confirm our observations, we further conduct pairwise comparisons using the Wilcoxon signed-rank test [48] with Holm correction [27], and report the results in Table 8. Although HisRag variants are not best across all approaches, the post-hoc analysis shows that HisRag significantly improves Conciseness compared to Llama-3-8B, and improves Informativeness and Acceptance rate compared to COME. These results suggest that HisRag can generate higher-quality commit messages in specific dimensions, and indicate its potential practical usability when compared to the GitHub Copilot.

To make a deep analysis, we also give an example of commit messages generated by different CMG approaches and GitHub Copilot. As shown in Figure 9, it can be seen that HisRag effectively enhances the performance of existing CMG approaches by improving both the relevance and style consistency of the generated messages. Figure 9 also shows the metrics scores (B-Norm, BLEU, ROUGE-L, METEOR, Log-MNEXT, and BRSA) associated

Table 7. Results of human evaluation. Inter-rate agreement is measured using Kendall's coefficient of concordance ($\alpha = 0.05$). Overall statistical differences among approaches are assessed using the Friedman test ($\alpha = 0.05$).

Approach	Informativeness	Conciseness	Expressiveness	Acceptance rate (%)
GitHub Copilot	3.25	2.54	3.07	62.33
Llama3-8B	2.81	2.74	2.74	53.33
COME	2.68	3.15	2.75	57.67
HisRag$_{Llama-3-8B}$	2.83	3.03	2.8	56.33
HisRag$_{COME}$	2.8	3.18	2.83	65.33
Kendall's coefficient	5.04×10^{-1}	2.293×10^{-1}	2.133×10^{-1}	3.044×10^{-1}
p-value (Kendall's coefficient)	<0.05	<0.05	<0.05	<0.05
Friedman χ^2	94.67	169.03	56.16	16.18
p-value (Friedman)	<0.05	<0.05	<0.05	<0.05

Table 8. Pairwise statistical significance of human evaluation results. Adjusted p-values from pairwise Wilcoxon signed-rank tests with Holm correction ($\alpha = 0.05$).

Approach *vs* Approach	Informativeness	Conciseness	Expressiveness	Acceptance rate
GitHub Copilot *vs* Llama3-8B	<0.05	<0.05	<0.05	0.60
GitHub Copilot *vs* HisRag$_{Llama-3-8B}$	<0.05	<0.05	<0.05	0.50
GitHub Copilot *vs* COME	<0.05	<0.05	<0.05	0.93
GitHub Copilot *vs* HisRag$_{COME}$	<0.05	<0.05	<0.05	1
HisRag$_{COME}$ *vs* COME	<0.05	0.57	0.56	<0.05
HisRag$_{COME}$ *vs* HisRag$_{Llama-3-8B}$	1.00	<0.05	1.00	<0.05
HisRag$_{COME}$ *vs* Llama3-8B	1.00	<0.05	0.72	<0.05
HisRag$_{Llama-3-8B}$ *vs* Llama3	1.00	<0.05	1.00	1.00
HisRag$_{Llama-3-8B}$ *vs* COME	<0.05	<0.05	1.00	1.00
COME *vs* Llama3-8B	0.12	<0.05	1.00	0.93

with the generated results. Compared with the baseline CMG approaches, the metric scores of HisRag enhanced approaches have relatively higher metric scores, which is consistent with the manual analysis.

Notably, HisRag-generated messages closely align with the ground truth in terms of structure and content, reflecting a better understanding of developer-specific tasks. Specifically, none of the baseline CMG approaches correctly include the commit type (*feat*) or scope (*cli*). For GitHub Copilot, although it successfully generates the correct type "feat", it fails in generating the correct scope "cli" and ignores the semantics of "uploading artifacts". However, these elements are essential and can be acquired from the retrieved commit message history of the developer and repository. In contrast, the variants enhanced by HisRag successfully include these key elements, closely aligning with the ground truth. This enhancement stems from retrieving relevant historical messages at the developer and repository levels, which preserve a consistent writing style that is often absent from coarse-grained, dataset-level retrieval.

[2]https://github.com/apache/camel-k/commit/f5e39aaa4dcf97ce9ef9fd4e0d8517ca0a35f61f

Just Accepted

Code Change	pkg/cmd/run.go
	@@ -925,10 +929,14 @@ func (o *runCmdOptions) uploadPomFromJar(gav maven.Dependency, path string, plat
	}
	}
	if pomExtracted {
	+ if o.skipPom() {
	+ o.PrintfVerboseOutf(cmd, "Skipping uploading extracted POM from %s \n", path)
	+ } else {
	gav.Type = "pom"
	// Swallow error as this is not a mandatory step
	o.uploadAsMavenArtifact(gav, pomPath, platform, ns, options, cmd)
	}
	+ }
	return nil
	})
	gav.Type = "jar"
	...						

Commit History$_{developer}$	• feat(cli): Add option to specify registry url when uploading artifacts • feat(cli): Add option to skip checksum files when uploading artifacts • fix(ci): update last released version to 1.9.2 • ...						
Commit History$_{repository}$	• feat(cli): Add option to skip checksum files when uploading artifacts • Fix Update command line option • Fix Let's make the default targetpath relative to it's parent directory • ...						
Commit$_{Database}$	• feat(run/property): new file format as build-property • chore(cli): Improve run command --dependency option description • feat(cmd): modeline properties support • ...						
Ground Truth	feat(cli): Add option to skip uploading POM from JAR when uploading artifacts to the image registry						
		B-Norm	**BLEU**	**ROUGE-L**	**METEOR**	**Log-MNEXT**	**BRSA**
COME	chore(e2e): Add skipPOM flag	0.08	0.07	0.09	0.20	0.04	0.26
Llama-3-8B	Update Maven wagon to skip POM for local files	0.06	0.06	0.23	0.08	0.11	0.30
DeepSeeker V3	Add `skipPOM` option for local file dependencies and update test case	0.06	0.06	0.14	0.05	0.07	0.24
DeepSeek-R1-Distill-Llama-8B	Add support for skipping POM when adding dependencies	0.05	0.04	0.24	0.11	0.14	0.28
GitHub Copilot	feat(cmd): Add skipPOM option for local file dependencies in run command	0.14	0.14	0.21	0.29	0.07	0.37
HisRag$_{COME}$	feat(cli): Add option to skip pom files when uploading artifacts	0.49	0.48	0.71	0.65	0.53	0.71
HisRag$_{Llama-3-8B}$	feat(cli): Add option to skip POM files when uploading local files	0.47	0.46	0.62	0.59	0.45	0.63
HisRag$_{DeepSeeker V3}$	feat(cli): Add `skipPOM` option for local file dependencies	0.27	0.27	0.31	0.36	0.17	0.44
HisRag$_{DeepSeek-R1-Distill-Llama-8B}$	feat(cli): Add option to skip POM files when adding dependencies	0.44	0.43	0.57	0.54	0.37	0.56

Fig. 9. An example of generated commit messages [2]. Ground truth is the commit message written by the developer.

Table 9. The computational cost of retrieving from different sources with different methods.

Method	Source	Average preparation time (s)	Average query time (s)
Lexical-based	Developer-level	0.00182	0.074
	Repository-level	0.00182	0.09
	Dataset-level	0.00182	13.05
Semantic-based	Developer-level	0.107	0.032
	Repository-level	0.107	0.057
	Dataset-level	0.107	0.264

Human evaluation indicates that HisRag improves the quality of commit messages compared to its corresponding baselines across all evaluation dimensions, with statistically significant gains observed in specific aspects. When compared with industry CMG tools, HisRag shows comparable ability on the acceptance rate. Overall, the results suggest the promising practical usability of HisRag.

Just Accepted

6.5 RQ5: Is HisRag computationally efficient in real-world usage settings?

To verify the practical feasibility of HisRag in deployment scenarios such as CI/CD pipelines, we evaluate the computational cost of retrieving from different sources with different methods. We measure two metrics for each instance in the test set: (1) the *preparation time*, including embedding generation (for semantic-based retrieval methods) or index construction (for lexical-based retrieval methods); and (2) the *query time*, i.e., the latency to compute the similarity score with candidate instances. We consider both lexical-based and semantic-based retrieval methods across three retrieval sources: developer-level, repository-level, and dataset-level. The implementations of retrieval methods remain the same with Section 3.2. Notably, we didn't utilize approximate retrieval tools for fair comparison. Since the embeddings and corpus are processed in advance, the average preparation time of retrieval sources under the same retrieval method (lexical-based or semantic-based) is the same.

Table 9 presents the computational cost of retrieving from different sources with different methods. First, although the average preparation time of the lexical-based retrieval methods is shorter than that of the semantic-based retrieval methods, their average query time is longer compared to semantic-based methods. Therefore, the type of retrieval method to be used in practical applications should depend on the specific needs. Semantic-based retrieval methods offer faster querying, but require more preparation time.

Second, developer-level and repository-level retrieval sources exhibit low latency for query time under both lexical- and semantic-based retrieval methods. Compared with the dataset-level retrieval source, they achieved 92× and 75× improvements in query efficiency, respectively. This indicates the advantage of retrieving less and the promise of applying to the real-world development process.

> Fine-grained retrieval (developer-level and repository-level) yields substantial query-time savings, supporting HisRag is computationally viable for deployment in real-world usage settings.

7 Discussion

7.1 Threats to Validity

Internal validity. A potential threat to internal validity is the implementation of different approaches. To mitigate this threat, if the replication packages of the related approaches are available and executable (NNGen [46], CCT5 [36], RACE [55], REACT [73], HACMG [20], and COME [24]), we directly reuse them and strictly follow their parameter settings in their original implementations; if the approaches lack replication packages (CodeT5 [67], CodeLlama-7B [52], Llama3-8B [2], Deepseek V3 [40], Deepseek-R1 [23]), we re-implement the techniques strictly following the prompt used in the study of Eliseeva et.al [20], where they constructed zero-shot prompting for LLM-based commit message generation. The presence of bugs in the source code implemented for this study poses a potential threat to validity. To mitigate this threat, we thoroughly tested the code and followed software engineering best practices to ensure that it was written in a clear and organized manner.

Construct validity. A potential threat to construct validity is the evaluation metrics used in the automatic evaluation. To mitigate this threat, we adopted lexical-overlap metrics (BLEU, B-Norm, METEOR, ROUGE-L, and Log-MNEXT) and a semantic-based metric (BRSA) widely used in previous work on commit message generation [14, 16, 20, 26]. Following existing studies [24, 55], we also performed human evaluation considering three aspects (informativeness, conciseness, expressiveness) to further mitigate this threat.

External validity. A potential threat to external validity is generalizability. To mitigate this threat, we conducted our study on 11 CMG approaches across four types (retrieval-based, learning-based, hybrid, and LLM-based) and used a multilingual CMG dataset CommitChronicle to verify the effectiveness of HisRag. Another potential threat to external validity is the generalizability to different programming languages. To mitigate this threat,

Just Accepted

we follow prior studies [22, 64] and select the repositories across 17 different programming languages to ensure generalization.

7.2 Implications

This study provides meaningful implications for future research on history retrieval-augmented CMG and practical application. Our research findings show that retrieving from commit message history can significantly help developers in automated commit message generation tasks and enhance the quality of commit messages. Compared with industrial CMG tools such as GitHub Copilot [4], HisRag provides more personalization by leveraging historical commit patterns. Without the commit history retrieval, commit messages may be less aligned with the writing style of the developer or the repository. With our approach, developers can benefit from personalized commit suggestions that reflect their historical writing style.

HisRag can be easily integrated into pipelines of existing CMG tools. At the same time, developers or companies can deploy their own models enhanced by HisRag to avoid potential privacy and sustainability concerns using proprietary models (i.e., GPT-4).

7.3 Limitations and Future Work

Although HisRag can further enhance the performance of existing CMG approaches by retrieving commit message history, we have found that this paradigm may still have some potential limitations. One of the limitations is that HisRag can only enhance the performance of commits with commit message history, and it is ineffective for commits without commit message history or when commit message history is sparse. This can be seen as a cold start problem, and it may occur in less active repositories. However, this limitation can be alleviated by adding an external retrieval database [73]. In this paper, we focus on the scenario with commit history and will delve into scenarios without historical data in the future.

The second limitation of HisRag is the impact of the retrieval method. The retrieved commit message from history may not always be relevant, and in some cases, it could negatively impact generation performance. When the generation model relies on historical data, incorrect or irrelevant retrievals may lead to misunderstandings or information redundancy, ultimately affecting the quality of the generated messages. Moreover, this study does not consider more complex settings such as selective retrieval [68] or iterative retrieval [72], which may limit adaptability to specific contexts or tasks. Future research could explore more sophisticated retrieval mechanisms to further enhance the performance and flexibility of the generation model.

The third limitation of HisRag is that the efficiency of the retrieval process in practical deployment depends on the size of the commit history. In real-world scenarios, a large amount of commit history can make retrieval less efficient. This situation is similar to the scenarios where approaches need a large amount of history [41, 57], and we believe that existing search strategies [11, 57] can help mitigate this issue.

In future work, we plan to develop a CMG extension that implements HisRag as a local history-aware retrieval module, providing personalized commit message recommendations.

8 Related Work

Existing work on CMG can be categorized into template-based approaches, retrieval-based approaches, learning-based approaches, hybrid approaches, and LLM-based approaches [75].

Template-based approaches [9, 12, 17, 18, 54] analyze code changes and generate commit messages with predefined patterns. Generally, template-based techniques are only effective when the situation fully conforms to predefined rules, making them less applicable as a universal method due to the diversity of commit messages.

Retrieval-based approaches [26, 28, 46, 63] utilize information retrieval techniques to adopt existing commit messages from similar code changes. For instance, given a code change as a query, Liu et al. [46] selected the

Just Accepted

most similar code changes from the training set using cosine similarity and BLEU. Similarly, Huang et al. [28] used syntactic similarity and semantic similarity as similarity measures.

Learning-based approaches treat CMG as a translation problem and use neural machine translation models to transform code changes into commit messages [16, 30, 36, 42, 44, 45, 53, 70, 75]. Dong et al. [16] proposed FIRA which does not directly input old and new versions of code into the translation model but represents code changes as abstract syntax trees and graphs by considering editing operations and subtokens in code changes. Lin et al. [36] proposed a pre-trained model called CCT5, which can both leverage the domain knowledge hidden in the large amount of unlabelled data and ensure the learned knowledge is adequately exploited during fine-tuning on code-change-related tasks. Liu et al. [45] proposed a novel approach named CCRep, which uses a query back to highlight the changed code and adaptively capture related context information to learn code change representations.

Hybrid approaches combine the previous approaches to generate commit messages. Liu et al. [43] proposed using an abstract syntax tree to represent code changes and integrating both retrieved and generated messages through a hybrid ranking system. Wang et al. [62] proposed a model that combines the advantages of retrieval and learning-based approaches for CMG. Shi et al. [55] proposed a novel exemplar-based neural CMG model, which uses similar commits as examples to guide the neural network to generate informative and readable commit messages.

LLM-based approaches utilize the powerful generative ability of LLMs to generate the commit message. Zhang et al. [74] conducted a preliminary study to evaluate the performance of LLMs. Lopes et al. [47] evaluated the performance of ChatGPT with the automated CMG models. Eliseeva et al.[20] used GPT-3.5-turbo to generate commit messages with the commit message history of the developer. In addition, there are several studies [61, 69, 71, 73] that have utilized In-Context learning (ICL) to enhance the performance of LLMs on the CMG task. Although these approaches have shown promising results, it is still unexplored whether the commit history will affect the commit message generation. This paper presents a comprehensive empirical study exploring LLMs' capabilities augmented with commit message history in the commit message generation task.

9 Conclusion

The commit history provides important context when developers handle different software development tasks in different repositories. The history retrieval-augmented commit message generation has not been fully explored yet. To fill this gap, we propose HisRag, a novel paradigm to use commit history for commit message generation. HisRag contains three stages: history retrieval stage, augmentation stage, and generation stage. The extensive experiments show that HisRag can significantly boost the performance of existing CMG approaches whether the model is based on SPLMs or LLMs. The ablation study shows the effectiveness of each stage. Our package is available at [3], which contains the dataset, artifacts, and scripts for reproduction.

10 Acknowledgments

This research / project is supported by the National Science Foundation of China (No.62476289), the National Science Foundation of China (No.62372398) and the National Research Foundation, under its Investigatorship Grant (NRF-NRFI08-2022-0002). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

[1] [n. d.]. all-MiniLM-L6-v2: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.
[2] [n. d.]. Llama 3: https://ai.meta.com/llama/.
[3] 2025. Replication Package: https://zenodo.org/records/17798074.
[4] 2025. GitHub Copilot in VS Code: https://code.visualstudio.com/docs/copilot/overview.

Just Accepted

[5] Toufique Ahmed, Kunal Suresh Pai, Premkumar Devanbu, and Earl Barr. 2024. Automatic semantic augmentation of language model prompts (for code summarization). In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[6] Fatih Kadir Akın. 2023. Awesome chatgpt prompts, GitHub, 2023. *URL: https://github. com/f/awesome-chatgpt-prompts, online* (2023).

[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*. 65–72.

[8] Christian Bird, Peter C. Rigby, Earl T. Barr, David J. Hamilton, Daniel M. German, and premkumar devanbu. 2009. The promises and perils of mining git. (2009), 1–10.

[9] Raymond PL Buse and Westley R Weimer. 2010. Automatically documenting program changes. In *Proceedings of the 25th IEEE/ACM international conference on automated software engineering*. 33–42.

[10] Kuljit Kaur Chahal and Munish Saini. 2018. Developer Dynamics and Syntactic Quality of Commit Messages in OSS Projects. In *Open Source Systems: Enterprise Software and Solutions - 14th IFIP WG 2.13 International Conference, OSS 2018, Athens, Greece, June 8-10, 2018, Proceedings (IFIP Advances in Information and Communication Technology, Vol. 525)*, Ioannis Stamelos, Jesús M. González-Barahona, Iraklis Varlamis, and Dimosthenis Anagnostopoulos (Eds.). Springer, 61–76. doi:10.1007/978-3-319-92375-8_6

[11] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. Spann: Highly-efficient billion-scale approximate nearest neighborhood search. *Advances in Neural Information Processing Systems* 34 (2021), 5199–5212.

[12] Luis Fernando Cortés-Coy, Mario Linares-Vásquez, Jairo Aponte, and Denys Poshyvanyk. 2014. On automatically generating commit messages via summarization of source code changes. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. IEEE, 275–284.

[13] Michael Denkowski and Alon Lavie. 2010. Meteor-next and the meteor paraphrase tables: Improved evaluation support for five target languages. In *Proceedings of the joint fifth workshop on statistical machine translation and MetricsMATR*. 339–342.

[14] Samanta Dey, Venkatesh Vinayakarao, Monika Gupta, and Sampath Dechu. 2022. Evaluating commit message generation: to BLEU or not to BLEU?. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*. 31–35.

[15] Jinhao Dong, Yiling Lou, Dan Hao, and Lin Tan. 2023. Revisiting learning-based commit message generation. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 794–805.

[16] 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*. 970–981.

[17] Natalia Dragan, Michael L Collard, Maen Hammad, and Jonathan I Maletic. 2011. Using stereotypes to help characterize commits. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 520–523.

[18] Natalia Dragan, Michael L Collard, and Jonathan I Maletic. 2006. Reverse engineering method stereotypes. In *2006 22nd IEEE International Conference on Software Maintenance*. IEEE, 24–34.

[19] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N Nguyen. 2013. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 422–431.

[20] Aleksandra Eliseeva, Yaroslav Sokolov, Egor Bogomolov, Yaroslav Golubev, Danny Dig, and Timofey Bryksin. 2023. From commit message generation to history-aware commit message completion. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 723–735.

[21] Milton Friedman. 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association* 32, 200 (1937), 675–701.

[22] Konstantin Grotov, Sergey Titov, Vladimir Sotnikov, Yaroslav Golubev, and Timofey Bryksin. 2022. A large-scale comparison of Python code in Jupyter notebooks and scripts. In *Proceedings of the 19th International Conference on Mining Software Repositories*. 353–364.

[23] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).

[24] Yichen He, Liran Wang, Kaiyi Wang, Yupeng Zhang, Hang Zhang, and Zhoujun Li. 2023. COME: Commit Message Generation with Modification Embedding. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 792–803.

[25] Abram Hindle, Daniel M. German, and Ric Holt. 2008. What do large commits tell us?: a taxonomical study of large commits. In *Proceedings of the 2008 international working conference on Mining software repositories* (Leipzig Germany, 2008-05-10). ACM, 99–108. doi:10.1145/1370750.1370773 TLDR: A case study that included the manual classification of large commits of nine open source projects is performed and it is shown that large commits are more perfective while small commit are more corrective..

[26] 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*. 518–529.

[27] Sture Holm. 1979. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics* (1979), 65–70.

[28] Yuan Huang, Nan Jia, Hao-Jie Zhou, Xiang-Ping Chen, Zi-Bin Zheng, and Ming-Dong Tang. 2020. Learning human-written commit messages to document code changes. *Journal of Computer Science and Technology* 35 (2020), 1258–1277.

Just Accepted

[29] Aaron Imani, Iftekhar Ahmed, and Mohammad Moshirpour. 2024. Context Conquers Parameters: Outperforming Proprietary LLM in Commit Message Generation. *arXiv preprint arXiv:2408.02502* (2024).

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

[31] Siyuan Jiang and Collin McMillan. 2017. Towards automatic generation of short summaries of commits. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, 320–323.

[32] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization, Vol. abs/1412.6980.

[33] Cong Li, Zhaogui Xu, Peng Di, Dongxia Wang, Zheng Li, and Qian Zheng. 2024. Understanding code changes practically with small-scale language models. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 216–228.

[34] Jiawei Li and Iftekhar Ahmed. 2023. Commit message matters: Investigating impact and evolution of commit message quality. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 806–817.

[35] Jiawei Li, David Faragó, Christian Petrov, and Iftekhar Ahmed. 2024. Only diff is not enough: Generating commit messages leveraging reasoning and action of large language model. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 745–766.

[36] Bo Lin, Shangwen Wang, Zhongxin Liu, Yepang Liu, Xin Xia, and Xiaoguang Mao. 2023. 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*. 1509–1521.

[37] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[38] Chin-Yew Lin and Franz Josef Och. 2004. Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *Proceedings of the 42nd annual meeting of the association for computational linguistics (ACL-04)*. 605–612.

[39] Mario Linares-Vásquez, Luis Fernando Cortés-Coy, Jairo Aponte, and Denys Poshyvanyk. 2015. Changescribe: A tool for automatically generating commit messages. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. IEEE, 709–712.

[40] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024).

[41] Fang Liu, Clement Yu, and Weiyi Meng. 2004. Personalized web search for improving retrieval effectiveness. *IEEE Transactions on knowledge and data engineering* 16, 1 (2004), 28–40.

[42] Qin Liu, Zihe Liu, Hongming Zhu, Hongfei Fan, Bowen Du, and Yu Qian. 2019. Generating commit messages from diffs using pointer-generator network. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 299–309.

[43] Shangqing Liu, Cuiyun Gao, Sen Chen, Lun Yiu Nie, and Yang Liu. 2020. Atom: Commit message generation based on abstract syntax tree and hybrid ranking. *IEEE Transactions on Software Engineering* 48, 5 (2020), 1800–1817.

[44] Shuo Liu, Jacky Keung, Zhen Yang, Fang Liu, Qilin Zhou, and Yihan Liao. 2024. Delving into Parameter-Efficient Fine-Tuning in Code Change Learning: An Empirical Study. *arXiv preprint arXiv:2402.06247* (2024).

[45] Zhongxin Liu, Zhijie Tang, Xin Xia, and Xiaohu Yang. 2023. Ccrep: Learning code change representations via pre-trained code model and query back. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 17–29.

[46] 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*. 373–384.

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

[48] Wilcoxon F Kotz S Johnson NL. 1992. Individual comparisons by ranking methods Breakthroughs in statistics.

[49] 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*. 311–318.

[50] Soumaya Rebai, Marouane Kessentini, Vahid Alizadeh, Oussama Ben Sghaier, and Rick Kazman. 2020. Recommending refactorings via commit message analysis. *Information and Software Technology* 126 (2020), 106332.

[51] Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval* 3, 4 (2009), 333–389.

[52] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).

[53] Maximilian Schall, Tamara Czinczoll, and Gerard de Melo. 2024. CommitBench: A Benchmark for Commit Message Generation. *arXiv preprint arXiv:2403.05188* (2024).

[54] 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 *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. IEEE, 103–112.

[55] Ensheng Shi, Yanlin Wang, Wei Tao, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2022. Race: Retrieval-augmented commit message generation. *arXiv preprint arXiv:2203.02700* (2022).

[56] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie Yan Liu. 2020. MPNet: Masked and Permuted Pre-training for Language Understanding. (2020).

Just Accepted

[57] Bin Tan, Xuehua Shen, and ChengXiang Zhai. 2006. Mining long-term search history to improve search accuracy. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 718–723.

[58] Wei Tao, Yanlin Wang, Ensheng Shi, Lun Du, Shi Han, Hongyu Zhang, Dongmei Zhang, and Wenqiang Zhang. 2021. On the evaluation of commit message generation models: An experimental study. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 126–136.

[59] 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*. 2389–2401.

[60] Bei Wang, Meng Yan, Zhongxin Liu, Ling Xu, Xin Xia, Xiaohong Zhang, and Dan Yang. 2021. Quality assurance for automated commit message generation. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 260–271.

[61] Guoqing Wang, Zeyu Sun, Jinhao Dong, Yuxia Zhang, Mingxuan Zhu, Qingyuan Liang, and Dan Hao. 2025. Is It Hard to Generate Holistic Commit Message? *ACM Transactions on Software Engineering and Methodology* 34, 2 (2025), 1–28.

[62] Haoye Wang, Xin Xia, David Lo, Qiang He, Xinyu Wang, and John Grundy. 2021. Context-aware retrieval-based deep commit message generation. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 4 (2021), 1–30.

[63] Liran Wang, Xunzhu Tang, Yichen He, Changyu Ren, Shuhua Shi, Chaoran Yan, and Zhoujun Li. 2023. Delving into commit-issue correlation to enhance commit message generation models. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 710–722.

[64] Tongjie Wang, Yaroslav Golubev, Oleg Smirnov, Jiawei Li, Timofey Bryksin, and Iftekhar Ahmed. 2021. Pynose: A test smell detector for python. In *2021 36th IEEE/ACM international conference on automated software engineering (ASE)*. IEEE, 593–605.

[65] Weishi Wang, Yue Wang, Shafiq Joty, and Steven CH 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*. 146–158.

[66] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers. 33 (2020).

[67] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859* (2021).

[68] Di Wu, Wasi Uddin Ahmad, Dejiao Zhang, Murali Krishna Ramanathan, and Xiaofei Ma. 2024. Repoformer: Selective Retrieval for Repository-Level Code Completion. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net. https://openreview.net/forum?id=moyG54Okrj

[69] Yifan Wu, Yunpeng Wang, Ying Li, Wei Tao, Siyu Yu, Haowen Yang, Wei Jiang, and Jianguo Li. 2025. An Empirical Study on Commit Message Generation using LLMs via In-Context Learning. *arXiv preprint arXiv:2502.18904* (2025).

[70] Shengbin Xu, Yuan Yao, Feng Xu, Tianxiao Gu, Hanghang Tong, and Jian Lu. 2019. Commit message generation for source code changes. In *IJCAI*.

[71] 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 preprint arXiv:2404.14824* (2024).

[72] Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation. *CoRR* abs/2303.12570 (2023).

[73] Linghao Zhang, Hongyi Zhang, Chong Wang, and Peng Liang. 2024. RAG-Enhanced Commit Message Generation. *CoRR* abs/2406.05514 (2024). arXiv:2406.05514 doi:10.48550/ARXIV.2406.05514

[74] Linghao Zhang, Jingshu Zhao, Chong Wang, and Peng Liang. 2024. Using Large Language Models for Commit Message Generation: A Preliminary Study. *arXiv preprint arXiv:2401.05926* (2024).

[75] 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* (2024).

Received 9 August 2025; revised 10 December 2025; accepted 7 January 2026

Just Accepted