Exploring the Correlation between Commit and Issue for Automatic Commit Message Generation

Anonymous Author(s)*

ABSTRACT

Commit message plays a crucial role in the process of software development, which provides developers with concise descriptive information of a commit. Researchers have proposed many methods for automatically generating commit messages from code change, including retrieval-based, learning-based and hybrid techniques. Existing automatic commit message generation (ACMG) approaches either use sequentiality code change as input or attach some data structures generated by code (such as abstract syntax tree). This can only tell the model how the code was modified in a commit, but not the reason for this commit. Therefore, results generated by these existing methods often lack vital rational information. In fact, a commit is usually explicitly or implicitly associated with an issue (or bug report). High-quality issues often provide valuable information that explains why the code should be modified, such as unexpected program behaviour or expected bug fixes. Undoubtedly, if such a correlation between commit and issue can be introduced into the CMG process, the generated results will be more reasonable and accurate.

In this work, we conduct a detailed research about identifying, extracting, and utilizing valuable correlations between commits and issues to generate more rational commit messages. First, we collect a large commit-issue parallel dataset **COMIS** (**com**mit-**is**sue) to figure out how the correlation arises. After clarifying what is rational information in issues, we propose a novel paradigm ***ExGroFi*** (**E**xtration, **G**rounding, **F**ine-tune) to extract it and to improve the performance of commit message generation models by semantic grounding. To evaluate whether our approach is effective, we perform comprehensive experiments with various learning-based and hybrid ACMG models. The results show that the performance of state-of-the-art methods can be significantly improved by introducing fine-grained issue information with ***ExGroFi***.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

datasets, neural networks, gaze detection, text tagging

ASE '23, September 11–15, 2023, Kirchberg, Luxembourg

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Anonymous Author(s). 2023. Exploring the Correlation between Commit and Issue for Automatic Commit Message Generation. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (ASE '23).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

When developers submit a commit to a codebase, a clear, concise, and informative commit message is supposed to be attached. It can be thought of as a natural language summary of code change [2, 3]. A well-written commit message can help others understand how the changes were made and why they were made [2, 17, 23]. Therefore, commit message assumes significant importance in the process of collaborative software development [2].

However, when the code change is complex, it is not easy to write a commit message that can summarize the modifications at a high level [18]. Which makes human-written commit messages often suffer from problems of ambiguous meaning and irrational information [5, 15, 16, 23]. In addition, in the situation of increasingly fast-paced modern software development, manually writing high-quality commit messages will also be time-consuming [21]. According to Loyola et al. [15], 16% messages in a widely used dataset CommitGen [9] are noisy, which means they contain little information about the commit.

To simplify the process of writing commit messages, researchers have proposed several techniques for automatic commit message generation (ACMG) . These approaches model ACMG as a summarization or translation task, where input is code change and output is a paragraph of descriptive natural language. Mainstream generation methods can be categorized into retrieval-based [8, 14], learning-based [4, 9, 12, 15, 18, 26] and hybrid models [13, 20, 24]. Retrival-based models calculate a similarity score between the requested input and other code changes in the training set, then utilize a ranking algorithm to select the code change with the highest similarity and use its corresponding commit message as output. Learning-based models first use a trained encoder to embed the input code change into semantic space. Then, a decoder or pointer-network will be used to generate output in natural language space according to the representation of code change provided by encoder. Hybrid models combine the characteristics of the above two types, such as RACE [20] which proposed a retrieval-augmented mechanism that can be leveraged in the generation phase. For learning-based and hybrid approaches, the process of embedding may also need some additional parsed structure information such as abstract syntax tree [13]. Additionally, with the development of pre-trained language models in natural language processing [11, 19], more and more pre-trained code change representation methods have emerged [6, 7, 25]. This also significantly improves the quality of embeddings and makes great progress in ACMG task.

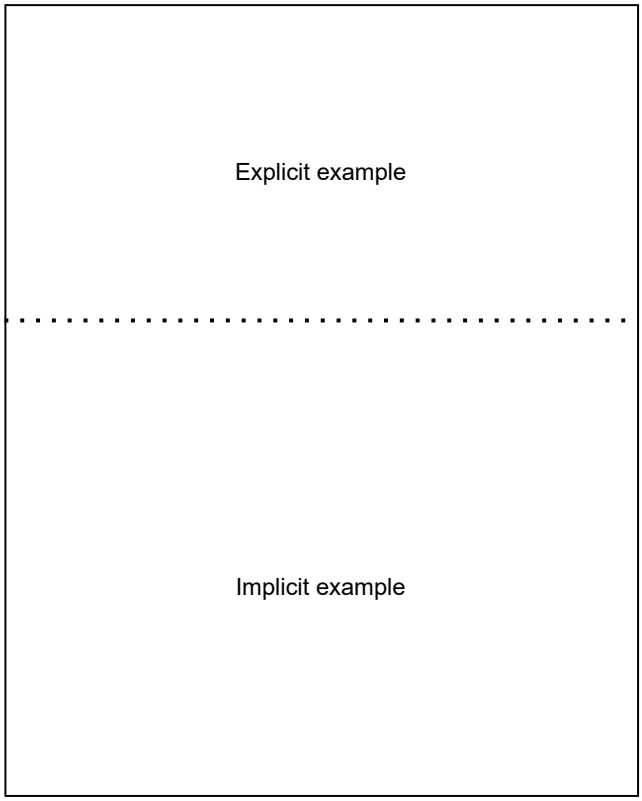

Figure 1: Motivation example.

Although existing approaches show promising performance, they still tend to generate meaningless and irrational commit messages[24]. An obvious restriction is that all these techniques just receive input information directly or indirectly from code itself. While the code, as the object of modification, can usually only reflect the result of the commit rather than the reason. This makes it difficult for existing methods to generate good commit messages that contain both summary (i.e. **What**) and motivation (i.e. **Why**) of the change [23]. In fact, a lot of commits are highly correlated to some issues of bug reports, which usually includes detailed reason of the commit [1, 27]. As shown in Figure 1: On the one hand, some commits may explicitly provide a link to the issue in the commit message; on the other hand, other commits may be mentioned in the discussion of the issue, implying that its motivation is to solve the corresponding problem in the latter. Among the existing research on ACMG tasks, none has focused on this correlation between commit and issue. Therefore, exploring this correlation in depth and leveraging it to improve the performance of ACMG approaches has become the main motivation of this paper.

In this work, we explored the correlation between commit and issue in perspective of data and approach. First, to fill the gap in research community on this topic, we collect a large commit-issue parallel dataset **COMIS** (**com**mit-**is**sue) to understand what kind of rational information issues can provide. During the construction of the dataset, we also think about the question of how to identify fine-grained knowledge in the data. Therefore, the whole dataset consists of two parts. One part is unlabeled and the other is labeled in which each data item is provided with human-annotated rational information in the issue. Further, in perspective of the model, we propose a novel training paradigm ***ExGroFi*** (**E**xtration, **G**rounding, **F**ine-tune) that can improve the performance of pre-trained ACMG models by introducing the correlation between commit and issue into the training phase. The paradigm consists of three pipelined stages. The first stage aims to extract fine-grained knowledge from issues. Then the second stage leverages the extracted knowledge to better ground the code change into semantic space. In the last stage, a fine-tune task is performed to generate commit messages.

To evaluate whether ***ExGroFi*** is effective for ACMG task, we perform an extensive evaluation on the collected **COMIS** dataset. The result of experiments shows that the performance of state-of-the-art ACMG models can be significantly improved by introducing correlation between commit and issue using ***ExGroFi***. The visualized reduction of distance in semantic space between vectors of code change and commit message can further prove the effectiveness of ***ExGroFi***. In addition, considering the incompleteness of automatic metrics, we also performed a human evaluation from four aspects: rationality, comprehensiveness, conciseness and expressiveness. Results show that ***ExGroFi*** can significantly improve the rationality of the generated commit messages while maintaining the level of other indicators.

In summary, the main contributions of this paper are:

- A high-quality commit-issue parallel dataset **COMIS**, which is the first dataset that can be used to explore the correlation between commit and issue.
- A novel paradigm ***ExGroFi*** for commit message generation, which consists of an extraction stage, a grounding stage and a fine-tune stage. The paradigm is designed to introduce the extracted rational information from issues to the training phase of commit message generation models.
- A comprehensive evaluation of ***ExGroFi***, including a comparison study, verification of the performance of each stage and a human evaluation. All results suggest the effectiveness of our paradigm from different perspectives.

The remainder of this paper is organized as follows. Sec. 2 elaborates the background of this work. Sec. 3 introduces the collected commit-issue parallel dataset **COMIS**. Sec. 4 explains in detail the proposed ***ExGroFi*** paradigm. Sec. 5 and Sec. 6 presents our experimental configuration and analyzes the results. Sec. 7 discusses the threat of validity and limitations of our work. Sec. 8 lists some related works. Sec. 9 gives a conclusion of this paper.

2 BACKGROUND

Commits and issues are always highly correlated. Intuitively, we can extract some helpful knowledge from issues to generate better commit messages. However, none of the existing automatic commit message generation approaches leveraged this correlation. In this section, we further analyze the reasons for this situation and lead to the motivation for our work.

2.1 Correlation Between Commit and Issue

Both issues and commits play a crucial role in collaborative software development. An issue refers to a bug report, a feature suggestion, or any problem that needs to be addressed for a project.

Issues can be identified by any user of the software, and they are typically tracked in a project management tool like Jira, Trello, or GitHub Issues. A commit is a specific change made to the source code of the software. Commits typically include a brief message that describes the changes made, along with any relevant metadata such as the author, timestamp, and the specific files and lines of code that were modified. Issues often drive the creation of commits. When developers start working on an issue, they will make one or more commits to the codebase as they implement the necessary changes to resolve the issue. (Many commits are associated with the issue it handles, and the issue is typically marked as "in progress" or "resolved" once the commits have been merged into the main codebase.)

In the scenario of writing a commit message, the developer will briefly describe the content of the modification according to the code change and attach an explanation by referencing an issue. One common way to relate a commit message to an issue is to reference the issue number or link in the message explicitly. For example, the commit message might include a line like "Refactor code related to issue 1234" or "Fixes <link to issue #1234>". In other cases, the developer may also directly extract some sentences from the issue as the reason. Therefore, intuitively, there should be a strong correlation between the text of issues and commit messages. Tian et al. [22] has validated the existence of this correlation from the perspective of semantic similarity. They used BERT to embed text in the semantic space. Moreover, the results show that the original issue (bug report) and associated commit message pairs are much more similar than the randomly selected ones. Consequently, it is easy to imagine that issues will contain much rational information for writing commit messages.

2.2 Lack of data and research

Although the correlation between commits and issues is obvious, there is still no research that exploits this kind of knowledge to solve the ACMG task. This situation is mainly due to the difficulty in obtaining high-quality issue data. As the open-source software (OSS) community continues to thrive, an increasing number of OSS projects are turning to public issue-tracking systems (ITS), such as Jira and the issue-tracking functionality of Github to manage issues. This enables timely feedback on problems in the project but also results in a diverse and inconsistent quality of issue data due to the following barriers. (1) Technical knowledge and expertise. Developers with different technical knowledge and expertise levels may have different abilities to describe complex issues. This can lead to some developers writing more complex and accurate issues, while others may have difficulty communicating the full extent of the problem or suggestion being reported. (2) Motivation and engagement. Developers with different levels of motivation and engagement may have different levels of interest in contributing to the project and writing high-quality issues. This can lead to some developers taking the time to carefully research and report well-written issues, while others may not be as invested in the project and may write lower-quality issues. (3) Communication skills. Developers with different communication skills may have different abilities to clearly and concisely describe an issue. This can lead to some developers writing more organized and easy-to-understand issues, while others may have difficulty expressing themselves in a way that is clear and concise.

Therefore, the current situation is that although a large amount of open-source issue data exists, there is no high-quality and sufficiently large dataset that provides issues, nor is there a sophisticated training paradigm that can effectively leverage the correlation between commit and issue. This leads to the motivation for our work: construct a high-quality commit-issue parallel dataset and explore methods to efficiently use the correlation information for automatic commit message generation.

3 THE COMIS DATASET

To investigate in depth the significance of the commit-issue correlation for the ACMG task, we constructed a commit-issue parallel dataset **COMIS**. This dataset fills a gap in the academic community in this area and may provide a good basis for future research. It provides not only the raw unlabeled corpus, but also labeled data that can guide models to learn from external knowledge. In this section, we first introduce the process of construction of the dataset. Then we explain how we defined the knowledge that should be labeled and the details of annotation. Finally, some useful statistics and data examples are presented.

3.1 Construction

3.1.1 collection. Considering the ease of access and volume of data, we collected data from Github, a web-based platform for version control and collaboration. GitHub not only has a large number of high-quality open source projects, but it also has a comprehensive issue-tracking functionality. We only focused on projects developed in Java because it is one of the most widely used programming languages in the industry and is the most studied in academic research. We first collected data from the top 1k Java repositories according to the number of stars using PyGithub, which is the python package of Github's REST API.

Our purpose is to construct a dataset that provides linked commit-issue pairs in real software projects. However, the original GitHub REST API does not provide an interface that can directly fetch this kind of data. Le et al. [10] proposed a discriminative model which can predict if a link exists between a commit message and a bug report. Regardless, the model is trained on generated commit messages and bug reports, which means it can not ensure the objectivity and truthfulness of data. After an in-depth examination of the GitHub REST API, we found that it provides a class of IssueEvent object that can help us reconstruct the information we need. A Github IssueEvent refers to an activity related to an issue. It occurs when an issue is opened, closed, edited, or referenced. For each project issue, we filter out all the commits that reference it by specifying the type of IssueEvent as "referenced". At the same time, we maintain a mapping from commit to a list of issues, and every time we get a commit that references an issue, we append that issue to the corresponding list in the mapping. In this way, we successfully collected all the primary data we needed, and we kept the following information for further processing:

- **Files**: A list of files that are modified in the commit. For each file, its relative path in the project (contains its filename) and code change (formatted as the output of the "*git diff*" command) is provided.
- **Commit message**: The whole commit message of the commit.
- **Issues**: A list of issues that are referenced by the commit. For each issue, its title (a summary of the issue in a few words) and its body (a detailed description of the issue) is provided.

3.1.2 Text processing. The text data obtained from the GitHub REST API usually contains a lot of noise. This makes it challenging to extract valuable insights from them without proper text-processing techniques. In our dataset, there are three types of text data: commit message, issue title, and issue body. Commit messages and issue titles are relatively easy to process because they are usually short. While for issue bodies, the situation is more complex. As a free-form text, issue bodies can include plain text, markdown, and even images. It should provide all the necessary information to understand and reproduce the issue, including steps to reproduce, expected behavior, actual behavior, error messages, example code snippets, and other relevant information. Therefore, as mentioned in the previous section, the quality of issue bodies also varies. Different projects and issue proponents have different writing habits. In order to obtain a more generalized knowledge from issues, we need to normalize text from the point of semantics. To do this, we replaced URLs and code snippets in issue bodies with special tokens "[URL]" and "[CODE]". Moreover, for issue numbers mentioned in commit messages, we replaced them by the token "[ISSUE_NUMBER]".

3.1.3 Filtering. Not every data item that we obtain can be deposited into the dataset. We also need to perform further filtering on the processed text data. According to Liu et al. [14], there will be some commit messages that are automatically generated by bots or trivial that contain little and redundant information. Following their work, we also removed the data containing these commit messages and kept only items where the text content was all in English. In addition, considering the length limitation of the existing pre-trained models on the input token sequence, we also filtered the data based on the length of tokenized sequences. Data containing fields that exceed the length limit are also excluded.

3.2 Fine-grained Information in Issues

3.2.1 Definition. As mentioned earlier, the writing style and text length of issues proposed by different people vary greatly. This also makes it challenging to explore the correlation between commits and issues. Even if we carefully process the text and eliminate the content that does not contain semantic information as much as possible, it is still difficult to effectively use it to provide valuable information for the models. Therefore, we try to manually annotate the key information in the issue so that this information can help the model to learn more knowledge and better embed code change in semantic space. Observing a large amount of data, we defined two types of fine-grained information that may be useful for pre-trained code change representations.

Issue type. Issues can be categorized into different types based on motivation. We hypothesize that the type of issue can be used as a guide to allow the model to accurately extract fine-grained information from the issue. We summarize three commonly used issue types: - bug report: A bug report is an issue that describes unexpected behavior or an error in the code. It needs to be fixed to ensure the proper functioning of the software. - Feature request: A feature request is a suggestion or proposal for a new feature or functionality that users would like to see in the software. The way users use the software will change because of the commits related to it. - Enhancement: An enhancement is a suggestion to improve an existing feature or functionality of the software. Unlike feature requests, it usually does not result in a change in the way users use the software. Common enhancement includes performance optimization, compatibility optimization, etc.

State information. Issues always contain some descriptive content of the current situation and expected results, and these descriptions are the most critical content to tell developers why the modifications should be done. We define this kind of knowledge as actual / expected state information. By analyzing the actual state and comparing it with the expected state, developers can make better changes to the code. The significance of a commit is to update the current codebase to meet expectations so that a commit message should be highly related to this information. From the perspective of representation, the actual / expected state can also be regarded as the projection of the code before and after the commit in the natural language space. This means that the text containing actual / expected state extracted from issues could be used as a supervised objective for code change representation learning.

3.2.2 Annotation. According to our definition of fine-grained information in issues, we performed an annotation of the collected data. Five well-trained annotators were asked to label the type of issue and to extract actual state expected state information from it. We use label-studio as the annotation platform. The commit message of a commit together with the title and body of all issues related to this commit are shown on the labeling page of each data item. We finally obtained xxx labeled data items.

3.3 Stratification

The whole COMIS dataset has two parts:

- The first part COMIS-base consists of all unlabeled commit-issue data. This part can be regarded as an ACMG dataset, but for each commit, all related issues are attached.
- The second part COMIS-fine consists of all labeled commit-issue data. Each of the data contains not only the related issue list but also the type and the actual / expected state information of each issue.

For COMIS-base, we randomly stratified it into train, valid and test split with a ratio of 8:1:1. Detailed statistics of the whole dataset are given in Table 1.

Table 1: Number of example for each split of COMIS.

Split	base-train	base-valid	base-test	base	fine
#Example	15409	1926	1927	19262	1008

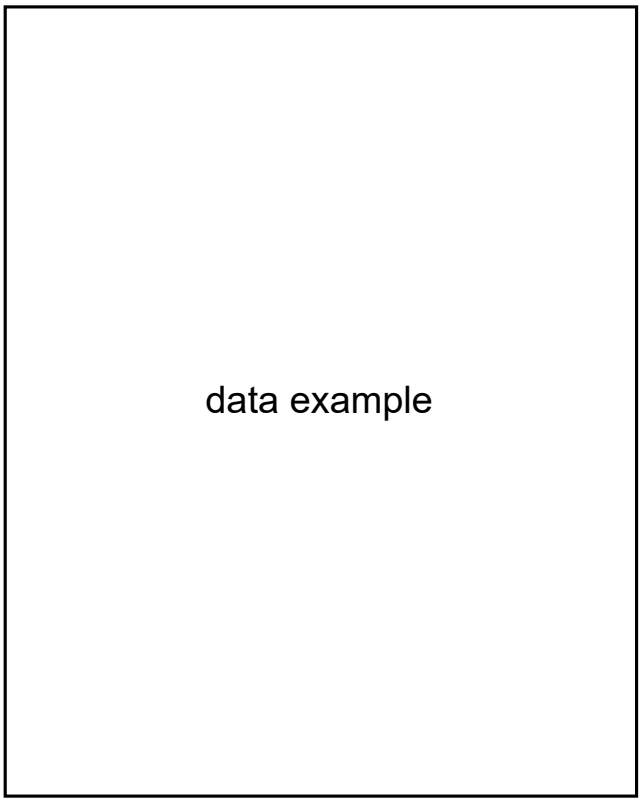

Figure 2: Data example of COMIS

3.4 Example

A data example of COMIS is shown in Figure 2

4 THE *EXGROFI* PARADIGM

In this section, we will introduce the proposed ***ExGroFi*** (Extration, Grounding, Fine-tune) paradigm, which leverages the correlation between commits and issues to generate more reasonable commit messages. The overall paradigm is illustrated in the 3. It is noteworthy that ***ExGroFi*** is not a simple model that takes code changes as input and generates commit messages as output. Instead, it integrates the fine-grained information carried by issues into an pre-trained ACMG model to generate improved commit messages. The entire paradigm consists of three stages: extraction stage, grounding stage and fine-tune stage. During the first stage, we extract fine-grained explanatory information from existing issues. Then, in the grounding stage, we utilize this information to train a model capable of embedding code change more precisely within a semantic space. Finally, in the fine-tune stage, a pre-trained generative model F is used to output commit messages using code diff. F can be any existing learning-based or hybrid ACMG model which uses sequentialized code change as input; however, the difference is that it is initialized using the parameters obtained in the grounding stage, which incorporates the knowledge of the correlation between commits and issues. We then describe each stage of ***ExGroFi*** in detail.

4.1 Extraction Stage

The purpose of this stage is to extract the actual/expected state information, as defined earlier, from a large corpus of issue texts. Through practical experimentation, we discovered that including the issue's type information in the input during extraction enhances the extraction performance. Consequently, we designed a pipeline model structure within this stage: for each issue to be processed, we first employ a classifier to categorize its type, then input both the classification result and the issue text into a sequence labeling model to extract the state information. The specific internal structure of this stage is illustrated in Figure 4.

4.1.1 Issue Type Classification. For a given input issue token list $T^{issue} = \{t_1^{issue}, t_2^{issue}, ..., t_n^{issue}\}$, we employ a pre-trained language model PLM_{cls} to acquire the distributed representation of each token $H^{issue} = \{h_1^{issue}, h_2^{issue}, ..., h_n^{issue}\}$. Subsequently, the representation vector of the first token h_1^{issue} is fed into the classification head MLP_{cls}, a linear layer with a softmax activation function, to obtain a probability distribution across all categories. The category with the highest probability is identified as the type of the input issue. Upon acquiring the type of the issue, we convey this information to the downstream model by concatenating a special token t_{type}^{issue} at the beginning of the input sequence. Specifically, the three types - bug report, feature request, and improvement suggestion - are respectively represented by "[BR]", "[FR]", and "[IS]". By training the downstream model, it will utilize this knowledge and enhance the efficacy of state extraction.

4.1.2 State Information Extraction. We model state information extraction as a sequence tagging (also known as sequence labeling) task. Sequence tagging is commonly employed to extract meaningful substrings from a lengthy character string, such as Named Entity Recognition (NER) task in Natural Language Processing. In the context of state information extraction, assuming that the AS (actual state) and ES (expected state) information is text within the issue, employing a sequence tagging approach for extraction is both reasonable and straightforward to implement. Under this modeling paradigm, the model classifies each input token by assigning a tag . Different tags represent the positions of their corresponding tokens relative to the content to be extracted. We utilize the **BIO** labeling scheme, which works as follows:

- **B** (begin): This tag is assigned to the first token of the content to be extracted, and it is followed by the content type, e.g., B-AS for the beginning of actual state information.
- **I** (inside): This tag is assigned to the tokens within the target content, also followed by the content type, e.g., I-ES for tokens inside expected state information.
- **O** (outside): This tag is assigned to tokens that are not part of any target content.

Similar to the previous step, the state information extraction model comprises a pre-trained language model PLM_{ext} and a linear classification head MLP_{ext}. The token sequence integrated with type information $T^{issue\prime} = \{t_{type}^{issue\prime}, t_1^{issue\prime}, t_2^{issue\prime}, ..., t_n^{issue\prime}\}$ is fed into the model to obtain hidden layer vectors in the semantic space $H^{issue\prime} = \{h_{type}^{issue\prime}, h_1^{issue\prime}, h_2^{issue\prime}, ..., h_n^{issue\prime}\}$. Subsequently, MLP_{ext}

Figure 3: Overview of *ExGroFi*.

performs classification on the hidden vectors at each position, yielding an output sequence composed of tags. Finally, the target information contained within the text can be decoded based on the schema. For each input sequence, the sum of the cross-entropy for every token is considered as the final loss.

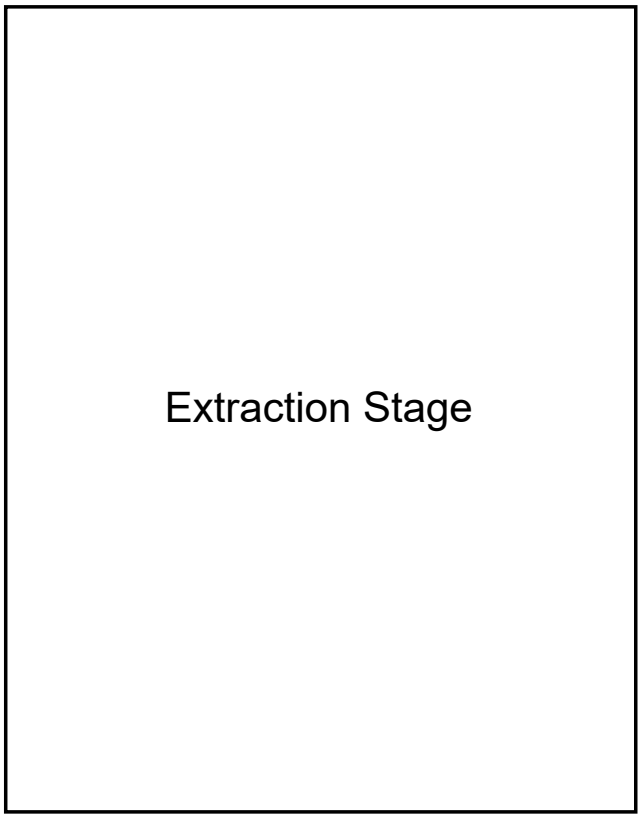

Figure 4: Extraction stage.

4.2 Grounding Stage

In the context of deep learning, grounding refers to connecting abstract representations learned by a neural network to real-world knowledge. In this paper, we draw inspiration from this terminology and refer to the process of enhancing the encoder-decoder model's performance as grounding. During the grounding stage, we aim to embed the code change into a representation vector that more accurately reflects its semantics. According to the definition of state information provided earlier, issues can offer descriptions of the states that the code should exhibit before and after the change. In other words, the state information can be regarded as natural language summary of the code change. Therefore, the state information can serve as a supervisory objective for code change representation learning.

The grounding stage can be initialized by any pre-trained commit message generation model F which takes code diff as input. Given a sequentialized code diff token list $C = \{c_1, c_2, ...c_m\}$, where m is the length of the list, the objective of this stage is to obtain a grounded model that can align C with words possessing the same semantic information. By utilizing a word sequence S concatenated by state information of related issues as the target language, we can model the training process of Fas a translation task. As shown in [Figure], we construct Sin the following way: All actual states and expected states are concatenated respectively, with four special tokens indicating their boundaries. Since the model outputs a probability distribution over the vocab on all translation steps, the summed cross-entropy loss is also used as objective function. After grounding, we acquire a model $F_{grounded}$ which possess the knowledge learned from correlation between commits and issues.

4.3 Fine-tune Stage

The fine-tune stage is relatively simple: Given an input sequence of code diff $C = \{c_1, c_2, ...c_m\}$, $F_{grounded}$ is required to generate commit message in this stage. Note that we use the same structure as F to build the generative model, but the parameters are initialized by $F_{grounded}$.

5 EXPERIMENTAL SETUP

5.1 Research Question

- **RQ1: Overall effectiveness.** To what extent can ***ExGroFi*** enhance the performance of existing ACMG models?
- **RQ2: Extraction performance.** How effective does the first stage extract fine-grained information from issues?
- **RQ3: Contribution of grounding stage.** Does the model really learn useful correlation knowledge through the grounding stage?
- **RQ4: Human evaluation.** How does ***ExGroFi***-enhanced models perform from the perspective of developers?

5.2 Data

Since there was no commit-issue parallel corpus before, we used the dataset COMIS described in Section 3 for all experiments. The annotated issues in COMIS-fine are used to train the extraction stage. COMIS-base is used to train the grounding model and the generation stage.

5.3 Enhanced Models

We used ***ExGroFi*** to enhance six state-of-the-art models as follows.

- **Encoder-only models** only provide contextualized code diff representation by a pre-trained encoder. We consider three encoder-only pre-trained models CodeBERT, CodeBERTa and unixcoder.
- **Encoder-decoder models** possess both pre-trained encoder for representation and decoder for generation. We consider two encoder-decoder models CodeTrans and CodeT5. We experimented with both the SMALL and BASE versions of these two models, respectively.
- **Hybrid models** use both retrieval-based and learning-based techniques. We select RACE, which is a retrieval-augmented neural commit message generation model.

5.4 Implementation

Models. We use the weight downloaded from Hugging Face to initialize our models. URLs for each model card can be found in Appendix. For encoder-only and encoder-decoder models, we directly use the downloaded pre-trained checkpoint. But encoder-only models do not have decoders for generation. So we randomly initialize a Transformer Decoder for encoder-only models. The hyperparamters of these random decoders are set according to the configuration of corresponding encoder model. For RACE, we initialized it with weight of CodeT5-base as reported in its original paper. In addition, for the pre-trained language model in extraction stage, we use CodeBERT.

Tokenizers. As mentioned in Sec. 4.1.1, we add three special tokens "[BR]", "[FR]", and "[IS]" for the tokenizer of extraction stage to indicate the type of issue type. Samely, for the tokenizer of grounding stage, four special tokens "<AS>", "</AS>", "<ES>", "</ES>" are used to point out boundaries of the actual state and expected state.

Environments. The experiments are performed on a NVIDIA DGX Station with Intel Xeon CPU E5-2698 v4 @ 2.2 GHz, running Ubuntu 20.04.4 LTS. The models are trained on one 32G GPU of NVIDIA TESLA V100.

5.5 Automatic Metrics

5.5.1 Commit Message Generation. Numerous automated metrics have been employed for the evaluation of the ACMG task. However, each evaluation metric possesses inherent limitations. Consequently, in order to assess ***ExGroFi*** from diverse perspectives, we have selected the following four distinct automated metrics for our analysis.

- **BLEU** measures the precision of n-grams (sequences of n words) in the generated text compared to the reference texts. It calculates the modified n-gram precision and applies a brevity penalty to avoid favoring shorter sentences.
- **ROUGE-L** is a metric that evaluates the quality of summaries by measuring the longest common subsequence (LCS) between the generated text and the reference texts. It is less sensitive to the exact word order than BLEU and provides a more recall-oriented evaluation.
- **METEOR** calculates the harmonic mean of unigram precision and recall between the generated text and the reference texts, considering exact matches, synonyms, and stemmed forms of words.
- **CIDEr** computes the Term Frequency-Inverse Document Frequency (TF-IDF) weighting for each n-gram to emphasize the importance of informative and rare n-grams. Therefore, it is more effective in capturing the relevance of specific details.

5.5.2 State Information Extraction. Drawing upon the information extraction domain, we opted to use precision, recall, and F1-score to evaluate the performance of extraction stage. During the actual training of the state information extractor, we utilized the COMIS-fine dataset, which was randomly divided into an 8:1:1 ratio for the training, validation, and testing sets. A micro-F1 score which is calculated over the whole test set is also presented. It is essential to note that the information we aim to extract is typically longer strings, so requiring the model's output to be identical to the golden answer might be excessively stringent. When calculating the metrics, we employ a fuzzification strategy to assess whether the extracted content by the model meets the requirements. Specifically, we first compute the Levenshtein distance between the model output string p and the golden string g. Then we calculate a similarity measure using the formula below. If the similarity exceeds a predetermined threshold τ (we set the value as 0.8), we consider the two strings to be matching, indicating that the model has extracted the correct answer.

6 RESULTS AND ANALYSIS

In this section, we present the overall effectiveness of ***ExGroFi***-enhanced models for ACMG (RQ1) in Sec. 6.1, the performance of extraction stage (RQ2) in Sec. 6.2, the contribution of grounding stage (RQ3) in Sec. 6.3 and human evaluation (RQ4) in Sec. 6.4.

Table 2: Overall effectiveness

Model	BLEU			ROUGE-L			METEOR			CIDEr		
	ori.	EGF	imp.	ori.	EGF	imp.	ori.	EGF	imp.	ori.	EGF	imp.
CodeBERT	8.81	9.09	**3%**	15.85	16.05	**1%**	4.75	5.0	**5%**	0.09	0.11	**22%**
CodeBERTa	8.43	8.50	**1%**	14.86	15.66	**5%**	4.92	5.12	**4%**	0.09	0.11	**22%**
unixcoder	9.46	9.54	**1%**	17.04	17.59	**3%**	5.57	6.18	**11%**	0.14	0.16	**14%**
encoder-only avg.	8.9	9.04	**2%**	15.92	16.43	**3%**	5.08	5.43	**7%**	0.11	0.13	**18%**
CodeTrans-small	11.08	12.66	**14%**	18.62	20.84	**12%**	7.98	9.07	**14%**	0.3	0.46	**53%**
CodeTrans-base	19.27	20.37	**6%**	27.34	28.53	**4%**	16.37	18.31	**12%**	1.0	1.08	**8%**
CodeT5-small	11.40	12.59	**10%**	20.70	22.68	**10%**	8.86	10.02	**13%**	0.36	0.49	**37%**
CodeT5-base	19.96	20.94	**5%**	30.07	31.46	**5%**	16.99	18.39	**8%**	1.04	1.11	**7%**
RACE	21.06	22.47	**7%**	30.17	36.09	**20%**	17.23	18.64	**8%**	1.14	1.51	**32%**
encoder-decoder + hybrid avg.	16.55	17.81	**8%**	25.38	27.92	**10%**	13.486	14.886	**10%**	0.768	0.93	**21%**

6.1 RQ1: Overall Effectiveness

The overall effectiveness of ***ExGroFi*** is presented in Table 2. It can be observed that the generative performance of each model is improved after being trained using the ***ExGroFi*** paradigm. Among them, the overall generative performance of encoder-only models is relatively inferior, which can be attributed to their decoders being initialized randomly. In contrast, the generative performance of pre-trained encoder-decoder models is considerably superior. Regardless of whether the models are initialized entirely with pre-trained parameters, those trained using the ***ExGroFi*** paradigm consistently outperform models that solely rely on fine-tuning. This demonstrates the effectiveness of ***ExGroFi*** in the ACMG task.

Specifically, for encoder-only models, ***ExGroFi*** can increase the average scores of BLEU, ROUGE, METEOR, and CIDER by 2%, 3%, 7%, and 18%, respectively. For encoder-decoder models, these four metrics can be improved by 8%, 10%, 10%, and 21%, respectively. The most notable improvement is observed in the CIDEr metric, indicating that models trained with ***ExGroFi*** are more likely to generate outputs that capture specific details or content deemed significant by humans. This is crucial in the context of the ACMG task, as different commit messages often contain vocabulary unique to specific code repositories, such as particular class names and variable names. Moreover, this result can be interpreted as ***ExGroFi*** successfully integrating the relationship information between commits and issues into the model, especially the state information contained in issues.

In the accompanying figure, we also present several comparative examples of commit messages generated before and after using ***ExGroFi***, as well as the corresponding issue information. It can be seen that the superiority of ***ExGroFi***-generated results becomes more apparent when the code diff is relatively complex. Previous models, limited by input length, could not fully cover all information in the code diff. However, with the aid of ***ExGroFi***, the model can directly obtain key natural language information, significantly reducing the loss of information when representing code diff and consequently producing more accurate and well-founded commit messages.

6.2 RQ2: Extraction Performance

In the ***ExGroFi*** paradigm, a learning-based model is employed to extract state information, necessitating an evaluation of its extraction performance. Drawing upon the information extraction domain, we opted to use precision, recall, and F1-score as the evaluation metrics. During the actual training of the state information extractor, we utilized the COMIS-fine dataset, which was randomly divided into an 8:1:1 ratio for the training, validation, and testing sets. In the Table 3, we present the precision (P), recall (R), and F1-score for the two categories of state information, as well as the micro-F1 for the entire test set. The middle column and the leftmost column of the table represent the extraction performance without and with the utilization of category information, respectively. It can be observed that the model's extraction performance exhibits a significant improvement when incorporating category information. Furthermore, the model demonstrates a better extraction capability for the expected state (F1-score 61.68%) compared to the actual state (F1-score 81.08%). This is primarily due to the fact that the expected state is often proposed by more experienced developers or users, resulting in higher data consistency than the actual state, which consequently makes it easier for the model to extract. However, the overall micro-F1 score reaches 71.56%. Based on the experience in the information extraction domain, this extraction result is sufficient to meet the demands of downstream applications. Figure 5 provides several examples of state information extracted from the issues in COMIS-base.

Table 3: Performance of state information extraction model

Approach		codebert	+ issue type
Actual State	**P**	56.90	**58.93**
	R	63.46	**64.71**
	F1	60.00	**61.68**
Expected State	**P**	80.77	**84.91**
	R	72.41	**77.59**
	F1	76.36	**81.08**
Micro-F1		68.18	**71.56**

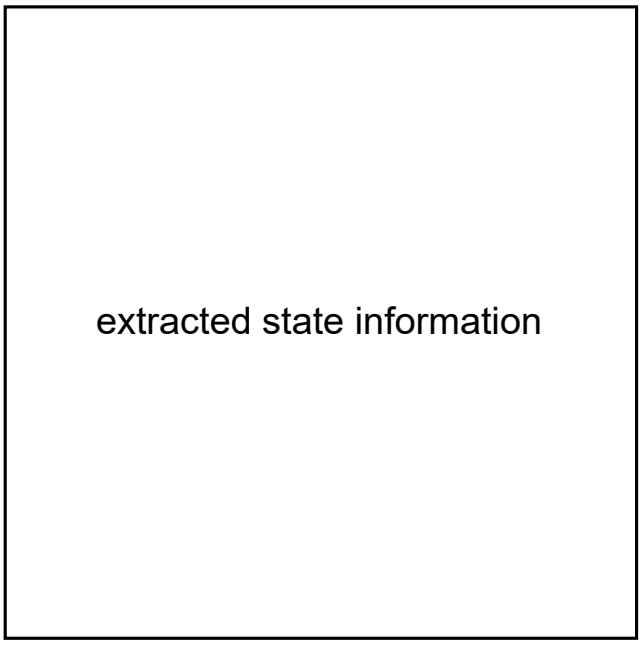

Figure 5: Example of extracted state information.

6.3 RQ3: Contribution of Grounding Stage

To validate whether the grounding stage of ***ExGroFi*** can integrate knowledge related to commits from issues into models, we conduct a comparative experiment. Firstly, we extract the encoders from the original CodeT5-base and the ***ExGroFi***-enhanced CodeT5-base. Then, we utilize these two encoders to individually embed the code changes and commit messages of each data example in the test set of the COMIS-base dataset, resulting in two pairs of vectors (v_o^{cc}, v_o^{msg}), (v_f^{cc}, v_f^{msg}). We also standardize these vector values to eliminate the influence of dimension. Then we compute the Euclidean distance between the vectors in each pair, representing their distance in the semantic space. A smaller distance implies a higher semantic similarity between the obtained code change representations and commit message representations. Figure 6 presents the distribution for pairs embedded by encoders before and after grounding. The results show that the representations provided by the fused model are more similar. The Mann-Whitny-Wilcoxon test (p-value: 1.9e-55) further validates the significance of the difference before and after grounding. This indicates that ***ExGroFi*** enables the model to more accurately align code changes within the semantic space, thereby generating more reasonable commit messages.

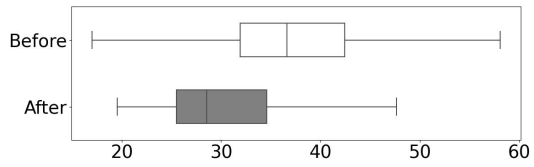

Figure 6: Euclidien distance before and after grounding.

6.4 RQ4: Human Evaluation

While automatic metrics can provide valuable insights, the lack of semantic understanding still makes them not sufficient for a comprehensive evaluation. Therefore, following [], we conduct a human evaluation to further study the quality of commit messages generated by ***ExGroFi***-enhanced models. We randomly select 50 data samples from the test set of COMIS-base and inference them by CodeT5-base, CodeTrans-base and RACE. We also design a questionnaire for annotation. For each sampled data, the questionnaire includes the code change, the ground truth commit message, the commit message generated by the original model and the commit message generated by the ***ExGroFi***-enhanced model. Four experienced annotators are invited to evaluate the result from four aspects for a generated commit message:

- **Rationality**: Whether it provides a logical explanation for the changes, addressing the "why" behind the commit.
- **Comprehensiveness**: Whether it covers all important details and provides a complete picture of the modifications made, addressing the "what" behind the commit.
- **Conciseness**: Whether it conveys information succinctly, ensuring readability and quick comprehension.
- **Expressiveness**: Whether its content is grammatically correct and fluent.

For each aspect, the annotators are required to indicate whether the original or the enhanced model's generated result is better. If the enhanced model is better, they assign "Win"; otherwise, assign "Lose". If they think the difference between the two is not obvious, they should assign a "Tie" label. To mitigate bias, each annotator fills in the questionnaire independently and the agreement among them is measured by Fleiss' kappa.

Table 4: Result of human evaluation.

Indicator	Win	Lose	Tie	Kappa
Rationality	**35 (70%)**	10 (20%)	5 (10%)	0.70
Comprehensiveness	**25 (50%)**	14 (28%)	11 (22%)	0.67
Conciseness	**21 (42%)**	20 (40%)	9 (18%)	0.65
Expressiveness	17 (34%)	17 (34%)	16 (32%)	0.66

Table 4 reports the results of human evaluation. The values of kappa are all above 0.6, indicating substantial agreement among the four annotators. According to the results, we notice that ***ExGroFi*** significantly improves upon rationality while achieves comparable performance on comprehensiveness, conciseness and expressiveness, which substantiates that our proposed paradigm indeed enables models to learn rational information from the correlation between commits and issues.

7 DISCUSSION

We discuss the threats to validity and enumerate a few limitations of our study.

7.1 Threats to Validity

The internal threat to validity lies in the implementation of approaches and the setting of hyperparameters. To reduce this threat, we directly reuse the open-source implementation of RACE and public checkpoints of all pre-trained models on Huggingface. To ensure the optimal hyperparameters for each model, we conducted

numerous iterative experiments and tested with various decoding parameter configurations to generate the best possible results.

The external threat to validity lies in the source of data that we used to construct dataset. To mitigate this threat, we employed the official GitHub API to collect data from the top 1,000 repositories ranked by the number of stars. Furthermore, referencing previous research, we conducted extensive work in cleaning, filtering, and processing the data to ensure the highest possible quality.

The threat to construct validity arises from the evaluation metrics employed. To mitigate this threat, we utilize four metrics that have been extensively employed in previous studies on commit message generation. Moreover, we conduct a human evaluation to assess the effectiveness from the developers' perspective. By rigorously adhering to the methodology of prior research and engaging experienced developers, we aim to minimize potential threats in human evaluation, such as a limited number of participants.

7.2 Limitations

This section discusses the limitations in our work. Firstly, the amount of annotated data utilized for training the extraction module is relatively limited. Although the current volume enables the module to possess a satisfactory extraction capability, additional data would undoubtedly improve the extraction performance. Secondly, due to the pipeline structure within ***ExGroFi***, error propagation exists between different modules. The quality of issue data and the performance of upstream modules both influence the final generation outcome. Lastly, the results generated by models enhanced using the ***ExGroFi*** paradigm contain more rational information; however, this occasionally renders commit messages less concise.

8 RELATED WORK

8.1 Automatic commit message generation

8.2 Code change representation

8.3 Pre-trained language model

9 CONCLUTION AND FUTURE WORK

In this work, we explore the correlation between commits and issues, addressing a previously unexplored aspect of automatic commit message generation. We collect a large commit-issue parallel dataset **COMIS**, which allows for a deeper understanding of the rational information provided by issues. A novel paradigm ***ExGroFi*** is also proposed to extract and incorporate fine-grained knowledge from issues into the commit message generation process. Extensive evaluation demonstrates the effectiveness of the ***ExGroFi*** paradigm in significantly improving the rationality of generated commit messages while maintaining other performance indicators.

This paper can serve as a foundation for future research in the area of commit message generation, particularly in exploring additional methods of extracting and incorporating relevant information from issues and other contextual sources. For instance, researchers can explore the pre-training of large-scale language models based on such commit-issue correlations, investigate more efficient methods for incorporating external knowledge into the commit message generation process, or transfer principles of ***ExGroFi*** to other tasks in the field of automatic software engineering.

REFERENCES

[1] John Anvik, Lyndon Hiew, and Gail C. Murphy. 2006. Who Should Fix This Bug?. In *Proceedings of the 28th International Conference on Software Engineering* (Shanghai, China) *(ICSE '06)*. Association for Computing Machinery, New York, NY, USA, 361–370. https://doi.org/10.1145/1134285.1134336

[2] Raymond P.L. Buse and Westley R. Weimer. 2010. Automatically Documenting Program Changes. In *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering* (Antwerp, Belgium) *(ASE '10)*. Association for Computing Machinery, New York, NY, USA, 33–42. https://doi.org/10.1145/1858996.1859005

[3] 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*. 275–284. https://doi.org/10.1109/SCAM.2014.14

[4] Jinhao Dong, Yiling Lou, Qihao Zhu, Zeyu Sun, Zhilin Li, Wenjie Zhang, and Dan Hao. 2022. FIRA: <U>Fi</U>Ne-Grained G<U>Ra</U>Ph-Based Code Change Representation for Automated Commit Message Generation. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 970–981. https://doi.org/10.1145/3510003.3510069

[5] 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)*. 422–431. https://doi.org/10.1109/ICSE.2013.6606588

[6] Ahmed Elnaggar, Wei Ding, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Silvia Severini, Florian Matthes, and Burkhard Rost. 2021. CodeTrans: Towards Cracking the Language of Silicon's Code Through Self-Supervised Deep Learning and High Performance Computing. *arXiv preprint arXiv:2104.02443* (2021).

[7] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[8] 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* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 518–529. https://doi.org/10.1145/3377811.3380361

[9] 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)*. 135–146. https://doi.org/10.1109/ASE.2017.8115626

[10] Tien-Duy B Le, Mario Linares-Vásquez, David Lo, and Denys Poshyvanyk. 2015. Rclinker: Automated linking of issue reports and commits leveraging rich contextual information. In *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE, 36–47.

[11] Junyi Li, Tianyi Tang, Wayne Xin Zhao, and Ji-Rong Wen. 2021. Pretrained language models for text generation: A survey. *arXiv preprint arXiv:2105.10311* (2021).

[12] 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)*. 299–309. https://doi.org/10.1109/MSR.2019.00056

[13] Shangqing Liu, Cuiyun Gao, Sen Chen, Lun Yiu Nie, and Yang Liu. 2022. ATOM: Commit Message Generation Based on Abstract Syntax Tree and Hybrid Ranking. *IEEE Transactions on Software Engineering* 48, 5 (2022), 1800–1817. https://doi.org/10.1109/TSE.2020.3038681

[14] 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* (Montpellier, France) *(ASE '18)*. Association for Computing Machinery, New York, NY, USA, 373–384. https://doi.org/10.1145/3238147.3238190

[15] Pablo Loyola, Edison Marrese-Taylor, and Yutaka Matsuo. 2017. A Neural Architecture for Generating Natural Language Descriptions from Source Code Changes. *CoRR* abs/1704.04856 (2017). arXiv:1704.04856 http://arxiv.org/abs/1704.04856

[16] Walid Maalej and Hans-Jörg Happel. 2010. Can development work describe itself?. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. 191–200. https://doi.org/10.1109/MSR.2010.5463344

[17] Mockus and Votta. 2000. Identifying reasons for software changes using historic databases. In *Proceedings 2000 International Conference on Software Maintenance*. 120–130. https://doi.org/10.1109/ICSM.2000.883028

[18] Lun Yiu Nie, Cuiyun Gao, Zhicong Zhong, Wai Lam, Yang Liu, and Zenglin Xu. 2021. CoreGen: Contextualized Code Representation Learning for Commit Message Generation. *Neurocomputing* 459 (Oct. 2021), 97–107. https://doi.org/

10.1016/j.neucom.2021.05.039

[19] Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences* 63, 10 (2020), 1872–1897.

[20] Ensheng Shi, Yanlin Wang, Wei Tao, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2022. RACE: Retrieval-augmented Commit Message Generation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 5520–5530. https://aclanthology.org/2022.emnlp-main.372

[21] 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)*. 126–136. https://doi.org/10.1109/ICSME52107.2021.00018

[22] Haoye Tian, Xunzhu Tang, Andrew Habib, Shangwen Wang, Kui Liu, Xin Xia, Jacques Klein, and Tegawendé F Bissyandé. 2022. Is this Change the Answer to that Problem? Correlating Descriptions of Bug and Code Changes for Evaluating Patch Correctness. *arXiv preprint arXiv:2208.04125* (2022).

[23] Yingchen Tian, Yuxia Zhang, Klaas-Jan Stol, Lin Jiang, and Hui Liu. 2022. What Makes a Good Commit Message? *arXiv preprint arXiv:2202.02974* (2022).

[24] Haoye Wang, Xin Xia, David Lo, Qiang He, Xinyu Wang, and John Grundy. 2021. Context-Aware Retrieval-Based Deep Commit Message Generation. *ACM Trans. Softw. Eng. Methodol.* 30, 4, Article 56 (jul 2021), 30 pages. https://doi.org/10.1145/3464689

[25] 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).

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

[27] Jiaxin Zhu, Minghui Zhou, and Audris Mockus. 2016. Effectiveness of Code Contribution: From Patch-Based to Pull-Request-Based Tools *(FSE 2016)*. Association for Computing Machinery, New York, NY, USA, 12 pages. https://doi.org/10.1145/2950290.2950364