Delving into Commit-Issue Correlation to Enhance Commit Message Generation Models

1st Liran Wang
Beihang University
Beijing, China
wanglr@buaa.edu.cn

2nd Xunzhu Tang
University of Luxembourg
Luxembourg City, Luxembourg
xunzhu.tang@uni.lu

3rd Yichen He
Beihang University
Beijing, China
hyc2026@buaa.edu.cn

4th Changyu Ren
Beihang University
Beijing, China
cyren@buaa.edu.cn

5th Shuhua Shi
Beihang University
Beijing, China
shishuhua@buaa.edu.cn

6th Chaoran Yan
Beihang University
Beijing, China
ycr2345@buaa.edu.cn

7th Zhoujun Li*
Beihang University
Beijing, China
lizj@buaa.edu.cn

***Abstract*—Commit message generation (CMG) is a challenging task in automated software engineering that aims to generate natural language descriptions of code changes for commits. Previous methods all start from the modified code snippets, outputting commit messages through template-based, retrieval-based, or learning-based models. While these methods can summarize what is modified from the perspective of code, they struggle to provide reasons for the commit. The correlation between commits and issues that could be a critical factor for generating rational commit messages is still unexplored.**

In this work, we delve into the correlation between commits and issues from the perspective of dataset and methodology. We construct the first dataset anchored on combining correlated commits and issues. The dataset consists of an unlabeled commit-issue parallel part and a labeled part in which each example is provided with human-annotated rational information in the issue. Furthermore, we propose *ExGroFi* (Extraction, Grounding, Fine-tuning), a novel paradigm that can introduce the correlation between commits and issues into the training phase of models. To evaluate whether it is effective, we perform comprehensive experiments with various state-of-the-art CMG models. The results show that compared with the original models, the performance of *ExGroFi*-enhanced models is significantly improved.

***Index Terms*—Commit Message Generation, Dataset Construction, Code Representation Learning**

I. Introduction

Commit messages are in the format of natural language for summarizing and explaining the intention of commits during the maintenance of software projects [1], [2]. Thus, a well-written commit message can help code reviewers quickly understand what the commit is and why the commit is proposed without any detailed look into the complex codes [3], [4].

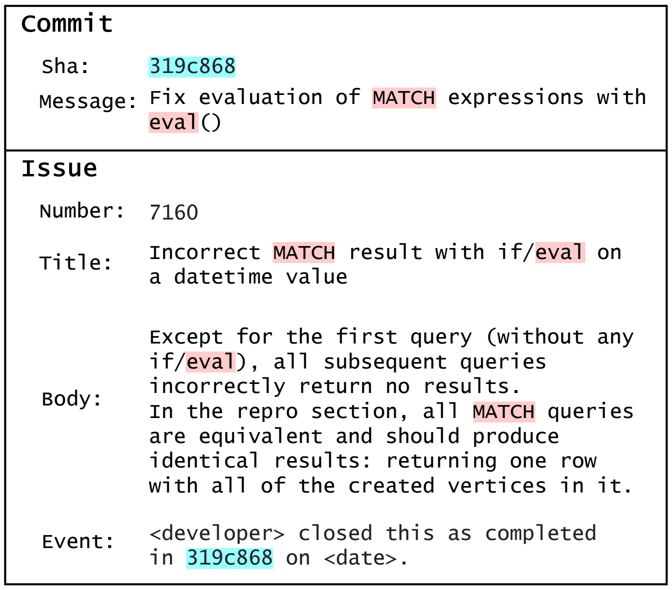
However, commits are often complex, making it difficult and error-prone to summarize with a concise commit message in manual ways [4], [5], [6], [7]. According to Loyola et al. [6], 16% messages in a widely used dataset CommitGen [8] are noisy, which means they contain little information about the commit. In addition, in the situation of increasingly fast-paced modern software development, manually writing high-quality commit messages will also be time-consuming [9].

To simplify the process of writing commit messages, researchers have proposed various techniques for generating commit messages automatically. Since the original input and target output of commit message generation (CMG) are in different modalities, code and natural language, most CMG approaches formulate the CMG pipeline as a translation task.

Mainstream generation methods can be categorized into template-based [2], [10], [11], retrieval-based [12], [13], learning-based [8], [6], [14], [15], [16], [17] and hybrid models [18], [19], [20]. Template-based techniques start from parsing the modification of source code and then generate commit messages with pre-defined rules. Retrieval-based models calculate a similarity score between the requested input and other code changes in the training set. Then, they 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 the encoder. Hybrid models combine the characteristics of learning-based and retrieval-based methods, such as RACE [19], 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 [18]. Additionally, with the development of pre-trained language models in natural language processing [21], [22], more and more pre-trained code change representation methods have emerged [23], [24], [25]. This also significantly improves the quality of embedding and makes excellent progress in CMG tasks.

Although existing approaches show promising performance, they still tend to generate meaningless and irrational commit messages [20]. An apparent restriction is that all these techniques only receive input information directly or indirectly

*Corresponding author


Commit

Sha: 319c868

Message: Fix evaluation of MATCH expressions with eval()

Issue

Number: 7160

Title: Incorrect MATCH result with if/eval on a datetime value

Body: Except for the first query (without any if/eval), all subsequent queries incorrectly return no results.
In the repro section, all MATCH queries are equivalent and should produce identical results: returning one row with all of the created vertices in it.

Event: <developer> closed this as completed in 319c868 on <date>.

Fig. 1: Correlation between commit and issue.

from the 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 [4]. In fact, a lot of commit messages are highly related to the content of some issues or bug reports, which usually describes the detailed reason for the commit [26], [27]. As shown in Figure 1: By reading the given issue, a developer can understand that there is a bug to be fixed because of "the incorrect result of MATCH with eval". The title of the issue summarizes the problem in a few words, and the body of the issue elaborates on the problem in detail by describing actual behaviors and expected behaviors. In the commit message, the keywords to the issue ("MATCH" and "eval") reemerge. Upon resolution of the issue, an event established a correlation between the commit and the issue. None of the existing research on CMG tasks has focused on this correlation between commit and issue. Therefore, exploring this correlation in-depth and leveraging it to improve the performance of CMG approaches has become the primary motivation of this paper.

In this work, we explored the correlation between commits and issues from the perspective of data and approach. First, to fill the gap in the research community on this topic, we collect a large commit-issue parallel dataset to understand what kind of rational information issues can provide. While constructing the dataset, we also consider identifying 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 example is provided with human-annotated rational information in the issue. Further, from the perspective of approach, we propose a novel training paradigm *ExGroFi* (**E**xtration, **G**rounding, **F**ine-tuning) that can improve the performance of pre-trained CMG 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 represent the code change into semantic space. In the last stage, a fine-tuning task is performed to generate commit messages.

To evaluate whether *ExGroFi* is effective for CMG task, we perform an extensive evaluation with the collected dataset. The overall experiment result shows that the performance of state-of-the-art CMG models can be significantly improved by introducing the correlation between commit and issue using *ExGroFi*. Further, the result of two fine-grained experiments can prove the effectiveness of each stage of *ExGroFi*. In addition, considering the incompleteness of automatic metrics, we also perform a human evaluation from four aspects: rationality, comprehensiveness, conciseness, and expressiveness. Results show that *ExGroFi* can 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, 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-tuning stage. The paradigm introduces the extracted rational information from issues to the training phase of commit message generation models.
- A comprehensive evaluation of *ExGroFi*, including a comparison study, a 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. II elaborates on the background of this work. Sec. III introduces the collected commit-issue parallel dataset. Sec. IV explains in detail the proposed *ExGroFi* paradigm. Sec. V and Sec. VI present our experimental configuration and analyze the results. Sec. VII discusses the threat of validity and limitations of our work. Sec. VIII lists some related works. Sec. IX gives a conclusion to this paper.

II. BACKGROUND

Issues are a great way to keep track of what needs to be done and what has been done in a project [28]. It can always guide developers in writing concise and descriptive commit messages. Therefore, commits and issues are always highly correlated [29], [30], [31]. 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.

A. Correlation between Commits and Issues

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. It can be identified by any developer and user of the software, and is typically tracked in a management tool like Jira [32], Trello [33], or GitHub [34]. A commit is a specific change made to the source code. It includes 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 [4]. When developers start working on an issue, they will make one or more commits to the codebase to resolve the issue.

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 [35], [36]. For example, the commit message might include a line like "Refactor code related to issue 1234" or "Fixes issue #1234". In other cases, the developer may also directly extract some sentences from the issue as the reason [37], [38]. Therefore, there should be a strong correlation between the text of issues and commit messages. Tian et al. [29] has validated the existence of this correlation from the perspective of semantic similarity. They used BERT [39] 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.

B. 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 CMG 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 [27], [40] , an increasing number of OSS projects are turning to public issue-tracking systems (ITS), such as Jira [32] and the issue-tracking functionality of Github [41] 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. Developers with different expertise levels may have different abilities to describe complex issues [42], [43]. 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. Developers with different motivations may have different levels of interest in contributing to the project and writing high-quality issues [44]. This can lead to some developers 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 [45]. 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 commits with related issues, nor is there a sophisticated 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 commit-issue correlation for commit message generation.

III. THE DATASET

To investigate the significance of the commit-issue correlation for the CMG task, we construct a commit-issue parallel dataset. This dataset fills a gap in the academic community in this area and may provide a reasonable basis for future research. It provides both raw unlabeled corpus and labeled data that can guide models to learn from external knowledge. In this section, we first introduce the construction process of the dataset. Then we explain how we define the information that should be labeled and the annotation details. Finally, some useful statistics and an example are presented.

A. Construction

1) Collection: Considering the ease of access and volume of data, we collect data from Github [34], a web-based platform for version control and collaboration. GitHub not only has a large number of high-quality open-source projects, but also has a comprehensive issue-tracking functionality [41]. We only focus 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 [9]. We retrieve data from the top 1,000 Java repositories according to the number of stars using PyGithub [46], the Python package of Github REST API [47].

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. [48] proposed a discriminative model to predict if a link exists between a commit message and a bug report. However, this model was 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 find that it provides a class of IssueEvent object that can help us reconstruct the needed information. 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 collect all the primary data we needed, and we keep the following information for further processing:

- **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) are provided.
- **Files**: A list of files that are modified in the commit. For each file, we keep its relative path in the project (contains its filename) and code change (formatted as the output of the *git diff* command).

2) Text processing: The text data obtained from the GitHub REST API usually contains a lot of noise [4], [9], [12]. 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 free-form content, issue bodies can include plain text, markdown, and 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 [28]. In addition, as mentioned in Sec. II-B, different proposers have different writing habits. Therefore, the quality of issue bodies varies. In order to obtain a more generalized knowledge from issues, we need to normalize their content. To do this, we respectively replace URLs and code snippets in issue bodies with special tokens "[URL]" and "[CODE]". Moreover, for issue numbers mentioned in commit messages, we replaced them with the token "[ISSUE_NUMBER]".

3) Filtering: Not every example that we obtain can be deposited into the dataset. We also perform further filtering on the processed text data. According to Liu et al. [12], 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 remove the data containing these commit messages and only keep English data. 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 (1024 tokens) are also excluded.

B. Annotation for Issues

1) Definition: As mentioned in Sec. II-B, 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 directly use it to provide valuable information for models. Therefore, we try to manually annotate the fine-grained information in issues such that it can help models to learn more knowledge and better embed code change in semantic space. Observing a large amount of data, we define two types of fine-grained information as follows.

a) Issue type: Issues can be categorized into different types based on motivation [49]. GitHub issues themselves do not have predefined types or topics. However, project maintainers can use a flexible labeling system to categorize issues based on their nature, priority, or other relevant criteria. By applying labels, maintainers can create a more organized issue-tracking system that is easier to navigate and manage. According to labels, 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.

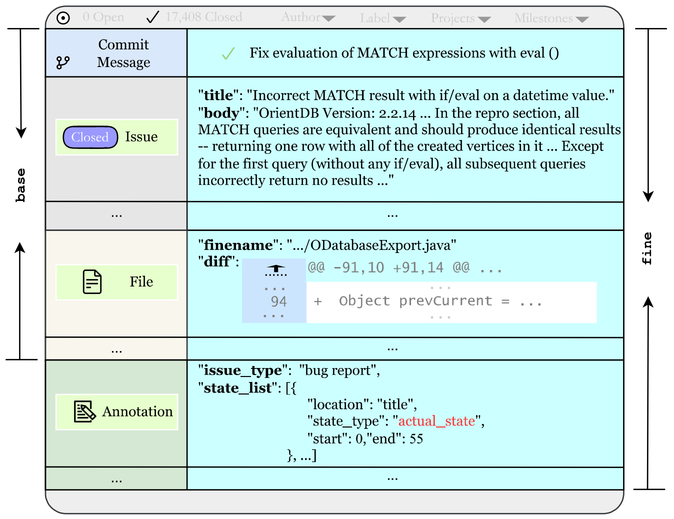
b) State information: Issues always contain some descriptive content of the current situation and expected results [28], and these descriptions are the most critical content to tell developers why the modifications should be done. We define this kind of knowledge as actual/expected state information. By analyzing the actual state and comparing it with the expected state, developers can make better changes to the code. Since the objective of a commit is to update the current codebase to meet expectations, its commit message should be highly related to the state information we defined. From the perspective of representation, the actual/expected state can also be regarded as the projection of the code before/after the commit in semantic space. This means that the text containing actual/expected states extracted from issues could be used as a supervised objective for code change representation learning.

2) Annotation: According to our definition of fine-grained information in issues, we perform an annotation of the collected data. We use Label Studio [50] 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 example. Five well-trained annotators were asked to label the type of each issue and to extract actual state expected state information from it. We finally obtain 960 labeled examples.

C. Stratification

The whole dataset has two parts:

- The base part consists of all unlabeled commit-issue data. This part can be regarded as a CMG dataset, but for each commit, all related issues are attached.
- The fine part consists of all labelled commit-issue data. Each of the data contains not only the related issue list



Fix evaluation of MATCH expressions with eval ()

"**title**": "Incorrect MATCH result with if/eval on a datetime value."
"**body**": "OrientDB Version: 2.2.14 ... In the repro section, all MATCH queries are equivalent and should produce identical results -- returning one row with all of the created vertices in it ... Except for the first query (without any if/eval), all subsequent queries incorrectly return no results ..."

"**finename**": ".../ODatabaseExport.java"
"**diff**":

```
@@ -91,10 +91,14 @@ ...
...
94  +  Object prevCurrent = ...
...
```

"**issue_type**": "bug report",
"**state_list**": [{
"location": "title",
"state_type": "actual_state",
"start": 0,"end": 55
}, ...]

Fig. 2: An example of the dataset

but also the type and the actual/expected state information of each issue.

For the base part, we randomly stratify it into training, validation and test split with a ratio of 8:1:1. The base part contains 19262 data entries in total, and the fine part comprises 960 data entries.

D. Example

An example of the dataset is shown in Figure 2. For each commit data in the base part of the dataset, we provide its commit message, a list of issues related to the commit, and a list of all files that have been modified in the commit. In the issue list, the title and body of each issue are given as strings. In the file list, the filename and code change for each file are also provided. For the fine part, in addition to containing all the information in the base part, it also includes annotation for each issue in the issue list. Specifically, we provide the type and the state information for each issue. The latter is represented by four key-value pairs: "location" indicates whether the information resides in the title or body; "state_type" signifies whether the information is an actual state or an expected state; "start" and "end" respectively denote the offset of the starting and ending characters within the string.

IV. THE *ExGroFi* PARADIGM

In this section, we will introduce the proposed *ExGroFi* (Extraction, Grounding, Fine-tuning) paradigm, which exploits the correlation between commits and issues to generate more reasonable commit messages. The overview of *ExGroFi* is illustrated in Figure 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 is a pipeline that integrates the fine-grained information carried by issues into a pre-trained CMG model to generate improved commit messages. The entire paradigm consists of three stages: extraction stage, grounding stage, and fine-tuning stage. During the first stage, we extract state information (defined in Sec. III-B1b) from existing issues. Then, in the grounding stage, we utilize the extracted information as target output and employ a sequence-to-sequence task to train a pre-trained CMG model F to obtain a refined model $F_{grounded}$. Finally, in the fine-tuning stage, $F_{grounded}$ is trained to generate commit messages using code change. F can be any existing learning-based or hybrid CMG model which uses sequentialized code change as input. After being trained with *ExGroFi*, F is transformed into $F_{grounded}$, which incorporates commit-issue correlation and possesses the ability to generate more rational commit messages. We then describe each stage of *ExGroFi* in detail.

A. Extraction Stage

The purpose of this stage is to extract state information (defined in III-B1b) from a large corpus of issues. Through practical experimentation, we find that including the type of issue as a feature in the input enhances the extraction performance. Consequently, we design a pipeline 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 [51] to extract the state information. The internal structure of this stage is illustrated in Figure 4.

1) Issue Type Classification: For a given issue token list $T = \{t_1, t_2, ..., t_n\}$ (n is the length of the sequence), we employ a pre-trained language model PLM_{cls} to acquire the distributed representation of each token $H = \{h_1, h_2, ..., h_n\}$. Subsequently, the representation vector of the first token h_1 is fed into the classification head MLP_{cls} [52], 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 input issue. Upon acquiring the type of the issue, we convey this information to the downstream model by concatenating a special token t_{type} at the beginning of the input sequence. Specifically, the three types (bug report, feature request, and enhancement) are respectively represented by "[BR]", "[FR]", and "[EN]".

2) State Information Extraction: We model state information extraction as a sequence labeling (also known as sequence tagging) task [51]. Sequence labeling is commonly employed to extract meaningful substrings from a lengthy character string, such as Named Entity Recognition (NER) task in natural language processing [53]. 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 labeling approach for extraction is both reasonable and straightforward to implement. Under this modeling strategy, 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 [54], 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.

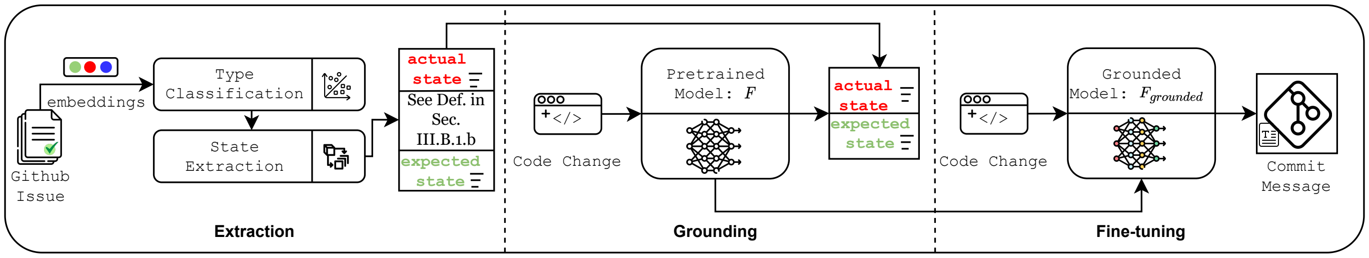


Fig. 3: Overview of *ExGroFi*.

- **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 targeted 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 $T^{'} = \{t^{'}_{type}, t^{'}_1, t^{'}_2, ..., t^{'}_n\}$ is fed into the model to obtain hidden layer vectors in the semantic space $H^{'} = \{h^{'}_{type}, h^{'}_1, h^{'}_2, ..., h^{'}_n\}$. Subsequently, MLP_{ext} 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.

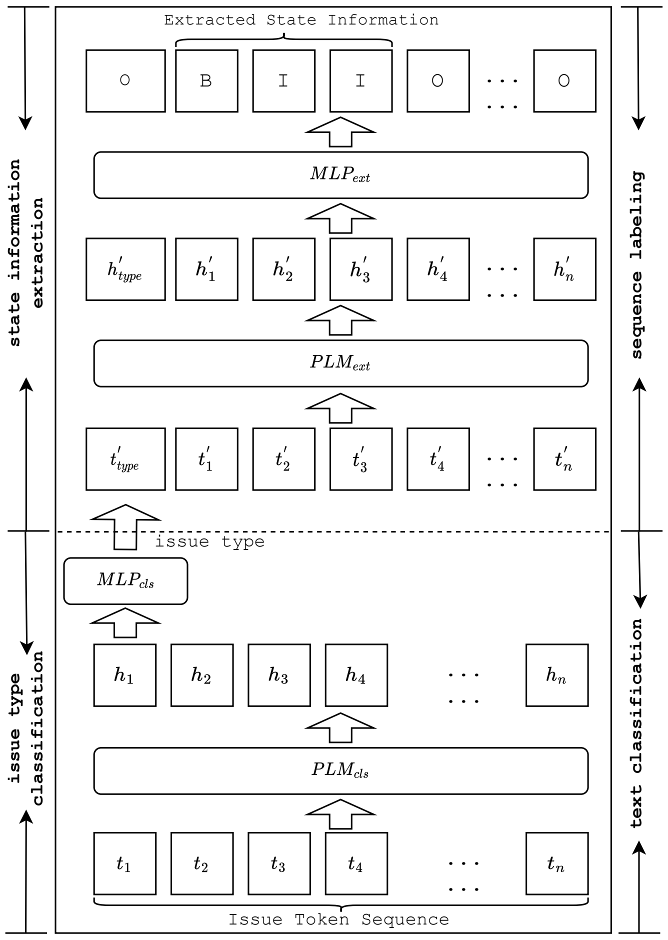


Fig. 4: Extraction stage.

B. Grounding Stage

In the context of deep learning, grounding [55] 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 model's capacity to represent code change 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 in Sec. III-B1, issues can offer descriptions of the state that the code should exhibit before and after the change. In other words, the state information can be regarded as a natural language summary of the code change. Therefore, it 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 sequence, the objective of this stage is to obtain a grounded model $F_{grounded}$ that can align input code change with words possessing the same semantic information. By concatenating state information of related issues as the target language, we can model the training process of F as a sequence-to-sequence task [56]. Since the model outputs a probability distribution over the vocabulary on all translation steps, the summed *cross-entropy* loss is also used as the objective function. After grounding, we acquire a model $F_{grounded}$, which possess the knowledge learned from the correlation between commits and issues.

C. Fine-tuning Stage

The fine-tuning stage is relatively simple: Given an input sequence of code change, $F_{grounded}$ is fine-tuned to generate a commit message in this stage. Note that the structure of $F_{grounded}$ and F are the same, but the parameters are different.

V. EXPERIMENTAL SETUP

A. Research Question

- **RQ1: Overall effectiveness.** To what extent can *ExGroFi* enhance the performance of existing CMG models?
- **RQ2: Extraction performance.** How effective does the first stage of *ExGroFi* extract state information from issues?

- **RQ3: Contribution of grounding stage.** Do models really learn useful correlation knowledge through the grounding stage?
- **RQ4: Human evaluation.** How do `ExGroFi`-enhanced models perform from the perspective of human evaluation?

B. Data

Since there was no commit-issue parallel corpus before, we use the dataset described in Section III for all experiments. The annotated issues in the fine part are used to evaluate models in the extraction stage. The base part is used to evaluate the grounding stage and the generation stage.

C. Enhanced Models

We use `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 [23], CodeBERTa [57], and UniXcoder [58].
- **Encoder-decoder models** possess both pre-trained encoder for representation and decoder for generation. We consider two encoder-decoder models CodeTrans [24] and CodeT5 [25]. We experimented with both the SMALL and BASE versions of these two models.
- **Hybrid models** use both retrieval-based and learning-based techniques. We select RACE [19], which is a retrieval-augmented neural commit message generation model.

Note that models using the ASTs of the code (such as FIRA [17] and ATOM [18]) do not participate in our evaluation. This is because the code diff fragments provided in the dataset we used are unparseable.

D. Implementation

a) Models: We use the official parameters downloaded from Hugging Face [59] to initialize our models. 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 [60] decoder for encoder-only models. The hyperparameters of these random decoders are set according to the configuration of the corresponding encoder model. For RACE, we initialized it with the weight of CodeT5-base as reported in its original paper. In addition, for the pre-trained language model in the extraction stage, we use CodeBERT.

b) Tokenizers: As mentioned in Sec. IV-A1, we add three special tokens “[BR]”, “[FR]”, and “[EN]” for the tokenizer of extraction stage to indicate the type of issue type.

c) Environments: The experiments are performed on an 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.

E. Automatic Metrics

1) Commit Message Generation: Numerous automated metrics have been employed for the evaluation of the CMG 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** [61] 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** [62] is a metric that evaluates the quality of summaries by measuring the longest common subsequence (LCS) [63] between the generated text and the reference texts. It is less sensitive to the exact word order than BLEU but provides a more recall-oriented evaluation.
- **METEOR** [64] calculates the harmonic mean of unigram precision and recall between the generated text and the reference text, considering exact matches, synonyms, and stemmed forms of words.
- **CIDEr** [65] 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.

2) State Information Extraction: Drawing upon the information extraction domain [66], we opted to use precision, recall, and F1-score (the harmonic mean of precision and recall) to evaluate the performance of the extraction stage. During the actual training of the state information extractor, we utilized the fine part of the dataset, which was randomly divided into an 8:1:1 ratio for the training, validation, and testing sets. A micro-F1 score 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. Therefore, 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 number of matching character between the model output string and the golden string. Then we calculate a similarity score by dividing the number of matching characters by the average length of the strings. If the score exceeds a predetermined threshold τ (we set the value as 0.8), we consider the two strings to match, indicating that the model has extracted the correct answer.

VI. Results and Analysis

In this section, we present the overall effectiveness of `ExGroFi`-enhanced models for CMG (RQ1) in Sec. VI-A, the performance of the extraction stage (RQ2) in Sec. VI-B, the contribution of grounding stage (RQ3) in Sec. VI-C and human evaluation (RQ4) in Sec. VI-D.

TABLE I: Generation performance for *ExGroFi*-enhanced commit message generation models.

Model	BLEU			ROUGE-L			METEOR			CIDEr		
	Ori.	Enh.	Imp.	Ori.	Enh.	Imp.	Ori.	Enh.	Imp.	Ori.	Enh.	Imp.
CodeBERT [23]	8.81	9.09	3%	15.85	16.05	1%	4.75	5.0	5%	0.09	0.11	22%
CodeBERTa [57]	8.43	8.50	1%	14.86	15.66	5%	4.92	5.12	4%	0.09	0.11	22%
unixcoder [58]	9.46	9.54	1%	17.04	17.59	3%	5.57	6.18	11%	0.14	0.16	14%
encoder-only average	8.9	9.04	2%	15.92	16.43	3%	5.08	5.43	7%	0.11	0.13	18%
CodeTrans-small [24]	11.08	12.66	14%	18.62	20.84	12%	7.98	9.07	14%	0.3	0.46	53%
CodeTrans-base [24]	19.27	20.37	6%	27.34	28.53	4%	16.37	18.31	12%	1.0	1.08	8%
CodeT5-small [25]	11.40	12.59	10%	20.70	22.68	10%	8.86	10.02	13%	0.36	0.49	37%
CodeT5-base [25]	19.96	20.94	5%	30.07	31.46	5%	16.99	18.39	8%	1.04	1.11	7%
RACE [19]	21.06	22.47	7%	30.17	36.09	20%	17.23	18.64	8%	1.14	1.51	32%
encoder-decoder + hybrid average	16.55	17.81	8%	25.38	27.92	10%	13.486	14.886	10%	0.768	0.93	21%

"**Ori.**" means Original; "**Enh.** " is Enhanced; "**Imp.**" indicates Improvement.

A. RQ1: Overall Effectiveness

The overall effectiveness of *ExGroFi* is presented in Table I. It can be observed that the generation performance of each model is improved after being trained using *ExGroFi*. Among them, the overall generation performance of encoder-only models is relatively inferior, which can be attributed to their decoders being initialized randomly. In contrast, the generation 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 *ExGroFi* consistently outperform models that solely rely on fine-tuning. This demonstrates the effectiveness of *ExGroFi* in the CMG 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 CIDEr, 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 scenario of CMG task [4], as different commit messages often contain vocabulary which is unique to specific code repositories, such as particular class names and variable names. In addition, being limited by input length, original models can not fully cover all information in the code change. But with the aid of *ExGroFi*, models can obtain knowledge from issues, significantly reducing the loss of information when representing code change and consequently producing more rational commit messages.

✍ **Answer to RQ-1:** ▶ *ExGroFi improves generation performance in the CMG task for encoder-only, encoder-decoder and hybird models, with encoder-decoder models showing superior results. Notable improvements are observed in CIDEr, indicating better capture of specific human-relevant details. ExGroFi also aids in reducing information loss, enabling more rational commit messages.* ◀

B. RQ2: Extraction Performance

In the extraction stage of *ExGroFi*, a learning-based model is employed to extract state information, necessitating an evaluation of its performance. During the training phase of the state information extractor, we utilized the fine part of the dataset, which was randomly divided into an 8:1:1 ratio for the training, validation, and testing sets. In Table II, 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 rightmost 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 expected state information (F1-score 81.08%) compared to actual state information (F1-score 61.68%). This is primarily due to the fact that expected state is often proposed by more experienced developers or users [28], 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 [66], this performance is sufficient to meet the demands of state information extraction for data in our scenario.

Figure 5 provides an example of the extracted state information from a real issue. In the figure, we show both the title and the body of this issue. Actual state information is

TABLE II: State information extraction performance

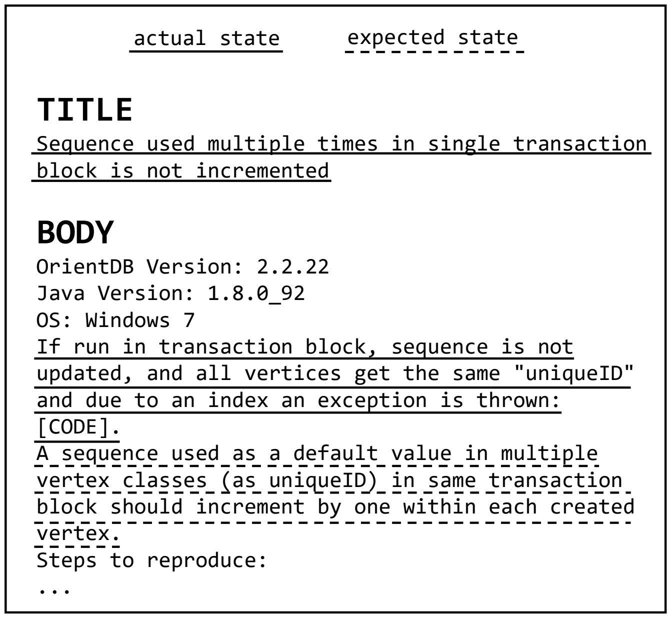
Approach		codebert	+ issue type
Actual State	**P**	56.90	**58.93**
	R	63.46	**64.71**
	F1	60.00	**61.68**
Expected State	**P**	80.77	**84.91**
	R	72.41	**77.59**
	F1	76.36	**81.08**
Micro-F1		68.18	**71.56**

indicated by solid underlines, while expected state information is indicated by dashed underlines. The former describes the actual behavior which is unexpected, and the latter suggests the expected behavior.

✍ **Answer to RQ-2:** ▶ *In the extraction stage of `ExGroFi`, incorporating category information significantly improves state information extraction performance, with better results for expected state (F1-score 81.08%) than actual state (F1-score 61.68%). The overall micro-F1 score reaches 71.56%, sufficient to meet state information extraction demands for the scenario.* ◀

C. 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 our dataset, resulting in two pairs of vectors. 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 the 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 grounded encoder are more similar. The Mann-Whitny-Wilcoxon test [67] (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.



```
actual state      expected state

TITLE
Sequence used multiple times in single transaction
block is not incremented

BODY
OrientDB Version: 2.2.22
Java Version: 1.8.0_92
OS: Windows 7
If run in transaction block, sequence is not
updated, and all vertices get the same "uniqueID"
and due to an index an exception is thrown:
[CODE].
A sequence used as a default value in multiple
vertex classes (as uniqueID) in same transaction
block should increment by one within each created
vertex.
Steps to reproduce:
...
```

Fig. 5: Example of extracted state information.

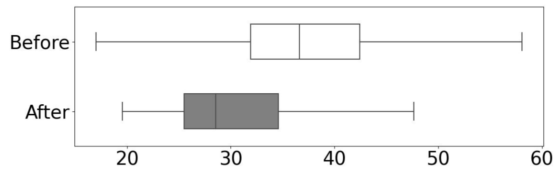


Fig. 6: Euclidien distance between embedding of code change and commit message before and after grounding.

✍ **Answer to RQ-3:** ▶ *The grounding stage of `ExGroFi` effectively integrates commit-related knowledge from issues. A comparative experiment shows significantly smaller Euclidean distances between code change and commit message representations in `ExGroFi`-enhanced models, indicating better semantic alignment and more reasonable commit message generation.* ◀

D. RQ4: Human Evaluation

While automatic metrics can provide valuable insights, the lack of semantic understanding still makes them insufficient for a comprehensive evaluation. Therefore, following Shi et al. [19], 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 the base part and design a questionnaire for evaluation. 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. We invited four experienced developers to conduct human evaluation, including two software engineers with over two years of work experience and two graduate students with long-term internship experience. Each evaluator needs to assess the quality of the generated commit messages from the following four aspects.:

- **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 developer must 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 developer fills in the questionnaire independently and the agreement among them is measured by Fleiss' kappa [68].

TABLE III: Result of human evaluation.

Indicator	Win (N-%)	Lose (N-%)	Tie (N-%)	Kappa
Rat.	35 (70%)	10 (20%)	5 (10%)	0.70
Comp.	25 (50%)	14 (28%)	11 (22%)	0.67
Conc.	21 (42%)	20 (40%)	9 (18%)	0.65
Expr.	17 (34%)	17 (34%)	16 (32%)	0.66

N is short for number; % indicates percentage.
Rat. means Rationality. **Comp.** is Comprehensiveness.
Conc. is Conciseness. **Expr.** represents Expressiveness.

Table III reports the results of human evaluation. The values of kappa are all above 0.6, indicating substantial agreement among the four developers. According to the results, we notice that *ExGroFi* significantly improves upon rationality while achieving 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.

After human evaluation, we also select three examples that intuitively demonstrate the enhancement of the model by the tool, as shown in Table IV. Each example not only provides the generated results of the original model and the *ExGroFi*-enhanced model, but also provides partial text from the issue related to that commit. It can be seen that, the commit message generated by the original model only provides a rough description of the modifications. In comparison, the *ExGroFi*-enhanced model's output includes specific descriptions of the modified objects (e.g.,"handler in Mono.subscribe()" in Example 2 and"when starting process instance" in Example 3) as well as the reasons for the modifications (e.g.,"to reclaim the memory" in Example 1 and "Fix bug" in Example 3). These examples further demonstrate that *ExGroFi*, by introducing correlation knowledge between commits and issues, enables models to have the ability to generate more rational results.

✍ **Answer to RQ-4:** ▶ *Human evaluation of commit messages generated by ExGroFi-enhanced models shows significant improvement in rationality while maintaining comparable performance in comprehensiveness, conciseness, and expressiveness. This confirms that ExGroFi effectively enables models to learn rational information from the correlation between commits and issues.* ◀

VII. Discussion

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

A. 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 [19] and public checkpoints of all pre-trained models on Hugging Face [59]. To ensure the optimal hyperparameters for each model, we conducted numerous iterative experiments and tested models 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 the dataset. To mitigate this threat, we employed the official GitHub REST API [47] to collect data from the top 1,000 repositories ranked by the number of stars. Furthermore, referencing previous research [4], [9], [12], 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 [12], [19]. 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.

B. Limitations

This section discusses the limitations of 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 stages. The quality of issue data and the performance of the upstream stage both influence the final generation outcome. Lastly, although the results generated by *ExGroFi*-enhanced models contain more rational information, these commit messages are sometimes not concise enough.

TABLE IV: Examples that demonstrate *ExGroFi*'s effect.

1	**Before**	Add support for custom field.
	After	Fix TimingKey **to reclaim the memory**.
	Issue	... There is no mechanism to reclaim the memory after the filters are unreachable ...
2	**Before**	Reduce visualisation to fix issue.
	After	Fix exception with error **handler in Mono.subscribe()**.
	Issue	... Error handler in Mono.subscribe() should handle exceptions thrown by subscriber ...
3	**Before**	Update process instance.
	After	**Fix bug when starting process instance**.
	Issue	... No tenant is set when starting process instance ... ... Expect a tenant user can start a process instance in task application ...

"**Before**" and "**After**" represent the commit messages generated by the model before and after using *ExGroFi*, respectively.
"**Issue**" provides part of the text in the related issue; among them, the red text indicates the actual state, and the green text indicates the expected state.

VIII. RELATED WORK

Existing methods for commit message generation can be categorized as template-based, learning-based, information retrieval-based and hybrid models.

Early methods were typically template-based [2], [10], [11]. These techniques used parsers to analyze the modification of source code and generated commit messages with pre-defined rules. For instance, Buse et al. [10] symbolically executed code changes to acquire path predicates, then generated commit message using a set of pre-defined rules based on the path. Cortés-Coy et al. [2] proposed a template based on stereotypes [69], [70], then filled the template with extracted information to generate commit message.

Learning-based methods [6], [8], [14], [15], [17] drew inspiration from the idea of neural machine translation and modeled the generation of commit message as a sequence-to-sequence task. Jiang et al. [8] early attempted to respectively treat code change and commit message as input and output. Loyola et al. [6] proposed an encoder-decoder model with the attention mechanism proposed by Luong et al. [71]. Dong et al. [17] represented code change via fine-grained graphs and generated commit message with graph neural network [72].

Retrieval-based approaches [12], [13] sorted existing commit messages according to similarity as output. Liu et al. [12] represented code change as bag-of-words vectors and then used the nearest neighbor algorithm to sort. Hoang et al. [13] learned a distributed representation for code change guided by commit message. The learned representations are used to adapt the bag-of-words vectors of the former model.

Hybrid architectures [18] combined the former two types of approaches. Liu et al. made use of the Abstract Syntax Trees of the code change in the learning module and sorted them according to cosine similarity in the retrieval module. The final result is output by a ranking module that prioritize the commit message obtained from the former two modules.

Influenced by powerful pre-trained language models in natural language processing [39], [21], many pre-trained models that provide distributed representations of code have emerged in recent years [23], [24], [25]. At present, the main paradigm to generate commit message is to combine pre-trained code representation models with task-specific generation modules [19].

IX. CONCLUTION AND FUTURE WORK

In this work, we delve into the correlation between commits and issues, addressing a previously unexplored aspect of automatic commit message generation. We collect a large commit-issue parallel dataset, allowing for a deeper understanding of the rational information issues provide. A novel paradigm `ExGroFi` is also proposed to extract and incorporate fine-grained knowledge from issues into the commit message generation process. The extensive evaluation demonstrates the effectiveness of `ExGroFi` 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 commit message generation, particularly in exploring methods of extracting and incorporating external information from issues and other modalities. For instance, researchers can explore the pre-training of large-scale language models based on such commit-issue correlation, 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 automated software engineering.

X. ARTIFACT AVAILABILITY

We make our code and dataset publicly available at: https://anonymous.4open.science/r/ExGroFi-FA21

XI. ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (Grant Nos. 62276017, U1636211, 61672081), the Fund of the State Key Laboratory of Software Development Environment (Grant No. SKLSDE-2021ZX-18) and the NATURAL project which has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant No. 949014).

REFERENCES

[1] R. P. Buse and W. R. Weimer, "Automatically documenting program changes," in *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 33–42. [Online]. Available: https://doi.org/10.1145/1858996.1859005

[2] L. F. Cortés-Coy, M. Linares-Vásquez, J. Aponte, and D. Poshyvanyk, "On automatically generating commit messages via summarization of source code changes," in *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*, 2014, pp. 275–284.

[3] Mockus and Votta, "Identifying reasons for software changes using historic databases," in *Proceedings 2000 International Conference on Software Maintenance*, 2000, pp. 120–130.

[4] Y. Tian, Y. Zhang, K.-J. Stol, L. Jiang, and H. Liu, "What makes a good commit message?" in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 2389–2401. [Online]. Available: https://doi.org/10.1145/3510003.3510205

[5] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen, "Boa: A language and infrastructure for analyzing ultra-large-scale software repositories," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 422–431.

[6] P. Loyola, E. Marrese-Taylor, and Y. Matsuo, "A neural architecture for generating natural language descriptions from source code changes," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 287–292. [Online]. Available: https://aclanthology.org/P17-2045

[7] W. Maalej and H.-J. Happel, "Can development work describe itself?" in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, 2010, pp. 191–200.

[8] S. Jiang, A. Armaly, and C. McMillan, "Automatically generating commit messages from diffs using neural machine translation," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2017, pp. 135–146.

[9] W. Tao, Y. Wang, E. Shi, L. Du, S. Han, H. Zhang, D. Zhang, and W. Zhang, "On the evaluation of commit message generation models: An experimental study," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2021, pp. 126–136.

[10] R. P. Buse and W. R. Weimer, "Automatically documenting program changes," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, 2010, pp. 33–42.

[11] J. Shen, X. Sun, B. Li, H. Yang, and J. Hu, "On automatic summarization of what and why information in source code changes," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. IEEE, 2016, pp. 103–112.

[12] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang, "Neural-machine-translation-based commit message generation: How far are we?" in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 373–384. [Online]. Available: https://doi.org/10.1145/3238147.3238190

[13] T. Hoang, H. J. Kang, D. Lo, and J. Lawall, "Cc2vec: Distributed representations of code changes," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 518–529. [Online]. Available: https://doi.org/10.1145/3377811.3380361

[14] S. Xu, Y. Yao, F. Xu, T. Gu, H. Tong, and J. Lu, "Commit message generation for source code changes," in *IJCAI*, 2019.

[15] Q. Liu, Z. Liu, H. Zhu, H. Fan, B. Du, and Y. Qian, "Generating commit messages from diffs using pointer-generator network," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 2019, pp. 299–309.

[16] L. Y. Nie, C. Gao, Z. Zhong, W. Lam, Y. Liu, and Z. Xu, "CoreGen: Contextualized Code Representation Learning for Commit Message Generation," *Neurocomputing*, vol. 459, pp. 97–107, Oct. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092523122100792X

[17] J. Dong, Y. Lou, Q. Zhu, Z. Sun, Z. Li, W. Zhang, and D. Hao, "Fira: Fine-grained graph-based code change representation for automated commit message generation," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 970–981. [Online]. Available: https://doi.org/10.1145/3510003.3510069

[18] S. Liu, C. Gao, S. Chen, L. Y. Nie, and Y. Liu, "Atom: Commit message generation based on abstract syntax tree and hybrid ranking," *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1800–1817, 2022.

[19] E. Shi, Y. Wang, W. Tao, L. Du, H. Zhang, S. Han, D. Zhang, and H. Sun, "Race: Retrieval-augmented commit message generation," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022, pp. 5520–5530.

[20] H. Wang, X. Xia, D. Lo, Q. He, X. Wang, and J. Grundy, "Context-aware retrieval-based deep commit message generation," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 4, jul 2021. [Online]. Available: https://doi.org/10.1145/3464689

[21] J. Li, T. Tang, W. X. Zhao, and J.-R. Wen, "Pretrained language model for text generation: A survey," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, Z.-H. Zhou, Ed. International Joint Conferences on Artificial Intelligence Organization, 8 2021, pp. 4492–4499, survey Track. [Online]. Available: https://doi.org/10.24963/ijcai.2021/612

[22] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, "Pre-trained models for natural language processing: A survey," *Science China Technological Sciences*, vol. 63, no. 10, pp. 1872–1897, 2020.

[23] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 1536–1547. [Online]. Available: https://aclanthology.org/2020.findings-emnlp.139

[24] A. Elnaggar, W. Ding, L. Jones, T. Gibbs, T. Feher, C. Angerer, S. Severini, F. Matthes, and B. Rost, "Codetrans: Towards cracking the language of silicone's code through self-supervised deep learning and high performance computing," *CoRR*, vol. abs/2104.02443, 2021. [Online]. Available: https://arxiv.org/abs/2104.02443

[25] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 8696–8708. [Online]. Available: https://aclanthology.org/2021.emnlp-main.685

[26] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 361–370. [Online]. Available: https://doi.org/10.1145/1134285.1134336

[27] J. Zhu, M. Zhou, and A. Mockus, "Effectiveness of code contribution: From patch-based to pull-request-based tools," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 871–882. [Online]. Available: https://doi.org/10.1145/2950290.2950364

[28] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2008, pp. 308–318.

[29] H. Tian, X. Tang, A. Habib, S. Wang, K. Liu, X. Xia, J. Klein, and T. F. BissyandÉ, "Is this change the answer to that problem? correlating descriptions of bug and code changes for evaluating patch correctness," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3551349.3556914

[30] R. Vieira, A. da Silva, L. Rocha, and J. a. P. Gomes, "From reports to bug-fix commits: A 10 years dataset of bug-fixing activity from 55 apache's open source projects," in *Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering*, ser. PROMISE'19. New York, NY, USA: ACM, 2019, pp. 80–89. [Online]. Available: http://doi.acm.org/10.1145/3345629.3345639

[31] Y. Zhou and A. Sharma, "Automated identification of security issues from commit messages and bug reports," in *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, 2017, pp. 914–919.

[32] Atlassian, "Jira: Issue and project tracking software," 2021, accessed: 2023-04-22. [Online]. Available: https://www.atlassian.com/software/jira

[33] Trello, Inc., "Trello: Team collaboration and management tool," 2021, accessed: 2023-04-22. [Online]. Available: https://www.trello.com

[34] GitHub Inc., "Github," 2021, accessed: 2023-04-22. [Online]. Available: https://github.com

[35] X. Tan and M. Zhou, "How to communicate when submitting patches: An empirical study of the linux kernel," *Proceedings of the ACM on Human-Computer Interaction*, vol. 3, no. CSCW, pp. 1–26, 2019.

[36] A. A. Sawant, G. Huang, G. Vilen, S. Stojkovski, and A. Bacchelli, "Why are features deprecated? an investigation into the motivation behind deprecation," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2018, pp. 13–24.

[37] O. Baysal, R. Holmes, and M. W. Godfrey, "Situational awareness: personalizing issue tracking systems," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 1185–1188.

[38] J. Aranda and G. Venolia, "The secret life of bugs: Going past the errors and omissions in software repositories," in *2009 IEEE 31st international conference on software engineering*. IEEE, 2009, pp. 298–308.

[39] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[40] A. Zagalsky, J. Feliciano, M.-A. Storey, Y. Zhao, and W. Wang, "The emergence of github as a collaborative platform for education," in *Proceedings of the 18th ACM conference on computer supported cooperative work & social computing*, 2015, pp. 1906–1917.

[41] GitHub Inc., "Issues: Github's issue tracking feature," 2021, accessed: 2023-04-22. [Online]. Available: https://github.com/features/issues

[42] S. Onoue, H. Hata, and K.-i. Matsumoto, "A study of the characteristics of developers' activities in github," in *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, vol. 2, 2013, pp. 7–12.

[43] Y. Li, M. Soliman, and P. Avgeriou, "Identifying self-admitted technical debt in issue tracking systems using machine learning," *Empirical Software Engineering*, vol. 27, no. 6, p. 131, 2022.

[44] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Frequently asked questions in bug reports," University of Calgary, Tech. Rep., 2009.

[45] G. Destefanis, M. Ortu, D. Bowes, M. Marchesi, and R. Tonelli, “On measuring affects of github issues’ commenters,” in *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering*, ser. SEmotion ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 14–19. [Online]. Available: https://doi.org/10.1145/3194932.3194936

[46] PyGithub Contributors, “Pygithub: A python library to access the github api v3,” 2021, accessed: 2023-04-22. [Online]. Available: https://pygithub.readthedocs.io/en/latest/index.html

[47] GitHub Inc., “Github rest api reference,” 2022, accessed: 2023-04-22. [Online]. Available: https://docs.github.com/en/rest?apiVersion=2022-11-28

[48] T.-D. B. Le, M. Linares-Vásquez, D. Lo, and D. Poshyvanyk, “Rclinker: Automated linking of issue reports and commits leveraging rich contextual information,” in *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE, 2015, pp. 36–47.

[49] Z. Liao, D. He, Z. Chen, X. Fan, Y. Zhang, and S. Liu, “Exploring the characteristics of issue-related behaviors in github using visualization techniques,” *IEEE Access*, vol. 6, pp. 24 003–24 015, 2018.

[50] I. Heartex, “Label studio: Open source data labeling and annotation tool,” 2021, accessed: 2023-04-22. [Online]. Available: https://labelstud.io

[51] A. Akbik, D. Blythe, and R. Vollgraf, “Contextual string embeddings for sequence labeling,” in *Proceedings of the 27th international conference on computational linguistics*, 2018, pp. 1638–1649.

[52] F. Murtagh, “Multilayer perceptrons for classification and regression,” *Neurocomputing*, vol. 2, no. 5-6, pp. 183–197, 1991.

[53] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, “Neural architectures for named entity recognition,” in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, Jun. 2016, pp. 260–270. [Online]. Available: https://aclanthology.org/N16-1030

[54] N. Alshammari and S. Alanazi, “The impact of using different annotation schemes on named entity recognition,” *Egyptian Informatics Journal*, vol. 22, no. 3, pp. 295–302, 2021.

[55] K. R. Chandu, Y. Bisk, and A. W. Black, “Grounding ‘grounding’ in NLP,” in *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. Online: Association for Computational Linguistics, Aug. 2021, pp. 4283–4305. [Online]. Available: https://aclanthology.org/2021.findings-acl.375

[56] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” *Advances in neural information processing systems*, vol. 27, 2014.

[57] Hugging Face, “Codeberta-small-v1 model,” 2023, accessed: 2023-04-30. [Online]. Available: https://huggingface.co/huggingface/CodeBERTa-small-v1

[58] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, “UniXcoder: Unified cross-modal pre-training for code representation,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 7212–7225. [Online]. Available: https://aclanthology.org/2022.acl-long.499

[59] Hugging Face, “Hugging face: Natural language processing and machine learning platform,” 2021, accessed: 2023-04-22. [Online]. Available: https://huggingface.co

[60] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[61] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: a method for automatic evaluation of machine translation,” in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[62] C.-Y. Lin, “Rouge: A package for automatic evaluation of summaries,” in *Text summarization branches out*, 2004, pp. 74–81.

[63] M. Paterson and V. Dančík, “Longest common subsequences,” in *Mathematical Foundations of Computer Science 1994: 19th International Symposium, MFCS’94 Košice, Slovakia, August 22–26, 1994 Proceedings 19*. Springer, 1994, pp. 127–142.

[64] S. Banerjee and A. Lavie, “Meteor: An automatic metric for mt evaluation with improved correlation with human judgments,” in *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 2005, pp. 65–72.

[65] R. Vedantam, C. Lawrence Zitnick, and D. Parikh, “Cider: Consensus-based image description evaluation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 4566–4575.

[66] Z. Nasar, S. W. Jaffry, and M. K. Malik, “Named entity recognition and relation extraction: State-of-the-art,” *ACM Comput. Surv.*, vol. 54, no. 1, feb 2021. [Online]. Available: https://doi.org/10.1145/3445965

[67] P. E. McKnight and J. Najab, “Mann-whitney u test,” *The Corsini encyclopedia of psychology*, pp. 1–1, 2010.

[68] J. L. Fleiss and J. Cohen, “The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability,” *Educational and psychological measurement*, vol. 33, no. 3, pp. 613–619, 1973.

[69] N. Dragan, M. L. Collard, M. Hammad, and J. I. Maletic, “Using stereotypes to help characterize commits,” in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2011, pp. 520–523.

[70] N. Dragan, M. L. Collard, and J. I. Maletic, “Reverse engineering method stereotypes,” in *2006 22nd IEEE International Conference on Software Maintenance*. IEEE, 2006, pp. 24–34.

[71] T. Luong, H. Pham, and C. D. Manning, “Effective approaches to attention-based neural machine translation,” in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 1412–1421. [Online]. Available: https://aclanthology.org/D15-1166

[72] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, “A comprehensive survey on graph neural networks,” *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.