Towards Generating the Rationale for Code Changes

Francesco Casillo*, Antonio Mastropaolo†, Gabriele Bavota‡, Vincenzo Deufemia* and Carmine Gravino*
*Department of Computer Science
University of Salerno
Emails: fcasillo@unisa.it, deufemia@unisa.it, gravino@unisa.it
†Computer Science
William & Mary
Email: amastropaolo@wm.edu
‡Software Institute
Università della Svizzera Italiana
Email: gabriele.bavota@usi.ch

***Abstract*—Commit messages are essential to understand changes in software projects, providing a way for developers to communicate code evolution. Generating effective commit messages that explain the rationale behind changes is a challenging and time-consuming task. While previous research has shown success in automating straightforward commit messages (e.g., "add README"), our study explores a more complex task: generating rationale explanations for code changes. We developed a method to identify rationale sentences in commit messages and compiled a dataset of 45,945 commits with their corresponding rationales. A pre-trained model was trained on this dataset to generate rationale explanations. While the approach we engineered for the extraction of rationale from commit messages exhibited a 75% precision, the model trained to generate the rationale only worked in a minority of cases. Our findings highlight the difficulty of the tackled task and the need for additional research in the area. We release our dataset and code to foster the investigation of this problem.**

***Index Terms*—Rationale of Code Changes, Deep Learning, Natural Language Processing.**

I. Introduction

The usage of proper development infrastructures and appropriate means of communication is crucial in both industrial and open-source projects [1]. Communication among developers does not only come through explicit channels (*e.g.,* issue tracker, instant messaging) but also via implicit information left during development activities, such as commit messages. The latter serves as a form of documentation aimed at explaining *what* was changed in a given commit (*e.g.,* "*updated README file ...*") and/or *why* a specific change occurred (*e.g.,* "*... to specify the new terms of service*") [2].

Despite their importance, developers do not always have the time to craft commit messages [2]. To support them in this task, researchers proposed techniques for the generation of commit messages [3]–[6]. While these techniques are promising, they are mostly successful in documenting *what* changed, rather than *why* a change has been performed.

For example, Liu *et al.* [7] showed that one of the state-of-the-art techniques was mostly successful in generating "trivial messages", namely those containing redundant information which can be easily inferred by looking at the list of impacted files (*e.g.,* "*update changelog*").

The focus on generating *what* changed rather than documenting *why* the change was performed is the result of design choices made when creating the dataset, and in particular the target message C_m. The latter usually is the first sentence in the commit message [4], [8], [9], which is extracted from what is known as "commit subject", namely a short text summarizing changes made in the commit. Indeed, as shown by Jiang *et al.* [10], more than 80% of commit subjects consist of just one sentence. However, commit messages include a "commit body", providing a longer explanation for the implemented changes (possibly including the *why*). We refer to the "*why*" as the *rationale for code changes*.

While the *what* can be inferred by looking at the commit diff, the *why* is more difficult to infer for a human and, as such, its automatic generation may help in better understanding the evolution of the code base: suggesting the reasons behind code changes is a critical need in software development [11], and previous studies reported the rationale behind code changes to be the most common [12] and crucial [13] piece of information developers look for in the code change history.

The availability of a tool that can automatically generate the rationale for code changes can promote a deeper understanding of code evolution. This is the problem we tackle by training a generative Deep Learning (DL) model on a dataset composed by pairs $\langle C_{diff}, \{C_r\}\rangle$, where $\{C_r\}$ represents sentences expressing the rationale for C_{diff}.

Fig. 1 overviews all the steps we take in this paper towards the goal of automatically generating the rationale for a code commit. There are two main steps. First, we engineered a procedure to automatically mine commits with their rationale to build the dataset of $\langle C_{diff}, \{C_r\}\rangle$ pairs. Then, we used such dataset to instruct a LLM to automatically generate a possible rationale for a code diff.

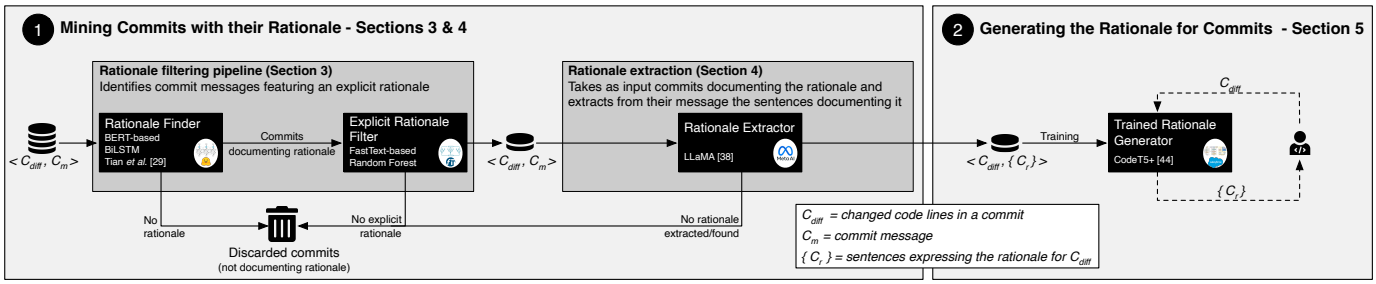

Fig. 1. Overview of the process we adopt towards generating the rationale for code changes

We defined a pipeline taking as input a commit message[1] and classifying it as containing or not a description of the rationale for the implemented change (Section III). While there is a technique in the literature to classify commit messages as containing or not the rationale for the change [2], this approach considers as a valid rationale what the authors define *implicit* rationale (*i.e.,* a rationale which is not explicitly stated but can be inferred from the message), which is inappropriate for our goal. Still, we use this approach proposed by Tian *et al.* [2] as the first step of our pipeline (*Rationale Finder* in Fig. 1) to perform a preliminary filtering discarding commits not featuring a rationale (either explicit or implicit). The commits classified by the *Rationale Finder* as featuring a rationale, are then provided as input to the *Explicit Rationale Filter*, being a classifier we trained to keep the commit messages featuring an explicit rationale and to discard the implicit ones.

We run the pipeline on ∼5.9k GitHub Java-based repositories, identifying commits documenting a rationale in their message. From those, we manually labeled a statistically significant sample of 665 commit messages, to (i) check whether they contained an explicit rationale and, in case of affirmative answer, (ii) extract the sentence(s) documenting the rationale. Concerning (i), we found out that the defined pipeline (*i.e., Rationale Finder* followed by *Explicit Rationale Filter*) can identify commit messages featuring an explicit rationale with a precision of 81%. The step (ii), instead, resulted in a dataset of 540 pairs $\langle C_m, \{C_r\}\rangle$, with C_m being the full commit message, and $\{C_r\}$ being the set of sentences from C_m documenting the rationale. This small dataset is not suitable to train a generative model for the automatic rationale generation task, but we used it to define data-driven solutions for the task of rationale extraction from a commit message (*i.e.,* input to the technique is C_m, expected output is $\{C_r\}$).

We used the extracted pairs to train and test a DL-based solution for the automated extraction of rationale from commit messages (*Rationale Extractor* in Fig. 1), to build the large-scale dataset needed for the generative model (Section IV). Among the experimented solutions, few-shot learning specializing a pre-trained large language model (*i.e.,* LLaMA [14]) for the task of interest worked the best. Using just 15 examples (*i.e.,* pairs $\langle C_m, \{C_r\}\rangle$), LLaMA learned how to extract a meaningful rationale from the input commit message with a ∼75% precision. LLaMA finalizes the definition of our procedure to mine commits with their rationale (see Fig. 1): this procedure starts with commits mined from software repositories and ends with a subset of those commits paired with the rationale extracted from their commit message (*i.e.,* the $\langle C_{diff}, \{C_r\}\rangle$ pairs).

As result, we built a dataset of 45,945 commits (diffs) associated with their rationale. These commits have been used to fine-tune CodeT5+ [15], a LLM pre-trained on code and natural language, for the commit rationale generation task (input to the technique is C_{diff}, expected output $\{C_r\}$) — see Section V. Once trained, the model can be used to generate the rationale for a code diff (see dashed lines in Fig. 1). Our empirical assessment shows that CodeT5+ can synthesize a meaningful rationale for a commit message in a minority of cases from our test set. The level of performance is definitively not sufficient to make our approach a solution for developers. However, we still feel the presented empirical investigation as a valuable step in addressing the tackled problem, considering the (at least partial) positive results achieved in the automated mining of rationale from commits and our documented experience in fine-tuning CodeT5+ for such a task. We hope that the released datasets and code [16] can be used to foster research in this area.

To sum up, the contributions of our work are:

1) Introduced a pipeline combining a *Rationale Finder* and an *Explicit Rationale Filter* to identify commits with explicit rationale from GitHub repositories.
2) Exploited few-shot learning to deal with the task of rationale extraction, implementing a LLaMA-based *Rationale Extractor*.
3) Conducted an empirical evaluation of the CodeT5+ model's ability to generate rationale, laying groundwork for future research into *Rationale Generator*.
4) Released datasets, code, and the pipeline to the public, supporting further research and development in automated rationale documentation [16].

Section II reviews related works in the field. Section III details the process of mining commits documenting the rationale. The methodology for extracting the rationale is outlined in Section IV.

[1]From now on we indicate with "commit message" the concatenation of the commit subject followed by the commit body.

The approach to generating rationale for commit differences is elaborated in Section V. Section VI addresses the threats to the validity of our work. Finally, Section VII concludes the paper offering an overview of the findings and our future research directions.

II. Related work

Our work relates to studies on rationale information in software artifacts. Al Safwan *et al.* [11] interviewed developers, finding that >80% highlight the importance of knowing the rationale behind code commits, while >30% face difficulties in locating this information. Some proposals [17], [18] aim to support developers by making decision rationales explicit or by creating automated rationale extraction systems, inspiring our research.

Other solutions focus on extracting rationales from specific artifacts [19], [20], but these rely on hand-crafted heuristics tailored to specific artifact types, such as Python Enhancement Proposals, and cannot be generalized to commit messages.

The most relevant work is by Tian *et al.* [2], who analyzed ∼1.6k commit messages, finding that only 44% documented both *what* changed and *why*. They developed a Bi-LSTM model identifying rationale with 84% accuracy. This serves as the basis for our *Rationale Finder* pipeline (Fig. 1), as we explain in Section III.

Our work also builds on automatic commit message generation techniques, which can be classified as *rule-based* [21]–[24], *retrieval-based* [7], [25], and *learning-based* [5], [6], [9], [26]–[33], with the latter being most related to our approach. While these techniques generate commit messages based on code diffs (C_{diff}), they primarily document *what* changed, rather than the rationale behind the changes, which is the focus of our work.

III. Mining Commits Documenting the Rationale for Code Changes

We present the process to identify commit messages containing an explicit rationale in Section III-A and the empirical study for accuracy assessment in Section III-B.

A. Approach Description

To avoid reinventing the wheel, we looked for works in the literature proposing solutions for the automated identification of the rationale in commit messages. The closest work we found is the one by Tian *et al.* [2] (see Section II), presenting a technique to classify commit messages as containing or not the *what* (what changed in the commit) and *why* (what we call "rationale") information. Their approach can identify commits containing *why* information with an accuracy of 84%. This result has been achieved on a dataset of 1,649 commit messages manually annotated by the authors as containing or not the *what*/*why* information. While such an approach represents a good starting point for our research, it has a limitation: it identifies as commit messages containing the *why* also those featuring what the authors define an *implicit rationale*.

For example, the commit "*add docs about null responses*" does not have an explicit rationale, but its main purpose to increase code readability can be inferred, so the approach classifies it as containing the *why*. However, these commit messages are unsuitable for our final goal (*i.e.,* training a model to generate the rationale for code changes). For this reason, we decided to build on top of the approach by Tian *et al.* to create a classifier specifically designed to identify commit messages containing an explicit rationale. To this aim, the first author manually inspected the 1,157 commit messages manually labeled by Tian *et al.* as featuring a rationale (which could be implicit or explicit), with the goal of marking those containing an explicit rationale. These commit messages are a subset of the overall 1,649 used for training and testing the approach by Tian *et al.*. Out of the 1,157 inspected commit messages, 351 were identified as reporting an explicit rationale (*e.g.,* "*Close response body to avoid connection leak*").

We used the dataset of 1,157 commit messages labeled as containing an explicit (351) or an implicit (806) rationale to train and test a Random Forest classifier. The idea is that such a classifier can receive as input the commit messages classified by the approach by Tian *et al.* as containing a rationale (explicit or implicit), and discriminate which of those actually feature an explicit rationale.

Since the input of the Random Forest is a text (*i.e.,* the commit message), we experimented with three different word embedding techniques to represent the input: Term Frequency–Inverse Document Frequency (TF-IDF) [34], fastText [35], and Bidirectional Encoder Representations from Transformers (BERT) [36]. TF-IDF is a popular text representation technique that measures the importance of a term in a document corpus by computing a weight based on its frequency in a document and its rarity across the entire corpus. FastText is a lightweight open-source library developed by Facebook AI Research for learning text representations and implementing text classification models. It utilizes n-gram features and employs techniques like subword embeddings to handle out-of-vocabulary words effectively. BERT is a pre-trained Deep Learning model introduced by Devlin *et al.* (2018). It utilizes a Transformer architecture [37] to learn contextualized word embeddings when trained on large textual corpora, and has achieved state-of-the-art performance in various NLP tasks.

We used the default settings of the Random Forest, being: 100 estimators, None as max depth of the trees, 2 min samples split, and 1 min samples leaf. While we experimented with a random grid search to optimize these parameters, we did not observe any major difference in performance, thus keeping the original settings.

B. Empirical Evaluation

We present the design of the empirical study aimed at evaluating the process used to identify commit messages likely to feature an explicit rationale (Section III-B1) and discuss the achieved results (Section III-B2).

1) Study Design: The formulated research question is:

RQ_1: *To what extent is our Random Forest classifier able to discriminate commit messages containing an explicit* vs *an implicit rationale?*

To answer RQ_1, we perform a 10-fold cross validation on the dataset of 1,157 commit messages labeled by the first author as containing (351) or not (806) an explicit rationale. Remember that these are commit messages manually labeled by Tian *et al.* as containing a rationale (implicit or explicit) which have been further specialized in our work. We experiment with the three variants of the Random Forest using different embeddings (*i.e.,* TF-IDF, FastText, and BERT). Since our goal is to use the Random Forest as a filter for the approach by Tian *et al.* (*i.e.,* excluding commits reporting an implicit rationale) to build a large-scale dataset featuring commits with their explicit rationale, it becomes fundamental the *precision* of both these techniques when they report a commit as including a rationale (Tian *et al.*) and as including an explicit rationale (Random Forest). In other words, we may accept false negatives (*i.e.,* commits actually containing an explicit rationale are not identified by the combination of the two techniques), but we want to minimize false positives (*i.e.,* commits wrongly retrieved as containing an explicit rationale). For this reason, we perform an analysis aimed at maximizing the precision of our pipeline.

Both classifiers provide as output a probability assigning each instance (commit message) to each of the output classes (*e.g.,* for the Random Forest, the two classes are implicit/explicit rationale). The higher the probability assigned to a class, the higher the confidence that the model has in the classification.

We study the impact of the classification confidence level on the recall and precision (i) of the approach by Tian *et al.* when classifying the commit message as containing a rationale and (ii) of our Random Forest when reporting an instance as containing an explicit rationale.

This analysis can help in identifying a confidence threshold under which the classifiers should not be trusted. For example, if we find that commit messages classified by the Random Forest as containing an explicit rationale with a probability lower than 60% result in a precision of 20% and a recall of 90%, while moving to a probability of 80% pushes the precision to 70% paying a reasonable cost in recall, we may consider sacrificing a few instances when mining data in the name of increasing the dataset quality.

Once identified the most suitable configuration (*i.e.,* minimum confidence level to trust the classification) to use for both the Tian *et al.* model and our Random Forest, we run the defined pipeline in the wild. Remember that at this point the pipeline works as follows: a commit message is given in input to the Tian *et al.* approach; if it is classified as not containing the *why*, the message is discarded, otherwise it is provided as input to the Random Forest which classifies it as containing or not an explicit rationale.

In case of negative answer, the commit message is discarded, otherwise, it is kept for further future processing in the next steps of our approach. Running this pipeline in the wild has two objectives: (i) it provides us with a population of commits likely to contain an explicit rationale from which we extract a statistically significant sample to manually inspect, thus assessing the capabilities of the pipeline to identify commits relevant for our research; (ii) using the collected commits in the subsequent steps of our approach to train a model for the automatic generation of the rationale behind code commits.

We run our "*Rationale filtering pipeline*" (see Fig. 1) on all commits featured in Java projects hosted on GitHub and having at least 500 commits, 10 contributors, and 10 stars. These filters aim to reduce the chances of considering toy projects in our study (as observed in a previous work [38]). The focus on Java aims instead at following the choice made by most of the works addressing the automated generation of commit messages [39]. We use the tool by Dabić *et al.* [40] to retrieve the list of 5,970 projects matching our selection criteria, while pydriller [41] is used to extract the commits performed in each subject repository, excluding commits that (i) had a message not written in English and (ii) were likely performed by bots. To identify commit messages not written in English we exploit a combination of three libraries/models. The first is the FastText model for language identification that has been trained on data from Wikipedia, Tatoeba, and SETimes [35], [42] to classify the text written in 176 different languages. The other two are the spaCy [43] and the langdetect [44] libraries, which include language detection capabilities. We exclude a commit message if it results to be classified as not written in English by at least one of the three techniques, thus being conservative in mining the commits to be used in our study. Concerning the exclusion of commits contributed by bots, we apply simple heuristics acting on the contributor's name and on the commit message as done in previous work [45]–[48]. For example, commits containing "bot" in the name of the contributor are excluded. The script implementing all filtering rules for bot-related commits is available in our replication package [16]. Commits classified by our pipeline as relevant for our work (*i.e.,* the approach by Tian *et al.* reports the commit as featuring the *why*, and our Random Forest indicates the presence of an explicit rationale) have been saved for later usage with their related information. The application of the proposed process provides a collection of 3,550,080 commits likely to contain an explicit rationale, which becomes 1,565,739 once we remove duplicates. We consider as duplicates commits sharing the same hash (likely due to forked projects) or the same commit message. The exclusion of commits having the same commit message already at this step avoids any sort of data leakage between training and test sets in the later steps of our approach, in which we use these commits as a starting point to create the pairs $\langle C_{diff}, \{C_r\}\rangle$ featuring code changes (C_{diff}) and their rationale ($\{C_r\}$).

TABLE I
RQ$_1$: Results of the 10-fold cross validation

Embedding	Accuracy	"Explicit" Class	
		Precision	Recall
TF-IDF	0.85	0.83	0.65
FastText	**0.84**	**0.87**	**0.56**
BERT	0.81	0.72	0.61

Thus, we exclude all commits which change more than a single diff hunk (with a hunk being a contiguous set of lines impacted by the change) and/or more than 10 code lines. This increases our confidence about the rationale described in the commit message to refer to the diff hunk. In a commit modifying several files, the documented rationale may refer only to a specific part of the diff, introducing noise in the training data. We acknowledge that this choice simplifies the tackled problem because we are focusing on "simple" commits. However, being our work the first attempt to automatically generate the rationale for code changes, we consider this as a safe choice to avoid negative results due to the low quality of the training data (*i.e.,* rationales not clearly linked to code changes). Out of the ∼1.5M commits previously collected, 567,534 impact a single file and 127,699 a single diff hunk. Among those, 96,034 (75%) feature changes impacting at most 10 lines of code.

From this set, a statistically significant sample (99% confidence ±5%) of 665 commits has been randomly extracted and manually inspected to verify whether they actually feature an explicit rationale. Each commit has been independently inspected by the first and the second authors of the paper. Conflicts, which arose in 188 commits (28%), have been solved by the other three authors (∼60 each). We discuss the results of the manual analysis as a validation of our pipeline.

2) Results Discussion: Table I reports the performance of the Random Forest when using different embeddings. We report (i) the accuracy (*i.e.,* the percentage of 1,157 commit messages correctly classified as reporting an implicit or an explicit rationale), and (ii) the recall and precision for the commit messages belonging to the "explicit" class. The recall indicates the percentage of commit messages featuring an explicit rationale that the model was able to identify (351 of those commits are present in our dataset). The precision indicates the percentage of commit messages classified by the Random Forest as reporting an explicit rationale that are actually correct.

Since our interest is in building a dataset of commit messages with an explicit rationale, the focus of our analysis is on the method that presents the best precision. From this perspective BERT is the least performing representation with a precision of 72%. The best in case is FastText (87%) with TF-IDF also producing solid recommendations (83% of precision). As expected FastText pays with a lower recall which, as explained, is not central in our work since we can run our pipeline on thousands of repositories, so building a large-enough dataset even in the presence of false negatives; FastText is the word embedding we adopt.

TABLE II
RQ$_1$: Precision and recall of the approach by Tian *et al.* and of the random forest at different confidence levels

Model	Metrics	Min Confidence level				
		0.5	**0.6**	**0.7**	**0.8**	**0.9**
BERT-based	Precision	0.86	0.88	0.88	0.89	**0.90**
BiLSTM	Recall	0.91	0.88	0.87	0.86	**0.85**
FastText-based	Precision	**0.87**	0.91	0.94	0.96	0.96
Random Forest	Recall	**0.56**	0.49	0.37	0.23	0.11

Concerning the confidence impact of both models on their classification accuracy, Table II reports the achieved results. For what concerns the model by Tian *et al.* (BERT-based BiLSTM in Table II), using the default settings (*i.e.,* 0.5 threshold) we observe a precision of 85% and a recall of 91% in identifying commit messages featuring a rationale (explicit or implicit). When considering as commits featuring a rationale those classified as such with higher confidence, the precision of the classification increases by paying a price in terms of recall. In particular, we have considered thresholds of 0.6, 0.7, 0.8, and 0.9 (*e.g.,* in the latter case, a commit is classified as featuring a rationale only if the confidence of the model is higher than 0.9). While the results are available in our replication package [16], we can highlight that using 0.9 as a confidence threshold results in a precision of 90% while still keeping rather high values of recall (85%). This is the configuration we have employed in our study to select the commits to provide as input to the random forest.

Concerning the confidence on the classification provided by the Random Forest, as it can be seen in Table II, the slight gain in precision when increasing the confidence threshold has an excessive cost in terms of recall, thus suggesting the usage of the default 0.5 confidence threshold.

Out of the 665 commits identified as featuring an explicit rationale, we classified 540 of them (81%) as correct, *e.g.,* "*remove unneeded check for final methods, in most cases the modifier needs to be removed anyway to implement it*" (all data publicly available in [16]). This result is inline with what expected considering the previously reported evaluations of the two components in isolation.

Indeed, by multiplying the errors, we expected a precision around 78% (*i.e.,* 90%×87%). The achieved results support the usage of the two techniques as the first step for the building of a large scale dataset featuring $\langle C_{diff}, \{C_r\}\rangle$ pairs. We discuss in the next section how, starting from the identified commits, we extract from the commit messages only the sentences documenting the rationale (*i.e.,* $\{C_r\}$).

Answer to RQ$_1$: Our Random Forest classifier can discriminate commit messages featuring an *explicit* rationale with a precision of 87% and a recall of 56%. When queued to the approach by Tian *et al.*, the whole pipeline can mine commits featuring an explicit rationale with a precision of 81%.

IV. Extracting Sentences Featuring the Rationale

We detail the "*Rationale Extractor*" (see Fig. 1) we use to convert the 96,034 $\langle C_{diff}, C_m \rangle$ pairs into $\langle C_{diff}, \{C_r\} \rangle$ pairs, with $\{C_r\}$ being the rationale in C_m. The approach is described in Section IV-A, while its empirical evaluation is presented in Section IV-B.

A. Approach Description

We started collecting data useful for approaching this task during the previously described manual inspection of 665 commits likely to feature an explicit rationale. In particular, besides classifying them as featuring or not an explicit rationale, we also extracted from the true positives (*i.e.,* those actually featuring an explicit rationale) the set $\{C_r\}$ of sentences documenting the rationale. We followed the exact same procedure previously described: the first two authors independently performed this task on all 665 commits. In this case, there was agreement on just 9% of the instances (60). Such a low agreement is explained by the nature of the performed task. Indeed, when extracting the sentences documenting the rationale in the commit message, we considered even minor differences in the extracted sentences as a conflict. For example, two sentences being identical but for a period terminating only one of the two were considered as a conflict. What we observed is that, in most cases, the rationale extracted from one author was a subset of the rationale extracted from the other author. Conflicting cases have been distributed among the three other authors to solve them. In the end, out of the 665 inspected commits: 125, as explained in Section III-B2, do not contain an explicit rationale, 450 feature a single sentence documenting the rationale, and 90 feature two or three sentences documenting the rationale.

To the set of 665 commits, we added 271 commits for which Tian *et al.* manually identified the sentences featuring the explicit rationale in the associated commit message [2] (similarly to what we did). Out of the overall 936 instances, 137 did not contain an explicit rationale (125 from our dataset plus 12 from the dataset by Tian *et al.*), while 799 feature at least one sentence explaining the rationale. We set apart 187 instances (20% of the total) as test set, leaving the remaining 749 to be used as the training dataset.

The built dataset has been used to train and experiment with the Meta AI's LLaMA [14], a set of foundational language models with parameter sizes ranging from 7B to 65B. These models have been pre-trained on trillions of textual tokens, achieving state-of-the-art performance in several NLP tasks. For example, LLaMA-13B (*i.e.,* 13 Billions of trainable parameters) exhibits superior performance compared to GPT-3 (175B) across various benchmarks [14]. We leverage LLaMA-7B due to the limited availability of hardware resources.

We specialize LLaMA for the task of interest using two different approaches designed to work with few training instances (given the 749 training instances at our disposal).

The first approach is what is called efficient fine-tuning [49], in which most of the model's parameters are frozen, and just a minority of them are fine-tuned to specialize the model for the task of interest. In our case, we used the LLaMA-Adapter [50], which only re-trains 1.2M parameters (out of the 7B) during fine-tuning. Each training instance is represented by a pair $\langle Input, ExpectedOutput \rangle$:
Input: "*[ASV-1778] Using a null parent to avoid having bug where there is a parent while trying to add view*".
Expected Output: "*to avoid having bug where there is a parent while trying to add view*".

The second approach is the few-shot learning [51], in which we exploited the pre-trained LLaMA in inference mode by asking it to generate a prediction for each instance in the test set via a specific prompt featuring n examples of the task to perform. The parameter n indicates the "few-shots" provided to the model. For example, in a 3-shot setting, 3 random examples from our training set are provided in the prompt asking the model to extract the rationale from a commit message in the test set. The 3 examples are the same for all instances in the test set (example of prompt in the replication package [16]).

Note that our training and test sets also feature instances without a rationale to extract (overall, 137 instances). These are instances which, in a real usage scenario, would be returned by the previous two components of our pipeline since we know that we can expect ∼19% of commit messages identified as likely to feature an explicit rationale to actually not containing it. For these cases, LLaMA is instructed during both strategies to return an empty string.

B. Empirical Evaluation

We present the design (Section IV-B1) and results (Section IV-B2) of the study conducted to assess the performance of our approach for the extraction of the commit rationale.

1) Study Design: The formulated research question is:

RQ_2: *To what extent is LLaMA able to extract the sentences documenting the rationale from commit messages?*

We assess the performance of the two strategies, *i.e.,* fine-tuning and few-shot, on our test set featuring 187 commit messages for which sentences describing the rationale (if any) have been manually extracted. For the efficient fine-tuning strategy, we run the fine-tuning on our training set for up to 10 epochs, reporting the performance achieved on the test set after 1, 3, 5, and 10 epochs. Concerning the few-shot learning, we show the performance of LLaMA in the 0-shot, 1-shot, 3-shot, 5-shot, 10-shot, 15-shot, and 20-shot learning scenarios, performed using one A100 GPU with 40GB of RAM provided by Google Colab.

We use BLEU (Bilingual Evaluation Understudy) score [52] as evaluation metric, commonly used to assess the similarity between generated and reference text, measuring the presence of candidate n-grams in the reference text.

TABLE III
RQ$_2$: Results of LLaMA for rationale extraction.

Strategy		BLEU score		
		Sentence	Corpus	Hyp$_{len}$
	# Epochs			
	1	0.064	0.189	3,087
Efficient	3	0.175	0.268	6,359
Fine-tuning	5	0.189	0.269	6,763
	10	0.202	0.269	7,421
	# Shot			
	0	0.015	0.028	4,812
	1	0.161	0.219	3,988
	3	0.214	0.360	3,471
Few-shot	5	0.263	0.408	3,330
	10	0.29	0.404	3,758
	15	**0.325**	**0.386**	**4,533**
	20	0.323	0.389	4,727

This computation results in a score ranging from 0 (completely dissimilar sequences) to 1 (identical sequences). We adopt the BLEU-4 variant implemented in the SacreBLEU library [53], considering the overlap in terms of 4-grams between the generated and the reference text.

We evaluate the sentence-level BLEU score, the corpus-level BLEU score, and the hypothesis length. The first is computed as the mean of the BLEU score obtained by individually contrasting each model output to the relative target in the test set. The corpus-level score, instead, is obtained by calculating the BLEU between the output corpus, *i.e.,* the concatenation of LLaMA output for all the instances of the test set, and the target corpus, *i.e.,* the concatenation of all targets in the test set. Finally, the length of the hypothesis (Hyp$_{len}$) is the total length of the LLaMA output. The latter measures the quantity of text generated by the approach and can be used to compare it with the expected target length. We also analyze the best-performing configuration (that, as we will discuss later, is the 15-shot learning) by running it on 400 previously unseen commit messages (*i.e.,* messages not featured in the 15 examples used for the few-shot learning) randomly collected from the set of 96,034 $\langle C_{diff}, C_m \rangle$ pairs likely to feature an explicit rationale. The first author manually inspected these 400 instances, looking at the original commit message and the rationale extracted from it by LLaMA. The goal of the inspection was to first flag instances lacking an explicit rationale in the commit message, which are false positives originating from the earlier stages of our process. In a real scenario, these instances represent failure cases for our pipeline to build a dataset of commits with their rationale. Thus, we consider them as such. For all remaining commits, the first author classified the extracted rationale as correct (*i.e.,* LLaMA correctly identifies the rationale in the commit message) or wrong. For each instance, he/she indicated whether LLaMA extracts all sentences documenting the rationale or only a subset of them. All instances classified as meaningful by the first author got a double check by the second author. After discussing 6 cases of disagreement, they agreed on the final classification.

2) Results Discussion: Table III shows the results obtained by LLaMA with the two strategies. Besides the BLEU score variants (sentence and corpus), the Hyp$_{len}$ reports the amount of tokens generated by the model for the entire test set, which should be compared to the target (expected) length of 3,342 tokens.

With efficient fine-tuning, the BLEU scores show a gradual increase as the number of training epochs increases. However, the improvement becomes marginal when transitioning from 5 to 10 epochs. Nevertheless, the Hyp$_{len}$ metric reveals that LLaMA generates an extensive amount of text which is not part of the target; indeed, it generates more than twice the needed text after 5 and 10 epochs (*e.g.,* 7,421 tokens against the expected 3,342 at 10 epochs). The situation is different with few-shot learning (bottom of Table III). Without any example in the prompt, *i.e.,* zero-shot, the BLEU score values are very low. However, introducing just a single example as input (*i.e.,* 1-shot) significantly improves performance (*e.g.,* from 2.8% to 21.9% in corpus BLEU). As the number of shots increases, the performance further improves, reaching a plateau at 15-shots. This setting is the one we adopt since it achieved the highest sentence BLEU and almost identical corpus BLEU when compared to the 20-shots. Notably, the Hyp$_{len}$ values are also closer to the expected target.

Regarding the qualitative analysis of the 400 instances, LLaMA does not generate an output for 5 of them (1%), despite these instances having a rationale in the commit message. Among the remaining 395, 47 had no rationale in the input commit message (12%), with LLaMA still trying to extract information from them. Finally, 296 commits are labeled as meaningful (75%). Out of these, 186 featured all rationales documented in the commit message, with 123 of them featuring only one rationale.

Answer to RQ$_2$: LLaMA can extract meaningful sentences documenting the rationale for code changes for 75% of commit messages. In 48% of cases, the extracted rationale is comprehensive, while in the remaining 27% is only partial.

V. Generating the rationale for commit diffs

Building on top of the previously described steps, we detail our generative approach to document the rationale for commit diffs (Section V-A) and its empirical evaluation (Section V-B).

A. Approach Description

Our starting point to create the dataset needed for training a generative model for rationale documentation are the 96,034 $\langle C_{diff}, C_m \rangle$ pairs obtained through the "*Rationale filtering pipeline*" (see Fig. 1).

While these are commits featuring an explicit rationale, we have to run on them the *Rationale Extractor* (*i.e.,* LLaMA in its best configuration, namely the 15-shot prompting) to create the pairs $\langle C_{diff}, \{C_r\} \rangle$.

Being LLaMA a large language model featuring 7B parameters, we rented 2 RTX A6000 GPUs featuring 48GB of RAM each, enabling the extraction of the rationale from 57,452 commits (60%), since the GPUs at our disposal were not sufficient to run it in inference mode (*i.e.,* to generate the predictions).

Since the goal is to use this data to train a generative model, two filters were applied to remove noisy instances before starting the learning process. First, we discarded instances for which LLaMA extracted a rationale whose length, in terms of tokens, was less than 5 or more than 50, resulting in a set of 49,270 instances (86%). The goal of this filtering is to remove meaningless rationales (*i.e.,* those featuring few words), and to exclude long rationales that could hinder the learning process of the generative model. Second, we removed extracted rationales starting with one of the following constructs: "git-svn-id:[url]", "Merged [branch] in [other-branch]", "Push [branch]", and "Signed-off-by:[author]". Indeed, we observed these as recurring patterns in the wrong rationales extracted by LLaMA (3,325 instances). At the end, we obtained 45,945 $\langle C_{diff}, \{C_r\}\rangle$ pairs, which have been split into training (36,756, 80%), evaluation (4,595, 10%), and test set (4,594, 10%).

We used the training set to fine-tune the CodeT5+ model [15], which has been pre-trained on over 8M functions written in eight different programming languages, with various objectives, such as contrastive learning, text-code matching, and causal language modeling. For our study, we employed the CodeT5+$_{base}$ variant, featuring 220M trainable parameters. For fine-tuning, we used the AdamW optimizer [54] with a learning rate of 5e-5, a batch size of 16 and fixing the input length to 512 tokens and the output length to 128 tokens (longer sequences are truncated). To prevent overfitting we used early stopping by saving model checkpoints every 2k training steps and assessing performance at each checkpoint, stopping training if no improvement is noted compared to 5 checkpoints (*i.e.,* 10k training steps) earlier. The evaluation of the checkpoints is conducted using the BLEU-4 [53], with the best-performing checkpoint identified after 30k training steps (∼13 epochs). After training the model, we used the beam search strategy [55] to generate predictions, enabling the synthesis of K candidate solutions (rationales) for a single commit diff, and we conducted experiments using various values of K ranging from 1 to 10.

B. Empirical Evaluation

We present the results of our empirical assessment of CodeT5+ for commits' rationale generation.

1) Study Design: The formulated research question is:

RQ$_3$: *To what extent is CodeT5+ able to automatically generate the rationale for code changes?*

RQ$_3$ represents the final evaluation of our complete pipeline. Indeed, the training set for CodeT5+ has been built by relying on the components we defined to mine a large dataset of $\langle C_{diff}, \{C_r\}\rangle$ pairs.

To address RQ$_3$, we run CodeT5+ against our test set of 4,594 instances (Section V-A). We compute (i) the BLEU-4 score of the predictions when contrasted towards the reference rationale, and (ii) the percentage of correct predictions, namely cases in which the predicted rationale is identical to the reference one.

We investigate whether the performance of CodeT5+ is influenced by its confidence in the prediction. CodeT5+ provides a *score* for every prediction, which indicates the log-likelihood of that prediction. To illustrate, a log-likelihood of -0.1 corresponds to a likelihood of 0.90 ($ln(x) = -0.1 \implies x = 0.90$). The likelihood represents the model's level of confidence in the prediction, ranging from 0.00 to 1.00 (higher values indicate greater confidence). We categorize predictions into one of ten buckets based on their confidence. The ten buckets represent all predictions having confidence between 0.0 and 0.1, 0.1 and 0.2, ..., 0.9 and 1.0. We show how the correct and wrong predictions, as well as the BLEU scores, are distributed across the 10 buckets. This can provide insights into the actual usability of our approach: even assuming the overall performance on the test set to not be satisfactory, if the high-confidence predictions are valuable, developers may agree to receive recommendations (*i.e.,* an automatically generated rationale) when the model is highly confident, reducing the burden of low-quality recommendations.

While the above-described evaluation is fully automated and can provide us with preliminary indications about the quality of the generated rationale, it has two important limitations. First, the test dataset has been built using our automated pipeline, which we know includes a percentage of wrong target rationales. Second, the wrong predictions (*i.e.,* cases in which the generated output is not identical to the reference one) might still be valuable for developers, as possibly being validly generated rationales just using a different wording as compared to the reference ones.

We conducted the following evaluation steps. First, two authors manually checked the correct predictions (*i.e.,* generated rationale matching the target) to verify the accuracy of the target rationale extracted by LLaMA. This check gives us confidence about the reported percentage of correctly generated rationales.

Second, we selected a statistically significant sample (95% confidence ±5%) of 363 incorrect high-confidence predictions to assess reliability. The first author labeled them as clear (the generated rationale can be understood) or unclear, with 324 clear instances further labeled by the first two authors as partially equivalent to the reference rationale (*i.e.,* conveying the same information but with additional details in the target). Conflicts in 47 instances (14%) were resolved through discussion, with a Cohen's kappa score of 0.69 [56], indicating substantial agreement.

Third, we computed the token-level Levenshtein distance [57] between incorrect predictions and target rationales, reporting the number of cases where the distance was at most one or two tokens.

TABLE IV
RQ$_3$: RESULTS OF CODET5+ FOR RATIONALE GENERATION

Beam Size	Performance Metric	
	Correct Predictions	BLEU score
1	1.91% (88/4,594)	0.06
3	1.93% (88/4,594)	0.063
5	2.04% (94/4,594)	0.064
10	2.09% (96/4,594)	0.065

These are wrong predictions which, however, with very low effort might be adjusted to obtain a meaningful rationale. Finally, we conduct an experiment aimed at evaluating the quality of the generated rationales. We selected 100 of the 324 commit messages that we classified as having a "clear" rationale and asked 20 participants having experience in Java to assess (i) the clarity of the generated rationales as we previously did (with the goal of factoring out a possible bias we as authors could have in our manual validation), and (ii) the semantic equivalence of the generated rationale with respect to the target rationale. The 100 messages were split into 10 surveys, each featuring 10 of them. Each participant was assigned to one survey, ensuring that each commit message was inspected by two independent evaluators. Participants had an average experience of 3.61 on a scale from 1 ("Very low experience") to 5 ("Very high experience"). For each commit message, participants expressed their level of agreement on a scale from 1 ("Totally Disagree") to 5 ("Totally Agree") to the following two statements: "The provided rationale is clear" (focus on the generated rationale) and "The two rationales are semantically equivalent" (both generated and target rationales are shown).

2) Results Discussion: From the Table IV reporting the results using different beam sizes, it is clear that CodeT5+ struggles to generate the rationale, since the percentage of correct predictions is ∼2%, independently from beam size.

These results are not surprising for two reasons. First, the challenging task we are addressing, that can be compared to generative tasks such as bug-fixing [58] or automated code review [59] in which similar success rates have been reported. Second, the strict definition of "correct prediction" that we are adopting: a generated rationale differing from the expected target of one term is considered wrong.

Through the analysis of the token-level Levenshtein distance [57] , we found that 40 (wrong) predictions were "1 token far" from the target rationale, usually a space or a period added at the end of the rationale in the prediction. These can basically be considered as correct instances, pushing a bit the correctness to ∼3%.)ther 14 predictions are 2-tokens apart from the target.

The manual analysis of wrong predictions provides insights about their potential usefulness. Out of the 363 inspected predictions, 39 (11%) have been classified as "unclear" (*i.e.,* it is not possible to judge its quality). For the remaining 324 instances, the two authors agreed on classifying them as partially equivalent to the rationale written by developers in 111 (34%) of cases.

One example is discussed in the following, while the set of validated rationales is publicly available [16].

Target: *Since the facilities can have any type of activity, this has to be true also for the knowledge of a person.*

Prediction: *It is allowed to add any kind of activity type to a facility for more flexibility against given input data.*

The rationale for code changes resides in the choice of allowing to add any activity to facilities, and that must, thus, apply to a person's knowledge. CodeT5+ partially captures the rationale behind the code change (allowing to add any activity to a facility) and adds the explanation that the change aims to enhance flexibility with respect to input data. This is classified as "partially equivalent".

The 363 wrong predictions we manually inspected were sampled from the high-confidence predictions, so these are basically the best-case scenario for our model. Still, ∼30% of meaningful predictions represent an encouraging result. The usefulness of the high-confidence prediction is supported by the analysis in which we relate the prediction quality and their confidence. 92% of wrong predictions fall in the first confidence bucket (*i.e.,* confidence lower than 0.1). These are instances for which the model struggles, likely because nothing "similar", in terms of code diff, has been seen in the training set. With the increase of confidence, the percentage of wrong predictions decreases, until reaching 0.6% in the highest confidence bucket (*i.e.,* confidence higher than 0.9). The latter features 41% of the 88 correct predictions generated by the model. In general, when looking at predictions having a confidence higher than 0.7, they feature 101 wrong (2% of the overall wrong) and 77 correct (88% of the overall correct) predictions.

The distribution of the BLEU score for predictions having different confidence is reported in Fig. 2: the higher quality of high-confidence predictions is confirmed, with the median BLEU score in the highest confidence bucket being 1.0 (*i.e.,* a correct prediction). In general, an upward trend is visible with the increase in the model's confidence.

The aggregated results of the surveys, shown in the Fig. 3, highlight the relationship between the confidence scores of generated commit messages and their perceived quality based on participant evaluations.

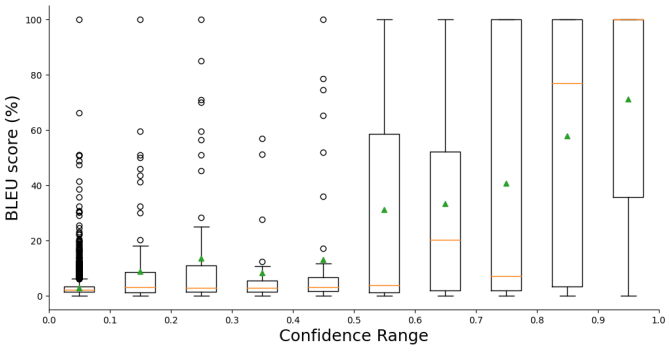

Fig. 2. BLEU scores in the different confidence buckets.

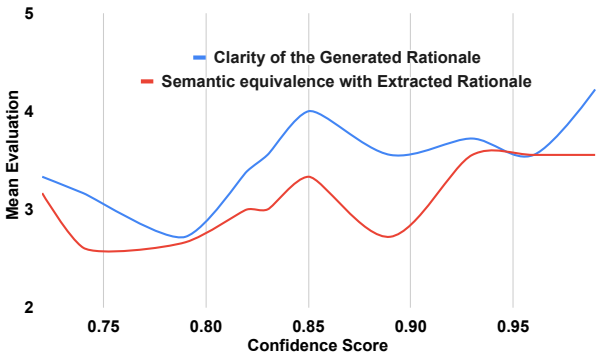

Fig. 3. Evaluations about clarity and meaning of the rationale.

For the clarity of the generated rationale, the mean was 3.56, indicating a moderately positive reception among participants regarding the messages' ability to clearly explain the motivation behind code changes. For semantic equivalence, the mean average evaluation was slightly lower, at 3.10, suggesting that while participants found some alignment between the generated and extracted rationales, there is substantial room for improvement in ensuring semantic consistency. The decline in ratings with decreasing confidence scores confirms that the model's confidence can be a useful indicator of the quality of generated messages.

In the end, most of the predictions fall in the first (lowest) confidence bucket. This indicates that the model exhibits a low confidence, possibly due to the limited size of the training set. When the confidence is high, the quality of the predictions increases. This is supported by the distribution of the correct predictions/BLEU score, as well as by our manual inspection of wrong predictions having high confidence, furthermore supported by evaluations given by Java experts involved for the surveys.

Answer to RQ$_3$: CodeT5+ generate a correct rationale for ∼3% of commits. On predictions with high confidence, manual analysis show that ∼34% of the generated rationales are partially correct.

VI. Threats to validity

Construct validity threats concern the relationship between theory and observation and are mostly related to the measurements we performed. We used standard evaluation metrics both when dealing with classifiers (*i.e.,* recall and precision employed to assess the *Rationale filtering pipeline*) as well as when evaluating generative models such as LLaMA for rationale extraction and CodeT5+ for rationale generation (BLEU-4 and percentage of correct predictions). These metrics are aligned to what similar works in the literature used (*e.g.,* [2], [5], [29], [32], [60]).

Internal validity threats concern factors within our study that can influence the findings. Subjectivity in manual analyses may be present, though all inspections were performed independently by two authors, except for labeling the 1,157 commit messages (Section III-A), which was done by the first author. These messages had already been classified by Tian *et al.* [2] as containing a rationale, and we further refined the classification. Additionally, the configurations of the machine learning models and preprocessing choices may have influenced the results, requiring further investigation (*e.g.,* hyperparameter tuning).

For LLaMA, we tested various prompts (*e.g.,* explaining the task) but found no improvement over the simpler $\langle Input, ExpectedOutput \rangle$ prompt (Section IV-A). We also experimented with RoBERTa [61] and CommitBART [62], fine-tuning them for rationale extraction, but they performed worse than LLaMA [16]. Lastly, the training set for CodeT5+ (∼36k instances) may be insufficient, and we plan to experiment with larger datasets in future work.

External validity threats relate to the generalization of findings. We targeted commits from Java projects and having specific characteristics (*e.g.,* impacting a single diff hunk). These choices affect the generalizability of findings, which is not claimed out of this commits population.

VII. Conclusion

Existing techniques for automated commit message generation mainly describe *what* has changed rather than *why*. In this work, we tackle the challenging problem of explaining the rationale for code changes, which requires building a large-scale training set documenting commit rationales. We proposed a pipeline to automatically create such a dataset, showing its ability to both (i) select commits documenting the rationale (∼81% of precision), and (ii) create the needed $\langle C_{diff}, \{C_r\} \rangle$ pairs, with $\{C_r\}$ representing sentences expressing the rationale for the code change (C_{diff}) — ∼75% of meaningful pairs. We ran such a pipeline in a collection of Java projects, creating a dataset to train a CodeT5+ [15] model for the task of rationale generation. We observed CodeT5+ struggling in such a task, despite encouraging results obtained when the model has high confidence in the generated predictions. Such (partially) negative results call for additional research on this problem.

Future work will focus on improving our pipeline to mine commits with their rationale, trying to further push its precision, or explore alternative approaches to using LLMs. This would allow to build higher quality training sets. Indeed, our findings indicate that the current approach of using generative models for generating the rationale is limited by the quality and precision of the training data. This outcome highlights the importance of not only focusing on the model architecture but also ensuring that the dataset is highly curated and informative. Also, we want to experiment with the impact of larger training datasets on the performance of the generative model.

References

[1] S. Weber, *The success of open source*. USA: Harvard University Press, 2004.

[2] 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 (ICSE)*, 2022, p. 2389–2401.

[3] T. Vu, T.-D. Do, and H. D. Vo, "Context-encoded code change representation for automated commit message generation," *arXiv preprint arXiv:2306.14418*, 2023.

[4] 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 (ICSE)*, 2022, pp. 970–981.

[5] 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, pp. 1800–1817, 2019.

[6] 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 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022, pp. 5520–5530.

[7] 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 International Conference on Automated Software Engineering (ASE)*, 2018, p. 373–384.

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

[9] S. Xu, Y. Yao, F. Xu, T. Gu, H. Tong, and J. Lu, "Commit message generation for source code changes," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, 2019, pp. 3975–3981.

[10] S. Jiang and C. McMillan, "Towards automatic generation of short summaries of commits," in *Proceedings of the 25th International Conference on Program Comprehension (ICPC)*, 2017, pp. 320–323.

[11] K. Al Safwan, M. Elarnaoty, and F. Servant, "Developers' need for the rationale of code commits: An in-breadth and in-depth study," *Journal of Systems and Software*, vol. 189, p. 111320, 2022.

[12] M. Codoban, S. S. Ragavan, D. Dig, and B. Bailey, "Software history under the lens: A study on why and how developers examine it," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 1–10.

[13] Y. Tao, Y. Dang, T. Xie, D. Zhang, and S. Kim, "How do software engineers understand code changes? an exploratory study in industry," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ser. FSE '12. New York, NY, USA: Association for Computing Machinery, 2012.

[14] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[15] Y. Wang, H. Le, A. D. Gotmare, N. D. Bui, J. Li, and S. C. Hoi, "CodeT5+: Open code large language models for code understanding and generation," *arXiv preprint arXiv:2305.07922*, 2023.

[16] F. Casillo, A. Mastropaolo, G. Bavota, V. Deufemia, and C. Gravino, "Towards generating the rationale for code changes," 2024. [Online]. Available: https://doi.org/10.5281/zenodo.8187207

[17] A. Kleebaum, B. Paech, J. O. Johanssen, and B. Brügge, "Continuous rationale identification in issue tracking and version control systems," in *Joint Proceedings of REFSQ 2021 Workshops*, ser. CEUR Workshop Proceedings, vol. 2857, 2021.

[18] M. Dhaouadi, B. J. Oakes, and M. Famelis, "End-to-end rationale reconstruction," *Proceedings of the 37th International Conference on Automated Software Engineering (ASE)*, 2022.

[19] P. N. Sharma, B. T. R. Savarimuthu, and N. Stanger, "Extracting rationale for open source software development decisions — a study of python email archives," in *Proceedings of the 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 1008–1019.

[20] B. Rogers, J. Gung, Y. Qiao, and J. E. Burge, "Exploring techniques for rationale extraction from existing documents," in *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, 2012, p. 1313–1316.

[21] R. P. Buse and W. R. Weimer, "Automatically documenting program changes," in *Proceedings of the 25th International Conference on Automated Software Engineering (ASE)*, 2010, pp. 33–42.

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

[23] M. Linares-Vásquez, L. F. Cortés-Coy, J. Aponte, and D. Poshyvanyk, "Changescribe: A tool for automatically generating commit messages," in *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, vol. 2, 2015, pp. 709–712.

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

[25] Y. Huang, Q. Zheng, X. Chen, Y. Xiong, Z. Liu, and X. Luo, "Mining version control system for automatically generating commit comment," in *Proceedings of the 11th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2017, pp. 414–423.

[26] R. Salakhutdinov, "Learning deep generative models," *Annual Review of Statistics and Its Application*, vol. 2, pp. 361–385, 2015.

[27] P. Loyola, E. Marrese-Taylor, J. Balazs, Y. Matsuo, and F. Satoh, "Content aware source code change description generation," in *Proceedings of the 11th International Conference on Natural Language Generation (INLG)*, 2018, pp. 119–128.

[28] P. Loyola, E. Marrese-Taylor, and Y. Matsuo, "A neural architecture for generating natural language descriptions from source code changes," *arXiv preprint arXiv:1704.04856*, 2017.

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

[30] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," *arXiv preprint arXiv:1704.04368*, 2017.

[31] 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, pp. 1–30, 2021.

[32] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[33] 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, 2021.

[34] C. Sammut and G. I. Webb, Eds., *TF–IDF*. Boston, MA: Springer US, 2010, pp. 986–987.

[35] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*, 2016.

[36] J. Devlin, M. 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 (NAACL-HLT), 2019, pp. 4171–4186.

[37] 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.

[38] A. Mastropaolo, L. Pascarella, and G. Bavota, "Using deep learning to generate complete log statements," *arXiv preprint arXiv:2201.04837*, 2022.

[39] W. Tao, Y. Wang, E. Shi, L. Du, S. Han, H. Zhang, D. Zhang, and W. Zhang, "A large-scale empirical study of commit message generation: models, datasets and evaluation," *Empirical Software Engineering*, vol. 27, pp. 1–43, 2022.

[40] O. Dabic, E. Aghajani, and G. Bavota, "Sampling projects in github for MSR studies," in *Proceedings of the 18th International Conference on Mining Software Repositories (MSR)*, 2021, pp. 560–564.

[41] D. Spadini, M. Aniche, and A. Bacchelli, "PyDriller: Python framework for mining software repositories," in *Proceedings of the 26th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2018, pp. 908–911.

[42] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext.zip: Compressing text classification models," *arXiv preprint arXiv:1612.03651*, 2016.

[43] I. Montani, M. Honnibal, M. Honnibal, A. Boyd, S. V. Landeghem, H. Peters, P. O. McCann, jim geovedi, J. O'Regan, M. Samsonov, D. de Kok, G. Orosz, M. Blättermann, D. Altinok, R. Mitsch, M. Kannan, S. L. Kristiansen, Edward, L. Miranda, R. Bournhonesque, P. Baumgartner, R. Hudson, E. Bot, Roman, L. Fiedler, R. Daniels, kadarakos, W. Phatthiyaphaibun, and Schero1994, "explosion/spaCy: v3.5.4: Bug fixes for overrides with registered functions and sourced components with listeners," Jun. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.8091979

[44] N. Shuyo, "Language detection library for java," 2010. [Online]. Available: http://code.google.com/p/language-detection/

[45] L. Erlenhov, F. Gomes de Oliveira Neto, R. Scandariato, and P. Leitner, "Current and future bots in software development," in *Proceedings of the 1st International Workshop on Bots in Software Engineering (BotSE)*, 2019, pp. 7–11.

[46] S. Amreen, A. Mockus, C. Bogart, Y. Zhang, and R. L. Zaretzki, "Alfaa: Active learning fingerprint based anti-aliasing for correcting developer identity errors in version control systems," *Empirical Software Engineering*, vol. 25, pp. 1136–1167, 2019.

[47] T. Dey, S. Mousavi, E. Ponce, T. Fry, B. Vasilescu, A. Filippova, and A. Mockus, "Detecting and characterizing bots that commit code," in *Proceedings of the 17th International Conference on Mining Software Repositories (MSR)*, 2020, p. 209–219.

[48] T. Dey, B. Vasilescu, and A. Mockus, "An exploratory study of bot commits," in *Proceedings of the 42nd International Conference on Software Engineering Workshops (ICSEW)*, 2020, p. 61–65.

[49] S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, and S. Paul, "Peft: State-of-the-art parameter-efficient fine-tuning methods," https://github.com/huggingface/peft, 2022.

[50] R. Zhang, J. Han, C. Liu, P. Gao, A. Zhou, X. Hu, S. Yan, P. Lu, H. Li, and Y. Qiao, "Llama-adapter: Efficient fine-tuning of language models with zero-init attention," *arXiv preprint arXiv:2303.16199*, 2023.

[51] Y. Wang, Q. Yao, J. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," *arXiv preprint arXiv:1904.05046*, 2020.

[52] 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 (ACL)*, 2002, pp. 311–318.

[53] M. Post, "A call for clarity in reporting bleu scores," *arXiv preprint arXiv:1804.08771*, 2018.

[54] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

[55] M. Freitag and Y. Al-Onaizan, "Beam search strategies for neural machine translation," *arXiv preprint arXiv:1702.01806*, 2017.

[56] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[57] V. I. Levenshtein *et al.*, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8. Soviet Union, 1966, pp. 707–710.

[58] M. Tufano, C. Watson, G. Bavota, M. Di Penta, M. White, and D. Poshyvanyk, "An empirical study on learning bug-fixing patches in the wild via neural machine translation," *ACM Trans. Softw. Eng. Methodol.*, vol. 28, no. 4, pp. 19:1–19:29, 2019.

[59] R. Tufano, S. Masiero, A. Mastropaolo, L. Pascarella, D. Poshyvanyk, and G. Bavota, "Using pre-trained models to boost code review automation," *arXiv preprint arXiv:2201.06850*, 2022.

[60] A. Mastropaolo, S. Scalabrino, N. Cooper, D. N. Palacio, D. Poshyvanyk, R. Oliveto, and G. Bavota, "Studying the usage of text-to-text transfer transformer to support code-related tasks," in *Proceedings of the 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 336–347.

[61] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[62] S. Liu, Y. Li, and Y. Liu, "Commitbart: A large pre-trained model for GitHub commits," *arXiv preprint arXiv:2208.08100*, 2022.