Automatic Commit Message Generation: A Critical Review and Directions for Future Work

Yuxia Zhang, Zhiqing Qiu, Klaas-Jan Stol, Wenhui Zhu, Jiaxin Zhu, Yingchen Tian, and Hui Liu

Abstract—Commit messages are critical for code comprehension and software maintenance. Writing a high-quality message requires skill and effort. To support developers and reduce their effort on this task, several approaches have been proposed to automatically generate commit messages. Despite the promising performance reported, we have identified three significant and prevalent threats in these automated approaches: 1) the datasets used to train and evaluate these approaches contain a considerable amount of 'noise'; 2) current approaches only consider commits of a limited diff size; and 3) current approaches can only generate the subject of a commit message, not the message body. The first limitation may let the models 'learn' inappropriate messages in the training stage, and also lead to inflated performance results in their evaluation. The other two threats can considerably weaken the practical usability of these approaches. Further, with the rapid emergence of large language models (LLMs) that show superior performance in many software engineering tasks, it is worth asking: can LLMs address the challenge of long diffs and whole message generation? This article first reports the results of an empirical study to assess the impact of these three threats on the performance of the state-of-the-art auto generators of commit messages. We collected commit data of the Top 1,000 most-starred Java projects in GitHub and systematically removed noisy commits with bot-submitted and meaningless messages. We then compared the performance of four approaches representative of the state-of-the-art before and after the removal of noisy messages, or with different lengths of commit diffs. We also conducted a qualitative survey with developers to investigate their perspectives on simply generating message subjects. Finally, we evaluate the performance of two representative LLMs, namely UniXcoder and ChatGPT, in generating more practical commit messages. The results demonstrate that generating commit messages is of great practical value, considerable work is needed to mature the current state-of-the-art, and LLMs can be an avenue worth trying to address the current limitations. Our analyses provide insights for future work to achieve better performance in practice.

Index Terms—Commit-based software development, open collaboration, commit message generation, benchmark

✦

1 INTRODUCTION

VERSION control systems, such as Git, are widely used to track changes in software repositories. Developers commonly write a message for every set of changes they make when collaborating via version control systems. The combination of a code change and its corresponding text is called a *commit*. The change itself is called a *diff*, which records which lines of content (code, documentation, etc.) were added or removed [1]. The descriptive text is known as a *commit message*; a good commit message describes *what* was changed, and *why* the change was made [2]. Commit messages play an important role in understanding and communicating code changes and software maintenance. Thus, writing high-quality commit messages is important for any type of software project, whether it be commercial or open-source.

Unfortunately, commit messages are frequently empty or of low quality, so their potential value is lost [2]. A recent study showed that 44% of commit messages lack essential information when compared with the recognized expectation of commit messages [2]. Another study of 23,000

projects showed that about 14% of commit messages were empty [3]. These results are problematic, because commit messages play an important role in communicating to others what changes were made, and why. Without making this information explicit in the form of a commit message, this information remains tacit, and thus may disappear when contributors leave a team or community [4].

To support developers and reduce the effort involved in writing commit messages, multiple approaches have been proposed that can generate commit messages automatically [1, 5–10]. These approaches can be organized into four categories based on the way they work: rule-based, retrieval-based, learning-based, and hybrid approaches [10]. Rule-based approaches are proposed to generate messages by summarizing code changes such as method additions based on specific pre-defined rules or templates [5–7]. However, rule-based approaches are considered outdated because their generated messages tend to be very long and cannot generate the rationale for a code change [1]. Retrieval-based approaches leverage information retrieval techniques to select existing commit messages from similar code changes [1, 11]. More recently, several studies have applied neural machine translation algorithms to translate diffs into commit messages [10]. Hybrid approaches [12] generate commit messages by combining both information retrieval techniques with neural machine translation algorithms [9, 13].

Except for the now considered outdated rule-based approaches, more recent approaches have achieved promising results. However, we have identified three common threats in

- *Yuxia Zhang, ZhiQing Qiu, Wenhui Zhu, and Hui Liu are with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China. E-mail: yuxiazh@bit.edu.cn*
- *Klaas-Jan Stol is with Lero, the Science Foundation Ireland Research Centre for Software and the School of Computer Science and IT, University College Cork, Ireland.*
- *Jiaxin Zhu is with the Institute of Software, Chinese Academy of Sciences, Beijing, China.*
- *Yingchen Tian is with Tmall Technology Co., Zhejiang, China.*

these automated approaches to generating commit messages. The impact of these three threats on the performance and practical utility of these state-of-the-art commit message generation approaches has remained unstudied as of yet.

First, these approaches are trained and evaluated using datasets that contain 'noisy' data, including commits made by bots or messages that contain limited information [2, 14]. For example, consider the message "*Update dependency com.puppycrawl.tools: checkstyle to v10.2*" generated by `renovate-bot`; messages such as these follow simple patterns and do not convey much information. Using such low-quality messages to train and evaluate generators may let the models 'learn' inappropriate messages and also lead to inflated performance evaluation results. Thus, the first research question we address in this article is:

RQ1. To what extent do noisy messages affect the performance of existing automated approaches for commit message generation?

Second, due to the constraints of translation models, learning-based approaches can only deal with diffs with a length of no more than 100 tokens (200 for AST-based approaches [10, 15]). Retrieval-based approaches also have these constraints as they reuse the same dataset used for learning-based approaches [8]. However, there are no empirical studies that present the size distribution of commit messages, which means that cut-offs of 100 or 200 tokens are arbitrary, nor are there any studies that investigate the impact of such arbitrary cut-offs. Measuring the impact of this constraint on the performance of current approaches is important given that many code changes in production settings are considerably larger than 100 tokens. Thus, the second question we address in this article is:

RQ2. Given that a commit's diff is usually longer than 100 or 200 tokens, is the performance of existing approaches as promising when considering commits that capture longer code changes?

Third, both retrieval- and generation-based methods can only generate the first sentence of a commit message; in most cases, this is the 'subject' of a commit message. However, a commit message also has a 'body' (see Figure 1), in which developers can provide further details about the context, background, or reasons for making changes [16]. Compared to a message's subject, the body is usually considerably longer and contains more information. Whether simply generating commit message subjects can satisfy developers' practical needs remains an open question. To that end, we ask the following research question:

RQ3. Do developers think generating commit message subjects is useful?

The first three questions consider the current state-of-the-art approaches dedicated to commit message generation. At the time of writing, large language models (LLMs) have gained rapidly increasing popularity within the software engineering field, which demonstrates promising results for a wide variety of tasks [17]. It is therefore very timely to also consider the use of LLMs in the task of automatically generating commit messages. Thus, the fourth research question we address in this article is:
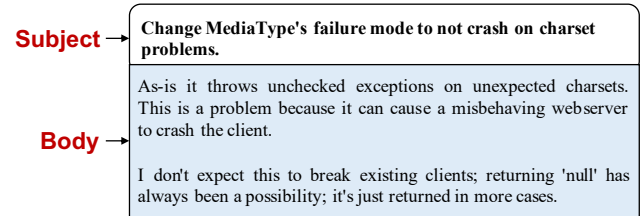
RQ4. How do LLMs perform in more real-world commit

Fig. 1: Example commit message from the Okhttp project (https://github.com/square/okhttp/)

message generation?

To answer these four questions, we conducted a multi-method study. We first selected three approaches for our evaluation, one from each of the three categories mentioned earlier: retrieval-based, combining both retrieval and learning (also noted as 'hybrid'), and learning-based approaches.[1] Besides, pre-trained models [e.g., 18, 19], which are designed for encoding diff, are also evaluated in generating commit messages, and achieved notable performance. Therefore, we also considered one state-of-the-art pre-trained model for our evaluation. Prior evaluations of the four selected approaches have used a dataset that was limited (i.e., commits with diff length of no more than 100 or 200 tokens); further, this dataset was created in 2017, over six years before the current study. For those reasons, we created a new dataset containing the commits of the top 1,000 most-starred Java repositories on GitHub. We then compared the performance of the four selected approaches before and after the removal of the commits submitted by bots, or that have uninformative messages. We find that the performance of the learning-based approach remains stable, while the performance of the other three approaches degrades considerably. Moreover, the performance of all four approaches degrades in our human evaluation. After an investigation of the distribution of diffs in the new dataset, we find that only 5% of the commits contain changes with no more than 100 tokens, and the median number of tokens in the new dataset is 632, considerably more than the limit of 100 that has been used in previous evaluations. When we replace the constraint of a diff length of 100 with larger sizes, the performance of the four selected approaches degrades significantly. We also surveyed developers to capture their perspectives on generating message subjects only. Most developers indicated that writing a message subject is difficult, and can be time-consuming because they need to summarize the change-related information into a short but meaningful sentence. Overall, the feedback from developers highlights the usefulness of generating commit messages automatically. In the last, we investigated the performance of large modes on commit message generation with long commit diffs. We chose UniXcoder and ChatGPT as representatives of LLMs and evaluated their performance in generating messages for diffs with larger lengths. The results revealed that although LLMs can be input larger diffs, their performance needs considerable improvement.

This study conducts a comprehensive reassessment of the state-of-the-art retrieval-, learning-, hybrid-, pre-trained-, and

---

1. We did not consider rule-based approaches because none of the state-of-the-art approaches rely on this approach.

LLM-based commit message generation, resulting in several practical and theoretical contributions to the literature. In sum, this article:

- Identifies and demonstrates the degree of performance degradation of state-of-the-art automated approaches when removing noisy data, and highlights the necessity of doing complementary data cleaning.
- Identifies and demonstrates the limited effectiveness of state-of-the-art commit message generating approaches to a larger and more realistic scenario.
- Demonstrates the practical value of generating commit message subjects from developers' perspectives.
- Demonstrates that commit messages generated by two popular LLMs (ChatGPT 3.5 and UniXcoder) are not satisfactory.
- Provides a new dataset for commit message generation, which can assist researchers in obtaining more reliable results in future evaluations.

Together, these results suggest a number of avenues for future research on automatically generating commit messages.

In the remainder of this article, we review related work in Sec. 2, outline our multi-method research approach in Sec. 3, and present the results of our study in Sec. 4. We discuss the implications for research and practice and the limitations of our study in Sec. 5. We conclude in Sec. 6.

2 RELATED WORK

We review approaches of automated commit message generation, followed by a discussion of commit message quality. We then consider bots that generate messages, the quality of the datasets that have been used to evaluate prior approaches, and also discuss the emergence of LLMs in software engineering.

2.1 Automated Commit Message Generation

Prior work on commit message generation can be categorized into rule-based or template-based, information retrieval-based, learning-based techniques, and hybrid methods [12]. Table 1 presents an overview.

A straightforward way to generate commit messages automatically is using predefined rules or templates [5–7, 20]. For example, DeltaDoc is based on symbolic execution to describe what a code change does [5]. Two approaches, ChangeScribe [6, 20] and AutoSumCC [7] summarize code changes, such as method additions, based on pre-defined rules or templates. However, as briefly mentioned, these rule-based approaches are only effective in generating text about *what* was changed and are not well able to provide the reasons behind code changes [1, 9], which is important information in a commit message [2].

A second category of approaches leverages information retrieval techniques to suggest commit messages from similar code changes [1, 21]. For example, the NNGen approach [1] leverages the Nearest Neighbor algorithm to generate commit messages from diffs. For a given new diff, NNGen first selects the most similar diff from the training data by calculating the cosine similarity and the BLEU score (see Sec. 3.3), then outputs the message from the selected diff as the generated one. Different from NNGen which treats each code change as a bag of words, LogGen (a retrieval-based approach [21]) uses code change vectors generated by CC2Vec as a novel form of diff. It is easy to imagine the challenges of these retrieval-based approaches when no similar diffs can be found in the corpus.

A third category of approaches uses learning-based techniques. These approaches present commit message generation as a translation problem and use neural machine translation models to translate code differences into commit messages [8–10, 13, 15, 22–25]. The most recently proposed approach within this category is FIRA [10]. Instead of directly feeding old-version and new-version code into translation models, FIRA represents code changes not as an abstract syntax tree, but as more fine-grained graphs by considering edit operations and sub-tokens in code changes. FIRA currently outperforms other techniques and could be considered the state-of-the-art of automatic commit message generation.

Two studies have combined information retrieval and neural machine translation techniques to generate commit messages [9, 13]. For example, CoRec [9] utilizes an information retrieval technique to address the word frequency issue, i.e., learning-based methods tend to generate high-frequency words but ignore low-frequency ones. As reported, CoRec achieves better performance than retrieval-based approaches [9]. There are also some pre-trained models designed for encoding diff, such as [18, 26], which are usually evaluated in the task of commit message generation.

Recently, Tao et al. [12] conducted an empirical study of the latest models of commit message generation. Different from our study, they primarily focused on how current approaches perform when evaluating them with alternative metrics, multiple programming languages, and dataset-splitting strategies. They found that these aspects have a considerable impact on the evaluation performance of existing models. Dong et al. explored the generated commit messages by learning-based approaches through the lens of patterns (i.e., frequent sequences) [27]. They found that the majority of generated messages belong to simple patterns [27]. Different from our study, Dong et al. focused on the output of these learning-based approaches, while this study explores the rationale of how we should design models to tackle the real problem of automatically generating commit messages.

2.2 Quality of Commit Messages

A well-crafted commit message is of great importance for understanding and communicating code changes [2]. Prior work suggests that the quality of commit messages needs improvements in most open source projects [2, 3, 28]. For example, Dyer et al. observed that approximately 14% of commit messages in over 23,000 open source projects were empty, and 66% of messages contained only a few words. On the basis of 11 syntactical measures, Chahal and Saini [28] constructed a model that can judge the quality of commit messages. In a recent study, Tian et al. [2] investigated the distribution, taxonomy, and classification of high-quality commit messages in five open source projects. They found that 44% of commit messages need further improvement. They also developed a taxonomy that describes how developers express 'why' and 'what' information. Of particular

TABLE 1
Overview of approaches for automated commit message generation

Category	Approach	Year	Evaluation	Diff filtering criteria[1]	Message filtering rules[1]
Rule-based approaches	DeltaDoc [5]	2010	Human	NA	NA
	ChangeScribe [20]	2014	Human	NA	NA
	AutoSumCC [7]	2016	Human	NA	NA
Retrieval-based approaches	NNGen [1]	2018	BLEU Human	○ diff length > 100 tokens ○ merge and rollback commits ○ larger than 1MB	○ message length > 30 tokens ○ message of non-Verb-Direct Object pattern • submitted by bot liferaycontinuous-integration • update changelog/gitignore/readme [md/file] • prepare version [version number] • bump version [version number] • modify dockerfile/makefile • update submodule
	LogGen [21]	2020	BLEU	○ diff length > 100 tokens ○ merge and rollback commits ○ larger than 1MB	○ message length > 30 tokens ○ message of non-Verb-Direct Object pattern ○ submitted by bot liferaycontinuous-integration ○ update changelog/gitignore/readme [md/file] ○ prepare version [version number] ○ bump version [version number] ○ modify dockerfile/makefile ○ update submodule
Learning-based approaches	CmtGen [8]	2017	BLEU Human	• diff length > 100 tokens • merge and rollback commits • larger than 1MB	• message length > 30 tokens • message of non-Verb-Direct Object pattern
	NMT [22]	2017	BLEU	○ diff length > 100 tokens	• message length > 20 tokens
	ContextNMT [23]	2018	BLEU METEOR Human	○ diff length > 100 tokens	○ message length > 20 tokens • certain repetitive patterns, such as the phrase merge pull request
	CODISUM [15]	2019	BLEU METEOR	• diff length > 200 tokens • diff files contain no .java files • duplicated diffs	○ message length > 20 tokens • contain less than three words
	PtrGNCMsg [24]	2019	BLEU ROUGE	○ diff length > 100 tokens ○ merge and rollback commits	○ message length > 30 tokens ○ message of non-Verb-Direct Object pattern
	CoreGen [25]	2021	BLEU ROUGE METEOR	○ diff length > 100 tokens ○ merge and rollback commits ○ larger than 1MB	○ message length > 30 tokens ○ message of non-Verb-Direct Object pattern ○ submitted by bot liferaycontinuous-integration ○ update changelog/gitignore/readme [md/file] ○ prepare version [version number] ○ bump version [version number] ○ modify dockerfile/makefile ○ update submodule
	FIRA [10]	2022	BLEU[2] ROUGE METEOR Human	○ diff files contain no .java files ○ duplicated diffs ○ diff length > 200 tokens	○ message length > 20 tokens ○ contain less than three words
Hybrid approaches	ATOM [13]	2020	BLEU ROUGE METEOR Human	○ diff files contain no .java files ○ merge or rollback commits • project initialization commits • fundamental updating	○ message length > 20 tokens • empty or contain non-ASCII messages
	CoRec [9]	2021	BLEU ROUGE METEOR Human	○ diff length > 100 tokens ○ merge and rollback commits ○ larger than 1MB	○ message length > 30 tokens ○ message of non-Verb-Direct Object pattern • keywords: changelog, gitignore, readme, release, version • ignore update [*] • modify dockerfile/makefile • update submodule(s)

[1] For each study cited in column 2, the table indicates whether that study proposed and applied new or additional data filtering conditions, marked as • in columns 5 (filtering criteria) and 6 (filtering rules); if a study applied previous filtering conditions, these are itemized with ○ .
[2] FIRA was evaluated using a variant of BLEU, called B-Norm BLEU.

relevance to the current study, Tian et al. [2] identified five types of trivial messages, i.e., single-word messages, submit-centered messages, scope-centered messages, redundant messages, and irrelevant messages. They also proposed several

classification models that can be used to identify commit messages with high (or low) quality. Based on Tian et al. [2], Li and Ahmed [29] designed a more advanced classification model, which particularly considers the contents of the links attached in commit messages. They also conducted an empirical study of the classified commits and found that the overall quality of the commit messages decreases over time, while developers believe they are writing better commit messages.

2.3 Bot-Generated Messages

A bot refers to a software agent that integrates its work with human tasks [30, 31], to support developers and increase their productivity by assisting in repetitive tasks [32]. Bots are now widely applied to support a wide variety of tasks in software engineering, including refactoring [33, 34], issue and pull request (PR) management [35, 36], automated generation of answer summaries [37]. For example, Dependabot[2] can automatically generate commits that fix security issues in a project's dependencies. It can also automatically generate messages to describe the changes. Figure 2 shows commit `#3ac5b0e`[3] in the `okhttp` project[4], which is generated by `renovate-bot`. This message tells that it bumps *com.puppycrawl.tools:checkstyle* to version 10.2. Commit messages generated by bots always have the same pattern which can be configured by rules.[5] There are two reasons why using such bot-generated commit messages in the evaluation of automatic commit message generators is of little use. First, these messages are already automatically produced by bots. Second, this type of message is simple and template- or pattern-driven, based on predefined 'structures', which can be easily learned by these automatic approaches and generated perfectly. Including these could lead to an overestimated performance of these approaches of commit message generation in practical scenarios, for example when generating messages for human-made (as opposed to bot-generated) code changes.
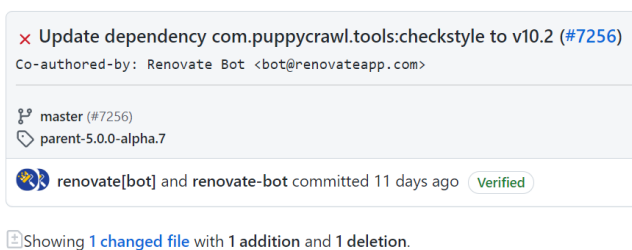

Fig. 2: Example of a commit message submitted by a bot

2.4 Quality of Datasets

Learning-based models require datasets of sufficient quality. In recent years, researchers have raised concerns about the reliability of datasets that have been used to construct models for diverse software engineering tasks. For example,

2. https://docs.github.com/en/codesecurity/dependabot
3. https://github.com/square/okhttp/commit/3ac5b0ed7627f23695cb12f18260165c46a33c5d
4. https://github.com/square/okhttp
5. https://docs.renovatebot.com/configuration-options

Kim et al. [38] measured the noise-resistant ability of two commonly used defect prediction algorithms and found that the prediction performance significantly decreased when the dataset contained 20% to 35% noise. Zhu et al. [39] applied noise handling techniques to enhance the accuracy of logging suggestions. Their results showed that removing noise data can effectively enhance a model's ability to learn common logging knowledge. For another example, Wu et al. [40] examined the impact of mislabeled instances on security bug report (SBR) prediction, and found that the performance of three baseline approaches consistently achieved better performance on clean datasets when compared to uncleaned (noisy) datasets.

In the context of commit message generation, researchers [1, 9] have repeatedly pointed out the presence of questionable data in the widely-used dataset collected by Jiang and McMillan [41], which we refer to as the J-M dataset (see Sec. 3.2). Although Liu et al. found that around 16% of the commit messages in the J-M dataset are already generated by bots, or describe repetitive trivial changes [1], which we consider 'noisy data' too, they did not remove such commits systematically: they only found one bot account (i.e., `liferay-continuous-integration`) and five trivial message patterns (see Table 1). Wang et al. [9] extended these patterns to filter out noisy commits but did not identify any more bot accounts. In this article, we report on a more thorough data cleaning procedure (see Sections 3.2.3 and 3.2.4).

2.5 Large Language Models

Large language models (LLMs) have garnered significant attention and adoption in both academic and industrial domains [42], including Software Engineering (SE) [17, 43, 44], due to their exceptional performance across a wide range of applications. For example, UniXcoder [45] has been used in the code summarization task, which is highly relevant to commit message generation. Additionally, ChatGPT's ability to answer questions in a conversational manner [46] allows it to generate commit messages by answering a diff-related prompt. To the best of our knowledge, the use of LLMs for commit message generation has not received much attention, and yet this appears a promising avenue. Thus, in this article, we also consider two popular LLMs.

3 STUDY DESIGN

To address the four research questions we pose in this article (see Sec. 1), we conducted a mixed-methods study using both quantitative and qualitative data. We selected four representative approaches for automatic commit message generation. Further, we collected and preprocessed commit data from the Top 1,000 most-starred Java repositories on GitHub, creating a new benchmark. We then evaluated the four approaches on the new benchmark, and evaluated their performance. Qualitatively, we surveyed developers to gain practical insights into commit message generation. Finally, we evaluated the performance of two LLMs to automatically generate commit messages. Figure 3 presents an overview of our approach.
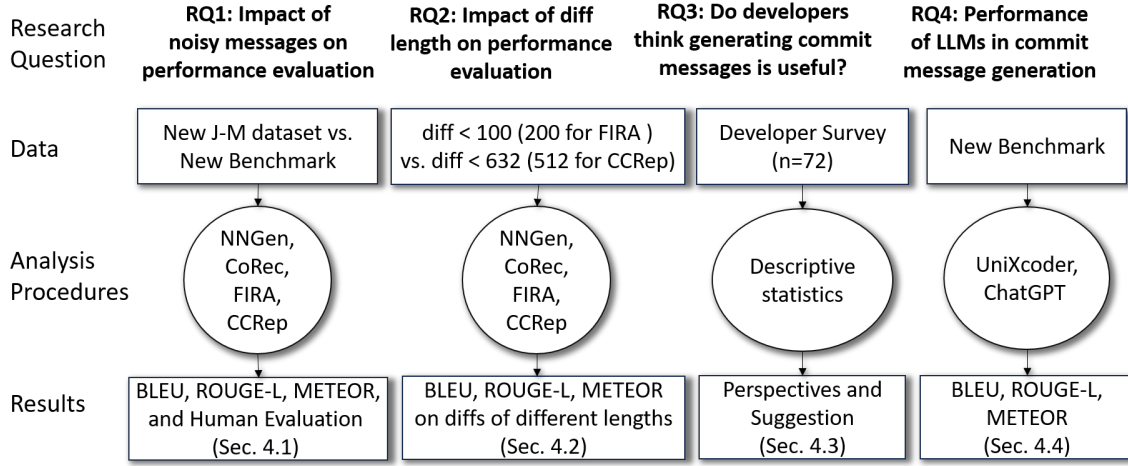
This article has been accepted for publication in IEEE Transactions on Software Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSE.2024.3364675

6

Research Question	RQ1: Impact of noisy messages on performance evaluation	RQ2: Impact of diff length on performance evaluation	RQ3: Do developers think generating commit messages is useful?	RQ4: Performance of LLMs in commit message generation
Data	New J-M dataset vs. New Benchmark	diff < 100 (200 for FIRA) vs. diff < 632 (512 for CCRep)	Developer Survey (n=72)	New Benchmark
Analysis Procedures	NNGen, CoRec, FIRA, CCRep	NNGen, CoRec, FIRA, CCRep	Descriptive statistics	UniXcoder, ChatGPT
Results	BLEU, ROUGE-L, METEOR, and Human Evaluation (Sec. 4.1)	BLEU, ROUGE-L, METEOR on diffs of different lengths (Sec. 4.2)	Perspectives and Suggestion (Sec. 4.3)	BLEU, ROUGE-L, METEOR (Sec. 4.4)

Fig. 3: Overview of our approach

3.1 Evaluated Approaches

3.1.1 Approaches of Commit Message Generation

We evaluated four prominent approaches for automatic commit message generation. As pointed out earlier, rule-based approaches suffer from generalization issues because the generated output tends to be very long and cannot generate a rationale for code changes [9, 10], which is why did not include a rule-based approach in this study. The other three categories of approaches have become the primary approaches for generating commit messages. For this study, we selected one approach that achieves the best performance from each of these three categories. The goal of this study is not to evaluate all these approaches, but rather to demonstrate the key shortcomings in the way they have been evaluated previously. The selected approaches are representative of the current state-of-the-art in automated commit message generation.

We selected NNGen as a representative of retrieval-based approaches,[6] FIRA [10] as a representative of learning-based approaches, and CoRec [9] as a representative of hybrid approaches with consideration of their performance and difficulty of reproduction. We briefly describe these.

NNGen [1] relies on the Nearest Neighbor algorithm to select a message from the training dataset. It first represents the diffs of all the commits as vectors by leveraging the "bags of words" representation that is widely used in information retrieval [47]. NNGen calculates the cosine similarity for a given diff and each diff in the training dataset and selects the top k training diffs with the highest similarity scores. After that, NNGen computes the BLEU-4 score based on the input diff and the selected diffs, and outputs the message of the selected diff with the highest BLEU-4 score.

FIRA [10] first represents code changes via an abstract syntax tree (AST) based fine-grained graph, which allows the consideration of code edit operations in diff and code tokens at sub-tokens and integral granularities. Then, FIRA implements a graph-based encoder and a transformer-based decoder to generate commit messages.

---

6. Although LogGen [21] reported slightly better performance than NNGen, Tao et al. [12] identified major implementation flaws, which is why we did not select LogGen.

CoRec [9] combines the advantages of both information retrieval and neural machine translation by building a context-aware encoder-decoder model and a sophisticated vocabulary. For a certain diff, CoRec first retrieves the most similar diff from the training set, and then uses the retrieved diff to enhance the performance of the final generated vocabulary.

In addition to the three selected approaches that are specifically designed for automatically generating commit messages, we also considered one pre-trained diff encoding model, i.e., **CCRep** [18], which can be used to generate commit messages and achieved the best performance when compared with other pre-trained models [18]. CCRep [18] first splits a given code change into the before-change code and the after-change code, leverages the pre-trained CodeBERT [48] to obtain the code embeddings, and uses a query back mechanism to extract and encode the changed code fragments and make them explicitly interact with the whole code change.

3.1.2 Large Language Models

Numerous LLMs have been released in recent years with great promise for a wide variety of software engineering tasks [17, 43, 44]. For this study, we selected two: UniXcoder [45], which is specific to code-related tasks, and ChatGPT [46], a well-known general-purpose LLM.

UniXcoder [45] is designed to handle code-related tasks. It is based on the capability to simultaneously pre-train both the encoder and decoder, enabling it to benefit from pre-training data for code-related understanding and generation tasks. UniXcoder utilizes a multi-layer transformer and incorporates multi-modal contents such as code, code comments, and abstract syntax tree (AST) to enhance code representation. The maximum input length for UniXcoder is 1,024 tokens.

ChatGPT [46] is a general-purpose LLM from OpenAI. It has a large number of network parameters and has shown excellent performance in many tasks [49, 50]. In this study we selected ChatGPT 3.5, since it is free to use and offers an open API interface. The maximum input size of ChatGPT based on `gpt-3.5-turbo-1106` is 16,385 tokens. Our objective is to establish a reasonable baseline for the potential

performance of LLMs in commit message generation. We defer experiments with the more sophisticated GPT-4 model to future research.

3.2 Data Collection and Preprocessing

The following describes the datasets and their corresponding cleaning steps.

3.2.1 The Jiang-McMillan (J-M) Dataset

Most approaches for automatically generating commit messages are based on a widely used dataset collected and processed by Jiang and McMillan in 2017 [41]. This dataset extracts the first sentence from the original commit messages contained in the Top 1,000 Java projects (as measured by 'stars') hosted on GitHub, excluding rollback/merge commits, commits with empty or non-English messages, duplicated code changes, and diff files that are larger than 1MB or containing non-ASCII codes. The dataset only retains the Verb-Direct Object (V-DO) messages, e.g., "update a method," because Jiang and McMillan argued such messages have better quality. The filtered dataset contains 509k diff files and corresponding commit messages. We label this dataset the "J-M" (for Jiang and McMillan) dataset.

Before using the J-M dataset to automatically generate commit messages, the three evaluated approaches also each have their own data preprocessing steps because of noisy data or specific requirements of the respective approaches. Table 1 shows the filtering rules for each approach, where these filtering conditions proposed by each approach are itemized using a black circle (•), and inherited preprocessing rules followed by previous studies are itemized using an open circle (○). CoRec has been evaluated on the same dataset processed by Liu et al. [1] and used by NNGen [9]. After cleaning, approximately 26k remaining commits were used to train and test NNGen and CoRec [1, 9], and 90k commits were used to evaluate FIRA [10]. Restricting messages or diffs can greatly reduce the size of the dataset, i.e., from 509k to 26k or 90k respectively. Further, bot-generated and uninformative commits also represent a negligible proportion.

3.2.2 Data Collection

While the J-M dataset and its various derivations have been widely used to automatically generate commit messages [1, 9], these datasets still contain noisy messages. Since the J-M dataset along with its filtered version only retains the anonymized message and diff of each commit and lacks key information such as the commit ID that is required for further processing, we decided to create a new dataset.

We re-collected commits from the Top 1,000 Java projects (as measured by Stars) hosted on GitHub, not counting projects that were not software projects, such as Java interviews or training documents; such projects were ignored. We initially obtained 2.5 million commits from the collected Java projects. We followed Jiang and McMillan [41] to filter the dataset (see Sec. 3.2.1 for details). Our final dataset contained 889,329 commits, which we refer to as the "New J-M Dataset."

Liu et al. observed that approximately 16% of commit messages in the J-M dataset are noisy, as they were automatically generated or described repetitive, trivial changes [1]. However, as briefly mentioned they did not conduct a thorough data cleaning: they only found one bot account (i.e., `liferay-continuous-integration`) and five 'trivial message' patterns; this is shown in Table 1. With the increasing popularity of bots and the potential existence of other trivial messages, we conducted a thorough cleaning of noisy messages in our dataset. Sec. 3.2.3 and Sec. 3.2.4 describe these steps in detail.

3.2.3 Removing Commits Submitted by Bots

To remove commits submitted by bots, we must establish a list of bot accounts that is as complete as possible. To this end, we conducted a systematic literature review (SLR) to identify bot accounts. We followed widely-used guidelines for conducting SLRs [51] to identify and filter related papers in four steps:

1) Define the search scope. We selected three digital libraries, ACM Digital Library, IEEE Xplore Digital Library, and Springer Link, which are commonly included in SLRs in software engineering.

2) Define the search rules. Based on the goal of our SLR, namely, to identify bot accounts, the search keywords are variants of 'bot,' such as "non-human." We also considered the plural form of each search keyword. To ensure the identified bots were related to software development, we also identified keywords capturing the context in which the bot accounts were applied. Table 2 lists the keywords in the search target and context categories. When forming the search queries, we selected one keyword from each category and joined them using the 'AND' operator. After applying these search queries in the digital libraries listed above, we initially obtained 213 publications.

3) Manual Selection. We manually read the title, abstract, and keywords of each of the 213 papers, and selected the papers that sought to identify bot accounts and identified bot lists for further analysis. After this step, we identified 21 papers.

4) Forward and backward snowballing. Using forward snowballing, we inspected the publications that cited the selected studies and found two additional papers. Using backward snowballing, we inspected the references of the selected studies and identified three additional studies. After this step, we identified a total of 26 studies.

5) Bot account listing. We reviewed the 26 relevant studies, mainly focusing on collecting bot accounts identified in these studies and their associated datasets that list bot accounts. After removing duplicated accounts, we obtained a staggering number of 5,229 bot accounts that were identified or built by previous studies. The full papers and bot accounts are listed in our appendix [52].

After obtaining the list of 5,229 distinct bot accounts, we removed all commits submitted by these accounts by comparing the author name of a commit to all bots in the list. We found that in the "New J-M Dataset" dataset, we identified 374 bot accounts that collectively made 21,656 commits (2.4%). Table 3 lists the commit distribution of the top five bots ranked by their submitted commits. We excluded these bot-generated commits from "New J-M Dataset" dataset, and labeled this cleaned dataset without bot messages the "No-bots dataset," which contains 867,673 commits.

TABLE 2
Systematic review search keywords

Category	Keywords[1]
Target	"bot*", "automated commit*", "non-human account*", "bot commit*", "bot identification", "devbot*"
Context	"commit*", "software development*", "collaborative software development", "open source", "OSS", "FOSS", "FLOSS"

[1] '*' means plural forms are included.

TABLE 3
Commit distribution of the top 5 bots

Bot	Commit count
dependabot-preview[bot]	3,585
dependabot[bot]	2,847
Renovate Bot	409
nextcloud-android-bot	242
Gary Bot	212

3.2.4 Removing Commits with Non-informative Messages

Except for the five trivial commit message patterns identified by Liu et al. [1], the dataset may still contain other types of non-informative messages. For example, some commit messages contain only a single token which cannot express accurate information, such as "Readme." A good commit message should explain what changes were made, and why [2, 53, 54]. In this article, we follow Tian et al. [2] and deem messages that neither explain what was changed, nor why those changes were made, as uninformative messages or trivial messages, which can reduce data quality for generating commit messages and should be removed. There are two advanced commit classifiers [2, 29]. Although the classifier proposed by Li and Ahmed [29] has a higher F1 score than Tian et al.'s [2], their method requires manually examining the content of the links contained in commit messages, which would not be feasible to apply in our large-scale dataset. Thus, we used Tian et al.'s Bi-LSTM-based classifier which can identify well-written messages with a precision of ca. 81% [2]. By leveraging the classifier in the No-bots dataset, we identified 39,144 (4.5%) trivial messages. We labeled this dataset, which excludes trivial messages, the "New Benchmark," which contains 828,539 commits. We conducted validation of whether the performance of the classifier can still hold in our dataset. Specifically, we randomly selected a dataset of size 384 (with an error margin of 5% and a confidence level of 95%) from the "No-bots dataset" constructed in Sec. 3.2.3. Two authors manually classified the messages as informative or non-informative (with a 0.66 kappa coefficient). Conflict annotations were solved by face-to-face meetings. When compared with the results classified by Tian et al.'s approach, we found 351 commits (91.4%) have the same labels, which indicates that the performance of Tian et al.'s classifier is kind of stable in our dataset.

3.3 Evaluation Metrics

To compare the performance of the four selected approaches on the dataset with different configurations, we adopted the widely-used metrics BLEU, ROUGE-L, and METEOR [12, 55]. We describe the three metrics' rationale and computation details next.

1) **BLEU** (Bilingual Evaluation Understudy) calculates the modified n-gram (for BLEU-4, n=1, 2, 3, 4) precision of a generated message to the reference message, then measures the average modified n-gram precision with a penalty for overly short sentences [56]. The BLEU value ranges from 0 to 1. A value of 1 indicates the generated message is identical to the reference one. BLEU can be calculated by the following formula:

$$\text{BLEU} = \text{BP} \cdot \exp \left(\sum_{n=1}^{N} w_n \log p_n \right) \quad (1)$$

where w_n is the weight of n-gram with a value of $1/N$. Normally, $N = 4$. p_n is n-gram precision, which can be obtained as follows:

$$p_n = \frac{c_n}{l_n} \quad (2)$$

where c_n represents the count of correctly predicted n-grams between the generated message to the reference message, and l_n represents the total number of n-grams in the generated message. BP is the brevity penalty factor:

$$\text{BP} = \begin{cases} 1, & \text{if} \quad \text{len}_{gen} > \text{len}_{ref} \\ e^{1-\text{len}_{ref}/\text{len}_{gen}}, & \text{if} \quad \text{len}_{gen} \leq \text{len}_{ref} \end{cases} \quad (3)$$

where len_{gen} is the length of the generated message and len_{ref} is the length of the reference message.

2) **ROUGE-L** measures the F-score of precision and recall on the basis of the longest common subsequence (LCS) between a generated message and a reference message [57]. Its value is calculated as follows:

$$F_{lcs} = \frac{\left(1 + \beta^2\right) R_{lcs} P_{lcs}}{R_{lcs} + \beta^2 P_{lcs}} \quad (4)$$

where F_{lcs} is the ROUGE-L score of the generated message and the reference one. $\text{LCS}(ref, gen)$ is the longest common subsequence (LCS) between the two sentences, and β is a hyperparameter commonly assigned a relatively large value. LCS considers sentence-level structure similarity and automatically identifies the longest co-occurring in-sequence n-grams. P_{lcs} and R_{lcs} represent the precision and recall, respectively, of LCS between a generated message and a reference message; these are calculated as follows:

$$P_{lcs} = \frac{\text{LCS}(ref, \, gen)}{\text{len}_{gen}} \quad (5)$$

$$R_{lcs} = \frac{\text{LCS}(ref, \, gen)}{\text{len}_{ref}} \quad (6)$$

3) **METEOR** is based on a generalized concept of 1-gram matching between machine-produced and human-produced reference messages [58], 1-gram can be matched based on their surface forms, stemmed forms, and meanings. METEOR computes a score for this matching using a combination of 1-gram precision,

1-gram recall, and a measure of fragmentation that considers the order of the matched words in the generated messages in relation to the reference. METEOR is calculated as:

$$\text{METEOR} = \text{F}_{mean}(1 - \text{Penalty}) \tag{7}$$

where F_{mean} is computed with 1-gram precision (P) and 1-gram recall (R), and Penalty represents the penalty factor applied to fragmentary matches:

$$\text{F}_{mean} = \frac{10\text{PR}}{\text{R} + 9\text{P}} \tag{8}$$

$$\text{Penalty } = 0.5 * (\frac{\#\text{chunks}}{\#\text{unigrams} - \text{matched}})^3 \tag{9}$$

where $\#\text{chunks}$ is the number of matched chunk and $\#\text{unigrams} - \text{matched}$ is the number of matched unigram (1-gram).

Originally, NNGen was only evaluated with BLEU to evaluate its performance (see Table 1), while CoRec, FIRA, and CCRep were all evaluated using all three metrics listed above. To keep the comparison as comprehensive as possible, we applied all three metrics to evaluate the performance of the four selected approaches. Further, different from NNGen and CoRec, the evaluation of FIRA was done using a variant of BLEU, i.e., B-Norm BLEU [10], which is insensitive to the character case. To retain consistency with [10], we also used B-Norm BLEU to evaluate the performance of FIRA. CCRep is evaluated by both the two types of BLEU scores in [18]. In this study, we chose B-Norm BLEU to evaluate CCRep.

3.4 Developer Survey

Developers are recommended to summarize the changes they made in a short sentence (the subject), and add a more extensive description (the body) to provide additional details explaining what was changed, and why [2]. While not every commit requires both a subject and a body, current learning-based or retrieving-based approaches are only aimed at generating the subject. To investigate the perceived importance and effectiveness of this, we conducted a qualitative survey to solicit opinions from experienced software developers. We selected the Top 10 most active developers (ranked by the number of commits they contributed) from the Top 200 Java projects. We identified 1,782 survey candidates from the 200 projects (157 developers participated in more than one project, resulting in 218 duplicated accounts), and invited them to fill out a short questionnaire (see appendix [52]).

The questionnaire started with a background question regarding the respondent's tenure as a contributor to open source projects, to ensure that respondents have a meaningful experience in collaborative software development, and thus could better appreciate the importance of commit messages. We then asked respondents to select the most difficult aspect and the most time-consuming aspect separately when writing commit messages with the following options: (1) writing the message subject; (2) writing the message body; or (3) other. We also invited respondents to explain their choices. We further asked respondents what they usually write on the message subject. Finally, we asked for respondents' opinions on the effectiveness of automatically generating message subjects, and solicited any further advice on commit message generation.

Thirty-five emails could not be delivered. We received 72 responses, resulting in a response rate of approximately 4.1% ($\frac{72}{1,782-35}$). This response rate is comparable to other research surveys in software engineering [cf. 59–61]. We used open coding [62] to analyze the responses to these three open questions, assigning codes to specific phrases of interest. We then sorted the codes, and merged them where appropriate. During the sorting process, we grouped related codes into categories. All codes and categories were reviewed, discussed, and validated by two authors of this article. During this process, we resolved any disagreements by elaborating our rationale for specific codes.

4 RESULTS

We now present the results of the four research questions outlined in Sec. 1. We first demonstrate the impact on the performance of the four selected approaches when noisy commit messages are excluded from datasets (RQ1, Sec. 4.1). We then demonstrate the impact of changing the size of the maximum commit's diff (the number of changes to contents) on the four approaches in Sec. 4.2 (RQ2). Sec. 4.3 presents developers' perspectives on the effectiveness of generating message subjects (RQ3). Finally, Sec. 4.4 presents the performance of UniXcoder and ChatGPT on commit message generation in a more real-world scenario (RQ4).

4.1 The Impact of Noise on Performance of Commit Message Generation Approaches

4.1.1 Procedure and Metric Evaluation

To investigate the impact of noisy commit messages (those submitted by bots and uninformative messages) we evaluated the performance of four representative approaches of the state-of-the-art (i.e., NNGen [1], CoRec [9], FIRA [10], and CCRep [18]), before and after removing these noisy messages. Each of these approaches may have different data requirements (see Table 1). For example, NNGen requires that messages should have no more than 30 tokens. We applied the filtering rules of NNGen [1] and FIRA [10] on "New J-M Dataset" and "New Benchmark" to prepare the datasets. Following the original evaluation design of CoRec [9] and CCRep [18], we evaluate the two approaches on the datasets prepared for NNGen. Table 4 shows the size of the processed datasets, where the training set, testing set, and validation set are divided following the settings of the four approaches. We then re-trained and tested the four approaches based on their replication packages. We calculated the BLEU, ROUGE-L, and METEOR metrics to evaluate the impact of removing noisy commit messages on their performance.

Table 5 shows the results of the four approaches running on the corresponding datasets before and after removing noisy commits, and column "Delta" presents the performance change after removing noisy commits in terms of the three metrics. The performance of NNGen, CoRec, and CCRep is considerably worse after removing the bot-submitted and uninformative messages from the evaluation results of all three metrics. For example, the BLEU score of NNGen on 'New Benchmark' is 46% (36% for CoRec and

TABLE 4
Statistics of RQ1's benchmark datasets.

	Training		Validation		Testing	
	Before	After	Before	After	Before	After
NNGen	41.5k	30.3k	2.3k	1.7k	2.3k	1.7k
CoRec	41.5k	30.3k	2.3k	1.7k	2.3k	1.7k
FIRA	41.2k	39.0k	5.2k	4.9k	5.2k	4.9k
CCRep	41.5k	30.3k	2.3k	1.7k	2.3k	1.7k

TABLE 5
Performance of the four approaches before and after removing noisy commits

	Metrics	Before[1]	After[2]	Delta	%
NNGen [1]	BLEU	39.43	21.41	−18.02	−45.7%
	ROUGE-L	47.25	36.01	−11.24	−23.8%
	METEOR	26.19	18.25	−7.94	−30.3%
CoRec [9]	BLEU	47.89	30.77	−17.12	−35.7%
	ROUGE-L	51.03	39.04	−11.99	−23.5%
	METEOR	29.60	20.85	−8.75	−29.6%
FIRA [10]	BLEU	15.43	15.75	0.32	2.1%
	ROUGE-L	19.90	20.11	0.21	1.1%
	METEOR	14.01	14.04	0.03	0.2%
CCRep [63]	BLEU	49.27	37.43	−11.84	−24.0%
	ROUGE-L	54.40	42.90	−11.50	−21.1%
	METEOR	33.44	22.10	−11.34	−33.9%

[1] Using the dataset 'New J-M dataset'
[2] Using the dataset 'New Benchmark'

24% for CCRep) *lower* than its performance on 'New J-M Dataset.' Such a high degree of degradation suggests that the high performance of the three approaches can be ascribed, at least in part, to the presence of noisy commits in the original J-M dataset. In contrast, the performance of FIRA remains largely the same in terms of the three metrics, and is slightly, though not statistically, higher. This indicates that removing noisy data has only a limited impact on the performance of FIRA, which is learning- and AST-based techniques, and mainly focuses on generating messages for code changes, e.g., changes happening in '.java' files. This sort of message can rarely be generated by bot accounts and usually has informative text because of the greater complexity of code changes when compared with updating documentation or configuration files. Specifically, we only found 2,841 (5.5%) noisy messages in the commits happening in Java files (i.e., a '.java' extension), which is considerably smaller than the percentage (27.3%) of noisy messages filtered out in the datasets for NNGen, CoRec, and CCRep.

TABLE 6
Statistical difference of the four approaches' results before and after removing noisy commits

Metric	NNGen	CoRec	FIRA	CCRep
BLUE	0.12[*]	0.15[*]	n/a[1]	0.16[*]
ROUGE-L	0.17[*]	0.17[*]	n/a[1]	0.16[*]
METEOR	0.16[*]	0.16[*]	n/a[1]	0.16[*]

[*] p-value < 0.001.
[1] Results for FIRA were not statistically significant.

To assess whether the observed performance change of the four approaches is statistically significant, we conducted a Mann–Whitney U test [64] on the value distribution of the three metrics of each evaluated approach before and after removing noisy commits, respectively. If a statistically significant difference ($p < .05$) was found, we also calculated the effect size, which provides a measure of how large the difference is [65].[7] Table 6 presents the results; for NNGen, CoRec, and CCRep, we observe that the differences between before and after removing noisy commits are statistically significant for all three metrics, though with small to medium effect sizes. This indicates that the performance of NNGen, CoRec, and CCRep decreases significantly when filtering out the commits with bot-submitted and uninformative messages. For FIRA, we found no statistical differences (and thus no effect size), meaning that the performance of FIRA is stable and not affected by the presence of noisy commit messages.

4.1.2 Human Evaluation: Evaluating Practical Impact

The three metrics (BLEU, ROUGE-L, and METEOR) afford a quantitative assessment of performance changes based on a numeric analysis of the overlap of tokens. However, such evaluations are rigid and lack subtlety. Generated messages may have only little overlap with ground truth messages, and yet convey a very similar meaning. Thus, we also conducted a qualitative assessment to complement the quantitative assessment to develop a more intuitive impression of the performance change.

We sampled 60 commits at random from the test sets of the four approaches, before and after removing bot-submitted and uninformative messages ($8 \times 60 = 480$ commits in total).[8] Each sampled commit has a generated message and a ground truth message. Two authors independently followed a scoring scheme based on previous studies [1, 9, 10] to manually score the generated message. The two raters did not know which of the approaches generated the messages, and whether these were based on datasets before or after removal of noisy messages, to reduce any potential threat to validity involved in the manual labeling process. Table 7 shows the scoring rules with five levels, ranging from 0 to 4; a higher score means better similarity between the generated commit message and the ground truth. We calculated the kappa coefficient (κ) between the two authors, which was 0.71 indicating a substantial level of consistency [66]. When the two raters differed in their scores, these differences were resolved through discussion leading to a consensus on the final scores. The difference in scoring was mostly only 1, meaning that the two raters varied in their judgment only by a small degree. During the discussion, we adjusted the scores of approximately 22.3% (#107) of the messages.

We followed prior work [10] in defining the scores 0 and 1 as low-quality, 2 as medium-quality, and 3 and 4 as high-quality. Table 8 presents the results of this human evaluation. The portion of messages with low scores (0-1, indicating irrelevance or no relation) increased for NNGen, CoRec, FIRA, and CCRep, with 25, 16.7, 15, and 6.67 percentage

7. Effect size ≥ 0.1, effect size ≥ 0.3, and effect size ≥ 0.5 represent small, medium, and large differences, respectively [65].
8. The reason for sampling 60 commits in each case is to ensure the overall sample size (480) is feasible for manual analysis.

TABLE 7
Scoring criteria

Score	Definition
0	Entirely irrelevant, and no tokens are shared.
1	Unrelated, but there is some token overlap.
2	Somewhat related, but the generated message includes redundant or missing information.
3	Similar, with mostly identical tokens, but variations in detail lead to inconsistent semantics.
4	Semantic consistency.

Fig. 4: Distribution of token numbers contained in diffs

points, after removing noisy messages respectively. However, the portions of generated messages with high scores (3-4, indicating messages are deemed similar or semantically consistent) decreased for NNGen, CoRec, and CCRep after removing bot-submitted and uninformative messages; the drop was considerable at 16.7, 25, and 16.7 percentage points, respectively. For FIRA, there was a small but insignificant increase. We explored the reasons for these drops in performance and found that most well-generated messages (i.e. with high scores, 3-4) in the New J-M dataset have simple patterns, such as "update <file_name>," "bump <version_number>," and "added <version_number>," which are either generated by bots, or uninformative and are removed in our New Benchmark. We also observe that the average scores of all four evaluated approaches are close to 1: the generated messages are *"unrelated, but there is some token overlap"* (Table 7), after removing these noisy messages.

Both the quantitative and qualitative evaluation results suggest that including noisy data has a serious impact on the current approaches' performance. Considerable work is needed to filter noisy data well and automatically generate high-quality commit messages that are of practical value.

Summary for RQ1: After removing bot-generated and uninformative commit messages from the training and testing datasets, the performance of NNGen, CoRec, and CCRep greatly declines in comparison to the original evaluations, in terms of BLEU, ROUGE-L, and METEOR metrics, while the performance of FIRA remains stable. The qualitative evaluation suggests that the performance of all four approaches declines after removing bot-generated and uninformative commit messages.

4.2 The Impact of Small Commit Size on Performance of Commit Message Generation Approaches

Most automated approaches for generating commit messages are based on datasets that only include commits with a diff length of no more than 100 tokens [8, 9], because neural machine translation techniques usually empirically set the lengths of source and target sequences between 50 and 100 [8, 67]. A few learning-based approaches (e.g., FIRA [10]), which can only handle code changes, set the diff length to no more than 200 tokens to learn more information through the AST of a diff. Although retrieval-based approaches (e.g., NNGen) have no length constraints from the implementation perspective, these approaches are only evaluated on datasets containing commits with diffs of no more than 100 tokens.

However, when we consider the distribution of the commit diff length, this 100-token limitation raises a critical concern. Figure 4 presents the diff length distribution of "New Benchmark," which contains 828k commits after filtering out bot-submitted commit messages and uninformative messages. The results show that only 5% of diffs have less than 100 tokens, 22% of diffs have less than 200 tokens, and 39% of diffs have more than 1,000 tokens. Figure 5 presents a commit with a diff length of 100 tokens; it is intuitively clear that many commits are, in fact, much larger than 100 tokens. In particular, the number of tokens in our dataset ranges from 12 to 4,000, with a median length of 632 tokens, considerably larger than the 100-token limit applied in the evaluation of almost all automatic approaches (or 200 tokens). Thus, it is clear that these approaches are not thoroughly evaluated, casting serious doubt on the practical relevance of these approaches. To move the field forward, we must conduct more rigorous evaluations to establish the limits of the state-of-the-art.

TABLE 8
Results of the qualitative evaluation

Approach	Score[1]	New J-M Dataset	New Benchmark	Delta[2]
NNGen [1]	Low Score	36.67%	61.67%	25.00
	Medium Score	15.00%	6.67%	−8.33
	High Score	48.33%	31.67%	−16.66
	Average Score	1.97	1.33	−0.64
CoRec [9]	Low Score	38.33%	55.00%	16.67
	Medium Score	8.33%	16.67%	8.34
	High Score	53.33%	28.33%	−25.00
	Average Score	2.17	1.48	−0.69
FIRA [10]	Low Score	58.33%	73.33%	15.00
	Medium Score	31.67%	15.00%	−16.67
	High Score	10.00%	11.67%	1.67
	Average Score	1.23	1.00	−0.23
CCRep [18]	Low Score	35.00%	41.67%	6.67
	Medium Score	8.33%	6.67%	−1.66
	High Score	45.00%	28.33%	−16.67
	Average Score	1.94	1.6	−0.34

[1] Low scores 0-1, Medium score: 2, High scores: 3-4
[2] The delta numbers are percentage points

This article has been accepted for publication in IEEE Transactions on Software Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSE.2024.3364675

12

Fig. 5: A real-world example commit with a diff length of 100 tokens [68]

To investigate the performance of the state-of-the-art techniques in a more realistic scenario, we changed the diff length constraints to 632 tokens (the median length) and evaluated the four representative approaches using the same three metrics as before (BLEU, ROUGE-L, and METEOR). We applied the filtering rules of the four approaches listed but replaced the diff length constraints with 632 tokens on the dataset 'New Benchmark'.[9] From the filtered dataset, we further randomly selected 1,000 commits according to the following range of diff lengths as testing datasets: (0, 100], (100, 200], (200, 300], (300, 400], (400, 500], (500, 632]. For FIRA, the length range of the testing dataset starts with (0, 200]. Since CCRep is based on CodeBERT [48], which has a maximum input size of 512 tokens, our experimental constraint for CCRep is set to a maximum length of 512.

We evaluated the generation performance of the four approaches in two scenarios. First, we reused the four models trained on 'New Benchmark' in RQ1, i.e., diff lengths of training commits are limited to (0, 100] (and max. 200 for FIRA), and then tested their performance with all testing datasets. This first scenario affords an evaluation of the performance of the current models, which are trained on commits with restricted code changes (up to 100 tokens for NNGen, CoRec, and CCRep; up to 200 for FIRA), when generating messages for commits that capture larger diffs. When the performance is not as good as the four approaches' reported results (i.e., the 'After' column in Table 5), one of the most obvious reasons is that these models did not learn the characteristics of larger commits. Therefore, we designed a second scenario to alleviate this concern: we trained the models with an unsegmented dataset, i.e., a training set of diff length between (0, 632] (up to 512 for CCRep),[10] and similarly tested their performance with all testing datasets.

Figure 6 shows the performance of the four evaluated approaches when tested on commits with different diff lengths. The *x*-axis represents the diff lengths of commits in the testing sets. The blue lines represent the performance of the approaches as trained on datasets with the original diff length requirements ((0, 100] for NNGen, CoRec, and CCRep, and (0, 200] for FIRA), whereas the orange lines represent the performance of the approaches when trained on a dataset that has a maximum of 632 tokens (512 for CCRep).

9. The reason for not directly deleting the diff restriction is to reduce the time needed to conduct the evaluation, which can be very time-consuming.
10. Note: there is no overlap between the training data and these testing datasets.

The performance of the four approaches degrades considerably when the diff length of the test dataset increases, regardless of whether they are trained with commits whose diffs are in their required length range or between (0, 632]. When the diff length of test data continues to grow, the performance of the four evaluated approaches remains stable. The extent of degradation of NNGen, CoRec, and CCRep is greater than FIRA. The performance of NNGen, CoRec, and CCRep dramatically degrades when the diff length of the test dataset increases from (0, 100] to (100, 200]. For example, consider the BLEU metric: the performance of NNGen, CoRec, and CCRep decreases by 88.7%, 94.3%, and 89.6% respectively in the first evaluation scenario, and decreases by 28.2%, 65.8%, and 32.9% respectively in the second scenario. When testing commits with diff lengths belonging to [200, 300) on FIRA in the two training scenarios, the performance degradation is 12.8% and 16.1%, considerably smaller than the performance degradation of the rest three approaches. The relatively small degradation of FIRA's performance may be because FIRA filtered out all non-code changes and its AST-based encoding learned some knowledge between messages and the corresponding code diffs. At the same time, complete code changes may also increase the difficulty of generating messages, because FIRA has achieved the relatively worst performance when compared with the other three approaches. When comparing the four approaches' performance in the two training scenarios, we can see that training on commits with larger diffs can slightly improve their performance, but the testing results with larger diffs are significantly decreased. This indicates that generating messages for larger code diffs remains a challenge.

We conducted a Mann–Whitney U test [64] to examine whether the four models' performance falls between the original testing set and other testing sets with larger diffs (see the supplementary appendix for complete results). All p-values are below .05 with a small or medium effect size, indicating the performance degradation shown in Fig. 6 is statistically significant.

> **Summary for RQ2:** The state-of-the-art approaches for automated commit message generation have limited their datasets to commits whose diffs have no more than 100 or 200 tokens. However, only 5% of commits have a diff length of no more than 100 tokens, with an average of 632 tokens in our cleaned dataset, which was created using a similar procedure as the original J-M dataset. The performance of four state-of-the-art approaches, i.e., NNGen, CoRec, FIRA, and CCRep on commits with larger diffs degrades significantly, regardless of whether they were trained with a diff length of up to 100 tokens (200 for FIRA) or up to 632 (512 for CCRep) tokens. When compared with NNGen, CoRec, and CCRep, diff sizes have a smaller impact on the performance of FIRA.

4.3 Is Commit Message Generation Helpful?

Current approaches are designed to only generate the subject of a message, which usually has no more than 30 tokens (see Table 1). A commit message's *body* may contain more information for developers to understand code changes [2].
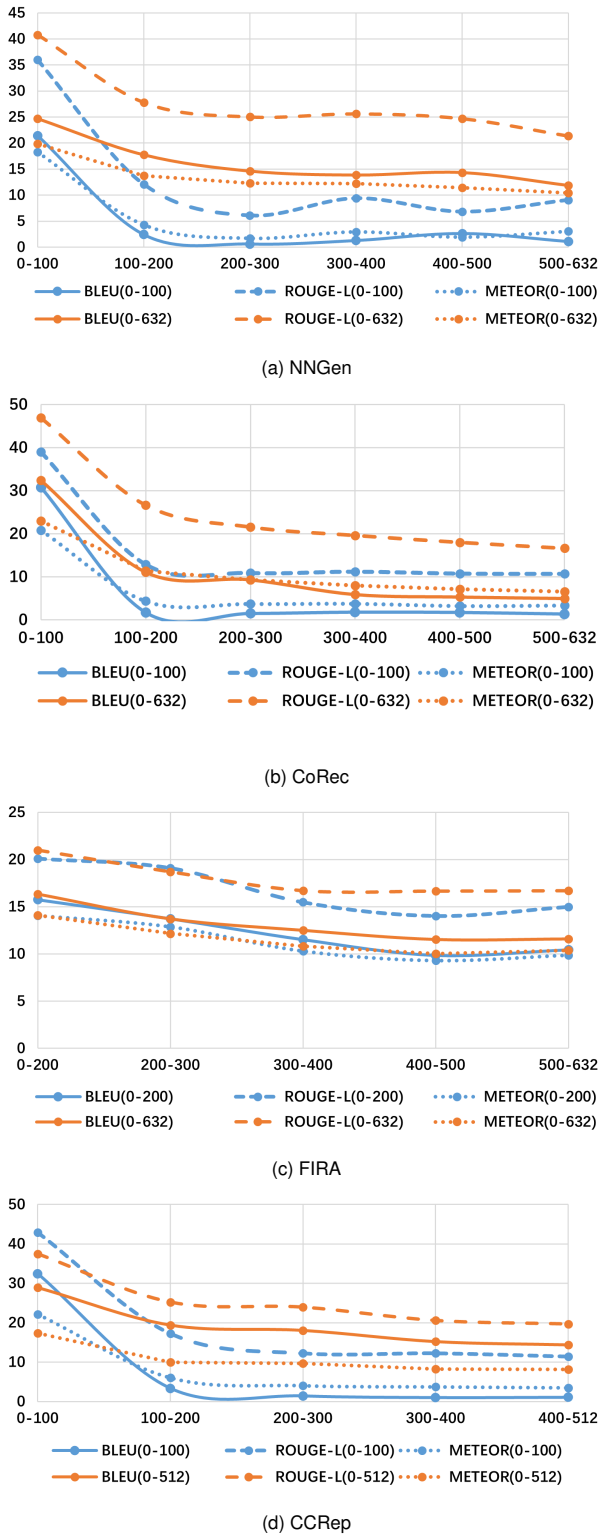
Fig. 6: Performance of the four approaches at different lengths of diff. Blue lines indicate that the diff length of the training sets is between 0 and 100. Orange lines indicate the diff length is no more than 632 (512 for CCRep) in the training sets.

Research Question 3 focuses on the perceived value of the automated generation of commit messages by developers. Clearly, these approaches are targeted at developers; if they do not perceive these approaches to have any value, then as

a field, we should reconsider whether further research into these approaches is valuable.

We conducted a survey among developers and received 72 responses; 81.9% of the respondents had participated in OSS projects for more than three years. The two most junior respondents in our survey had contributed to open source for more than three months. We present what developers perceive to be (1) the hardest work and (2) the most time-consuming part when writing commit messages, and why they think so. We also present feedback on future commit message generation.

4.3.1 What Makes Writing Commit Messages Hard?

We asked developers what aspect of writing commit messages they perceived to be the most challenging. For this question, we offered three options: the subject of a commit message, the body of a commit message, or an 'other' option, which invited respondents to provide further details. The results show that 69.4% (#50) of respondents indicate that *writing the subject of a commit message* is the hardest work, 19.4% (#14) of developers think writing the message body is the hardest work, and two respondents hold the view that both are difficult. Among the seven 'Other' responses, developers mentioned *"keeping commit messages consistent across time"* (#2), *"figuring out what to write"* (#1), *"Writing meaningful content that will be helpful to future developers."*(#1) and *"keeping things concise/compact"* (#1). Besides, two respondents indicated writing commit messages is not difficult.

Since the message body is usually longer than the subject, it is perhaps surprising that most respondents indicated that writing the subject is more challenging; understanding why this is so is worthwhile exploring further. After analysis of respondents' reasons for their choice, we identified two reasons that alone or together make writing the subject of a commit message difficult. First, it is difficult to create a good description within a *limited space* (mentioned by 42 developers). For example, one respondent indicated that: *"Providing a concise and informative subject in less than the recommended 50 characters is challenging."* Another respondent commented that *"It's a little hard to summarize what I have done in this commit."* Three respondents identified a second reason why writing the subject of a commit message is hard, namely, community norms: despite the size constraints and informative requirements, developers also need to tailor their subjects to a conventional commit standard which makes it more difficult. For instance, one respondent highlighted the importance of community norms and expectations regarding commit messages: *"One major issue with open source projects is the difficulty in unifying and standardizing commit messages."*

4.3.2 What Makes Writing Commit Messages Time-Consuming?

Since hard things are not always time-consuming, we also ask developers another question: What is the most time-consuming part when writing commit messages? Of the respondents, 37.5% (#27) found that writing the message subject costs the most time, while 51.4% (#37) of respondents found the body is the most time-consuming part of commit messages. Besides, three respondents claimed that both the subject and body are time-consuming. Among the five developers who chose the 'Other' option, two issues were

mentioned: first, the complexity of a change to code, and second, the ability to explain things within the scope of a message to a reviewer, as one respondent explained: *"How to make the reviewer better understand the meaning of this PR and make it easier to be accepted."*

After analyzing the reasons behind their choices, we found most respondents, who indicated writing message subjects is the most time-consuming, gave the same reasons as to why they consider subjects the hardest part: difficult to create a good description within a *limited space* (#9) and community norms (#1). For example, one developer indicated that:

> *"To make the subject short, I often go through many versions until I find a subject that fits."*

These results can be easily inferred, i.e., hard things usually need more time. Respondents also provided extra reasons for this choice: two held the view that *"most commits don't need a body, so subjects take more time in total"*; two respondents deem improving the readability of message subjects time-consuming. As for those who consider writing the message body to be more time-consuming, the most commonly mentioned reason is that the body of a message is longer than the subject and more details are needed. For example, one respondent said that:

> *"It typically requires a detailed explanation of what changes were made and why. This can involve summarizing the context of the changes, the rationale behind them, and any potential impact they may have."*

Four respondents also indicated that making the message body easy to understand is time-consuming. One interesting finding is that the majority of developers perceive writing message subjects as the hardest while writing message bodies is the most time-consuming. It means that researchers should consider generating the body of commit messages for the sake of improving efficiency.

4.3.3 Suggestions For Generating Commit Messages

Current learning-based, retrieval-based, hybrid, and even diff pre-trained approaches can only generate one sentence of less than 30 (or 20) tokens (see Table 1) as the commit message for a given diff, which are usually the commit message subjects. A complete commit message is expected to contain two parts: 1) explain why made this change, and 2) describe what was changed [2]. To better understand why some developers find writing message subjects hard and time-consuming, we asked developers what they usually write in the subjects. Approximately 80.6% (#58) of respondents indicated that they usually describe *what* was changed in

the subject of a commit message. One acclaimed strength of non-rule-based approaches is that they can provide the reasons behind code changes [1, 9]. However, fewer than 20% of developers explain why they made their change in subjects, and thus evaluating existing retrieval- and learning-based approaches on message subjects cannot demonstrate their ability to generate code change reasons.

Next, we asked developers to score the effectiveness of automatically generating message subjects on reducing their burden during development. We defined a 5-point scale, ranging from 0 (useless) to 4 (very effective); the mid-point (2) indicates 'neutral.' As shown in Fig. 7, most developers (n=27+8, 48.6%) are thinking highly of automatically generating message subjects. Fourteen respondents held a neutral attitude toward this task. This indicates that there is considerable interest in this topic from practitioners, and thus further work in this area will be very welcome. However, twenty-three developers did not perceive automatic subject generation as helpful. Combined with their answers for the hardest and most time-consuming parts of writing commit messages, one reason might be the fact that current approaches do not generate a message body that in their views seems to be in more urgent need of automated support. This means future approaches will require further sophistication before they may be seen as helpful, which may include the ability to generate longer or even complete commit messages to fulfill practical requirements.

Finally, we asked respondents to provide further advice on commit message generation. Their suggestions can be categorized as follows: 1) Follow common commits message guidelines for writing generated messages[11] (#10); 2) Take characteristics of different projects into consideration when generating messages, because message style and background knowledge can vary among projects (#4); 3) Integrate message generation approaches with existing development tools (#2); 4) Link relevant bug tickets / pull requests that provide extensive information that is useful to generation (#2); and 5) Focus on the body of commit messages (#1).

Summary for RQ3: Developers indicate that writing the subject of a commit message is hard, and approximately 37% of developers also find writing subjects time-consuming. Nearly half of the respondents hold a positive attitude towards automatically generating message subjects, indicating that generating commit messages (even just the first sentence of a message) is of great practical value, but needs to consider generating more complete messages.
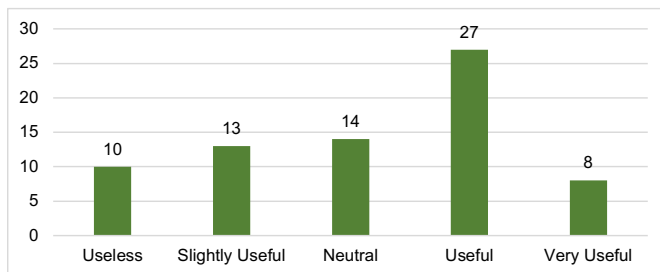

4.4 Performance of LLMs in Generating Realistic Commit Messages

We now address RQ4, which seeks to explore the performance of LLMs in generating realistic commit messages. We explore the performance of two representative LLMs, UniXcoder [45] and ChatGPT [46].

In this study, we employed the pre-trained UniXcoder-base model and further fine-tuned it for the task of com-

Fig. 7: Developers' view towards the effectiveness of generating commit message.

11. For example, https://cbea.ms/git-commit/#seven-rules and https://www.conventionalcommits.org/en/v1.0.0/

mit message generation. Specifically, we reused the code, hyperparameters, and dataset partitioning strategy of the UniXcoder-base model for the task of code summarization in Java, and replaced the "code + summary" dataset with the paired data "diff + message", which are selected from the "New Benchmark" meeting the requirement of diff lengths not exceeding 896.[12]

For ChatGPT, we randomly selected 1,000 data samples from the "New Benchmark" for experimentation. Following the instructions of White et al. [69], we designed the following prompt: "*This is the list of diff(s) for a commit, please generate the corresponding commit message, and please note providing the commit message directly without additional statements: <diff code>*".[13] We evaluated the generated messages against the ground truth using the three metrics introduced in Sec. 3.3.

Table 9 presents the performance of the two LLMs in terms of BLEU, ROUGE-L, and METEOR. The performance of UniXcoder on commits with a diff length of no more than 896 tokens is comparable with NNGen and FIRA but less promising than CoRec and CCRep on small diffs, i.e., a diff length of no more than 100 tokens for NNGen, CoRec, and CCRep, and no more than 200 tokens for FIRA, as shown in the fourth column in Table 5. For example, CCRep achieves a BLEU score of 37.43 for diffs with lengths in the range of 0-100 tokens, while the BLEU score of UniXcoder is 24.24 with a diff length constraint of 896 tokens. In a more realistic, we did not manually control the diff lengths of the commits when testing ChatGPT, and the evaluation results shown in Table 9 are not ideal. A BLEU score of 10.38 usually indicates the generated messages are of low quality [10]. Moreover, as shown in Sec. 4.1.2, human evaluation seems to have more restrictive standards than quantitative assessment. For example, the average score of the best-performing approach (i.e., CCRep) for RQ1 is 1.6, meaning the generated messages have some token overlap or are somewhat related. Given that the quantitative evaluation scores of UniXcoder and ChatGPT are lower than those for CCRep, we can conclude there is much room for improvement when generating messages for real-world diffs.

We also explored whether LLMs can generate full commit messages by comparing the length distribution of the ground truth commit messages and the generated ones; the distributions are shown in Fig. 8. The results indicate that commit messages generated by UniXcoder are generally

---

12. The 1,024 window size of UniXcoder is the sum length of source (i.e., 896) and target (i.e., 128). It means the maximum length of diff UniXcoder can handle is 896.

13. We specify "without additional statements" to prevent the generation of unrelated content, which would significantly decrease its evaluation score.

TABLE 9
Performance of LLMs: UniXcoder and ChatGPT

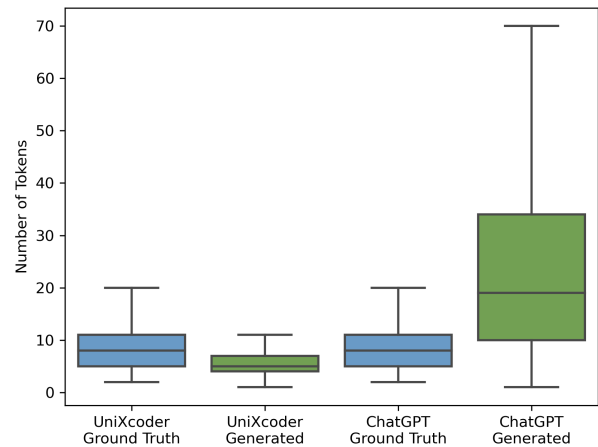|	UniXcoder [45] (diff lengths < 896)	ChatGPT [46] (diff lengths < 4,000)[*]
BLUE	24.24	10.38
ROUGE-L	33.03	13.74
METEOR	14.32	9.19

[*] This is the maximum diff length in our dataset.

Fig. 8: Length distribution of ground truth and two LLMs' generated commit messages, excluding outliers for simplicity.

shorter than ground truth messages, implying their inability to generate realistic messages. As for the messages generated by ChatGPT, these tend to be longer than the original (ground truth) ones. After a manual inspection, we found it predominantly focused on describing *what* was changed in the commits in detail, lacking an explanation of *why* those changes were made.

> **Summary for RQ4:** Although LLMs can take larger diffs as input, their performance of generating messages leaves much to be improved. UniXcoder tends to generate short messages, while ChatGPT can generate more detailed messages, which are very different from those written by developers.

5 DISCUSSION

We summarize the key findings of this study and suggestions for future work in Table 10. Automated generation of commit messages is a topic of substantial interest among researchers and also has practical relevance to software developers given that these approaches have considerable potential in reducing developers' workload. Several approaches to achieve this have been proposed, which can be organized into four categories (see Table 1). In this article, we have critically reviewed the performance of four approaches that are representative of the current state-of-the-art. The results demonstrate that the performance as reported in prior work is based on datasets that contain noisy data, and are limited in that they contain only commits with a diff length of up to 100 (or 200) tokens. In this article, we demonstrate that this has led to an overestimation of their performance. Our study shows that when more realistic datasets are used (commits with diffs larger than 100 or 200 tokens), without noisy data, the performance of these approaches degrades significantly. Based on these findings, we argue that the state-of-the-art approaches are not yet ready for adoption in the industry.

Several researchers in data-driven software engineering have pointed out that the appropriate use of data is critical [70–72]. For this line of work to be relevant and rigorous, future work must address these limitations. In this section,

we discuss several implications for future work, after which we discuss the limitations of this study.

5.1 Implications

5.1.1 Removing Irrelevant Commits

Current commit message generation approaches are trained and tested with datasets that contain bot-generated and low-quality messages. Their performance decreased after removing these noisy data. We suggest future work to conduct thorough data cleaning before evaluating their models. For bot-generated messages, we realized the limitation that the bot list we collected can easily become outdated. However, several studies are dedicated to automatically identifying bot accounts in OSS repositories, [cf. 14, 73, 74]. Future studies can select these bot detectors to clean their dataset before training and testing their commit message generation models. For low-quality messages, although there are automatic classifiers [2, 29] that can be leveraged, their accuracy is around 85% and may be specific to their own manually labeled datasets. Future studies should manually label a small set of data and evaluate the usability of the two classifiers first. Improving the performance of classifying high-quality commit messages is also an avenue for future research.

5.1.2 More Realistic Dataset

The sizes of commits vary widely; some commits change only a single character, whereas others capture very extensive changes including adding (or deleting) files. As mentioned, current datasets used are limited in the commits they contain, filtering out commits with a diff size of more than 100 or 200 tokens. After generating a new dataset according to the same procedures as those used for a frequently used dataset (see Sec. 3.2.1), we found that approximately 95% of commits are larger than 100 tokens, with a median of 632 tokens. In other words, it is very common in practice that commits include larger diffs, but current approaches are not trained or evaluated on these. For example, commit `#ccae978` [75] in the repository *spring-data-examples* includes 426 changed files, with 427 additions and 427 deletions. It is necessary to consider the practicality of generating commit messages. When testing the four state-of-the-art approaches with larger commits, their performance significantly decreases. This indicates that further work is needed to also be able to handle commits with bigger diffs.

Since the four evaluated approaches in RQ2 can achieve a relatively promising performance on small diffs, one suggestion is to shorten the length of long diffs on the basis of token importance. The importance of tokens can be measured through Shannon entropy [76], which indicates the richness and importance of the semantics provided by the token. Then, the top 100 tokens can be selected to build and run the models of message generation. We also see great potential in the use of LLMs such as GPT [77], PaLM [78], and LLaMA [79] to build message generation models, due to their ability to take very long input strings. This may be one avenue to address the current limitation of diff lengths. However, the performance of two LLMs we evaluated to address RQ4 (Unixcoder and ChatGPT) is not as promising

as expected: they either generate short messages or long-winded change descriptions, failing to convey the rationale for code changes. Further work still is needed to improve LLMs' performance in this task.

5.1.3 Generating Subjects of Commit Messages

Although a commit message's body is often much longer than the subject, most developers (69.4%) in our survey indicated that writing the subject is more difficult. This demonstrates the practical value of having an automated approach that can generate an appropriate subject for a given code change. However, in this study, we found that current approaches are not capable of fulfilling this task, because their performance decreases significantly when noisy data and larger, more realistic commits are excluded from datasets to train these approaches.

As pointed out repeatedly by both industry developers (see Sect. 4.3.3) and existing research [2, 27], commit messages may follow some patterns, which are related to project context, programming languages, maintenance activities, etc. Besides, the issue reports and pull requests of the corresponding code changes contain effective information for generating commit messages. Instead of simply taking the commit messages and diffs as input, we suggest taking project-specific characteristics, commit types, and other sources of information into consideration.

5.1.4 Non-code Commits

In practice, many commits include non-code diffs, e.g., diffs of text files, configuration files, and data files. Some commit message generation studies excluded such commits [10, 15]. These commits might be more complex than simple code changes, involving natural language and implicit context. When handling such diffs containing natural language, the problem can be viewed as finding synonyms and generating summaries. A neural machine translation technique may also work with some adaptations. To capture an implicit context, we think that there are many implicit conventions of commit messages that could be collected. For example, if one commit change removed a series of outdated configuration properties, then its message can be: "This commit removes the following outdated properties: <list of removed items>." Rule-based methods are good at extracting such patterns and filling in key information by scanning commit diffs [7]. To summarize, we suggest a mixed-methods approach to handle different kinds of commits and different parts of diffs.

5.2 Threats to Validity

We can identify some threats to the validity of this study.

One potential threat relates to the identification of bot-generated messages. To identify the messages generated by bots, we first established an extensive list of known bots through a detailed systematic literature review (see Sec. 3.2.3). Once we identified a list of bots, we could filter out all commits made by those bots. However, it is possible that we missed some bot-generated messages, and thus that our 'New Benchmark' dataset also contains some bot-generated messages. This, however, does not affect the validity of our conclusions, because the goal of the creation of a new benchmark dataset is to demonstrate a critical shortcoming in

TABLE 10
Findings and Implications

Research Question	Findings	Implications
RQ1. To what extent do noisy messages affect the performance of existing automated approaches for commit message generation?	The performance of NNGen and CoRec drops significantly both in the 3 key metrics and human evaluation when 'noisy' messages are removed from the datasets. The performance of FIRA remains stable when evaluated by the metrics but also declined in the human evaluation.	The performance of state-of-the-art approaches as reported in current literature is over-estimated; the observed performance degradation suggests that these approaches are not ready for use in production settings. Future work should rely on more realistic datasets to evaluate approaches. Considerable opportunities to investigate how current approaches can be tailored and customized to understand how they can be improved for use in practice.
RQ2. Given that a commit's diff is usually longer than 100 or 200 tokens, is the performance of existing approaches as promising when considering commits that capture longer code changes?	Prior approaches have been tested with datasets with commits of up to 100 or 200 tokens. However, following the same procedures used for the widely-used J-M dataset, we found that only 5% of commits have a length of no more than 100 tokens, and the median diff length is 632 tokens. After testing three representative approaches on commits with larger diffs, the performance of these degrades significantly.	Existing commit message generation models do not appear ready to meet the needs of realistic development scenarios where commits are frequently over 100 or 200 tokens. Future work should consider more varied datasets containing commits of a more wide-ranging size consideration. Given their ability to handle large amounts of text, large language models and key feature extraction of diffs are worth investigating (see also RQ4).
RQ3. Do developers think generating commit message subjects is useful?	Writing the subject of a commit message is seen as the hardest part to write, and in some cases, it may also be the most time-consuming. 52% of developers are positive towards automatically generating message subjects.	Generating commit messages (even just the first sentence of a message) is of great practical value but has a long way to go to meet practical needs. Future work can be on improving the performance of generating subjects for commits. Meanwhile, automatically generating full commit messages is of practical importance and a fruitful avenue to investigate.
RQ4. What is the performance of LLMs in generating realistic commit messages?	Although LLMs can take larger diffs as input, there is a significant gap when leveraging LLMs to generate messages for real-world code changes.	LLMs cannot be directly used to address the long diff and full messages challenges faced by the state-of-the-art approaches dedicated to generating commit messages. Further work still needs to improve the performance of LLMs on this task.

the evaluations of the state-of-the-art approaches to generate messages. If some bot-generated messages are left in our new dataset, then that would mean that the shortcoming is even more serious than what we observed. The goal of the study is not to establish exactly the extent of the degradation in performance, but only to establish that there is such a significant drop in performance. A more thorough data cleaning will be needed if future studies reuse our New Benchmark dataset because the use of commit bots is increasing [14]. We also reused a quality-aware classifier to identify trivial messages without describing the reasons for changes and what was changed. The reported accuracy of the classifier we reused is 85% [2], and thus false positives may be possible. Given the complexity of defining and identifying trivial messages, we believe such results are sufficient for our analysis, which was to empirically reveal the impact of uninformative messages on the performance of the state-of-the-art in commit message generation. While not ideal, the incorrect removal of informative messages (i.e., false positives) would not be overly problematic given that our dataset is large and the removed non-informative messages only account for 4.5%.

Another internal threat in our study is the replication of the four evaluated approaches, i.e., NNGen, CoRec, FIRA, and CCRep. We carefully read the original papers [1, 9, 10, 63], focusing on how the original datasets were prepared, and we reused the implementations of the four evaluated approaches from their available replication packages. Notwithstanding, it is not impossible that mistakes were made.

A threat to external validity is that our conclusions are based on the evaluations of four approaches, but may not hold for other approaches; as reported, we identified 14 approaches (see Table 1). We selected these four approaches because they represent the three primary categories of techniques, i.e., retrieval-based, learning-based, and hybrid models. Previous evaluations [1, 10] suggest that they represent the state-of-the-art. The selection of UniXcoder and ChatGPT also faces the same issue.

Finally, the findings are based on a dataset that we prepared, which only includes Java projects hosted on GitHub. This may influence the generalizability of our findings, because commit messages are the result of developers' behavior, and the population of developers active on GitHub is not necessarily representative of all developers worldwide. Further, while Java is one of the most popular programming languages, and the evaluation of current approaches to commit message generation is based on the commit data from Java projects, how they perform in other programming languages is one potential strand for future work.

6 CONCLUSION

Commit messages carry important information that is helpful for developers to understand and improve a code base in software development. Several automated generation approaches of commit messages have been proposed. However, we revealed three significant shortcomings in the area of commit message generation: their datasets contain noisy data (commits submitted by bots or with non-informative messages), the datasets are limited to 'small' diffs, i.e. a maximum of 100 (or 200) tokens; current approaches only generate the message subject, not the message body. In this study, we first conduct an empirical investigation on

This article has been accepted for publication in IEEE Transactions on Software Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSE.2024.3364675

18

the impacts of these three threats on the state-of-the-art approaches' performance using three widely-used metrics: BLEU, ROUGE-L, and METEOR, and developer survey. In the last, we explored whether LLMs can address the shortcomings. The study findings emphasize the necessity of data cleaning before generating commit messages, convey the extent the state of the art of generating messages can achieve in a more realistic scenario, can assist researchers in realizing the needs and expectations of developers towards commit message generation, and demonstrate considerable work of leveraging LLMs to address these shortcomings. To facilitate future investigation towards commit messages, we open the datasets, scripts, survey, and results in the online appendix [52].

ACKNOWLEDGMENTS

REFERENCES

[1] 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 *33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*. ACM, 2018, pp. 373–384.

[2] Y. Tian, Y. Zhang, K.-J. Stol, L. Jiang, and H. Liu, "What makes a good commit message?" in *44th IEEE/ACM International Conference on Software Engineering*. Pittsburgh, PA, USA: ACM, 2022.

[3] 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 International Conference on Software Engineering*. IEEE Computer Society, 2013, pp. 422–431.

[4] M. Nassif and M. P. Robillard, "Revisiting turnover-induced knowledge loss in software projects," in *IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2017, pp. 261–272.

[5] R. Buse and W. R. Weimer, "Automatically documenting program changes," in *ASE 2010, 25th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2010, pp. 33–42.

[6] M. Linares-Vásquez, L. F. Cortés-Coy, J. Aponte, and D. Poshyvanyk, "Changescribe: A tool for automatically generating commit messages," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2. IEEE, 2015, pp. 709–712.

[7] 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*, vol. 1. IEEE, 2016, pp. 103–112.

[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*. IEEE, 2017, pp. 135–146.

[9] H. Wang, X. Xia, D. Lo, Q. He, X. Wang, and J. Grundy, "Context-aware retrieval-based deep commit message generation," *ACM Transactions on Software Engineering and Methodology*, vol. 30, no. 4, 2021.

[10] 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 *44th International Conference on Software Engineering*, 2022, pp. 970–981.

[11] Y. Huang, Q. Zheng, X. Chen, Y. Xiong, Z. Liu, and X. Luo, "Mining version control system for automatically generating commit comment," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2017, pp. 414–423.

[12] 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, no. 7, p. 198, 2022.

[13] S. Liu, C. Gao, S. Chen, L. Y. Nie, and Y. Liu, "ATOM: commit message generation based on abstract syntax tree and hybrid ranking," *CoRR*, vol. abs/1912.02972, 2019. [Online]. Available: http://arxiv.org/abs/1912.02972

[14] T. Dey, S. Mousavi, E. Ponce, T. Fry, B. Vasilescu, A. Filippova, and A. Mockus, "Detecting and characterizing bots that commit code," in *17th International Conference on Mining Software Repositories*. ACM, 2020, pp. 209–219.

[15] S. Xu, Y. Yao, F. Xu, T. Gu, H. Tong, and J. Lu, "Commit message generation for source code changes," in *Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019, pp. 3975–3981.

[16] A. Mockus, R. T. Fielding, and J. Herbsleb, "A case study of open source software development: the apache server," in *22nd International Conference on Software engineering*, 2000, pp. 263–272.

[17] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large language models for software engineering: Survey and open problems," *arXiv preprint arXiv:2310.03533*, 2023.

[18] Z. Liu, Z. Tang, X. Xia, and X. Yang, "Ccrep: Learning code change representations via pre-trained code model and query back," in *45th International Conference on Software Engineering*. IEEE Press, 2023, p. 17–29.

[19] B. Lin, S. Wang, Z. Liu, Y. Liu, X. Xia, and X. Mao, "Cct5: A code-change-oriented pre-trained model," *arXiv preprint arXiv:2305.10785*, 2023.

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

[21] T. Hoang, H. J. Kang, D. Lo, and J. Lawall, "Cc2vec: Distributed representations of code changes," in *ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 518–529.

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

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

[24] 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*. IEEE, 2019, pp. 299–309.

[25] L. Y. Nie, C. Gao, Z. Zhong, W. Lam, Y. Liu, and Z. Xu, "Coregen: Contextualized code representation learning for commit message generation," *Neurocomputing*, 2021.

[26] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," *arXiv preprint arXiv:2109.00859*, 2021.

[27] J. Dong, Y. Lou, D. Hao, and L. Tan, "Revisiting learning-based commit message generation," in *2023 IEEE/ACM 45th International Conference on Software Engineering*. IEEE, 2023, pp. 794–805.

[28] K. K. Chahal and M. Saini, "Developer dynamics and syntactic quality of commit messages in oss projects," in *IFIP International Conference on Open Source Systems*, vol. 525. Springer, 2018, pp. 61–76.

[29] J. Li and I. Ahmed, "Commit message matters: Investigating impact and evolution of commit message quality," in *IEEE/ACM 45th International Conference on Software Engineering*, 2023, pp. 806–817.

[30] U. Farooq and J. Grudin, "Human-computer integration," *interactions*, vol. 23, no. 6, pp. 26–32, 2016.

[31] M.-A. Storey and A. Zagalsky, "Disrupting developer productivity one bot at a time," in *2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 928–931.

[32] L. Erlenhov, F. G. D. O. Neto, and P. Leitner, "An empirical study of bots in software development: Characteristics and challenges from a practitioner's perspective," in *28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 445–455.

[33] V. Alizadeh, M. A. Ouali, M. Kessentini, and M. Chater, "Refbot: Intelligent software refactoring bot," in *IEEE/ACM International Conference on Automated Software Engineering*, 2019, pp. 823–834.

[34] M. Wyrich and J. Bogner, "Towards an autonomous bot for automatic source code refactoring," in *IEEE/ACM 1st International Workshop on Bots in Software Engineering*, 2019, pp. 24–28.

This article has been accepted for publication in IEEE Transactions on Software Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSE.2024.3364675

19

[35] M. Wessel, I. Steinmacher, I. Wiese, and M. A. Gerosa, "Should I stale or should I close? an analysis of a bot that closes abandoned issues and pull requests," in *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering*, 2019, pp. 38–42.

[36] M. Wyrich, R. Ghit, T. Haller, and C. Müller, "Bots don't mind waiting, do they? comparing the interaction with automatically and manually created pull requests," in *IEEE/ACM Third International Workshop on Bots in Software Engineering*, 2021, pp. 6–10.

[37] B. Xu, Z. Xing, X. Xia, and D. Lo, "Answerbot: Automated generation of answer summary to developers' technical questions," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering*, 2017, pp. 706–716.

[38] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *33rd International Conference on Software Engineering*, 2011, pp. 481–490.

[39] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang, "Learning to log: Helping developers make informed logging decisions," in *IEEE/ACM 37th International Conference on Software Engineering*, 2015.

[40] X. Wu, W. Zheng, X. Xia, and D. Lo, "Data quality matters: A case study on data label correctness for security bug report prediction," *IEEE Transactions on Software Engineering*, vol. 48, no. 7, pp. 2541–2556, 2021.

[41] S. Jiang and C. McMillan, "Towards automatic generation of short summaries of commits," in *2017 IEEE/ACM 25th International Conference on Program Comprehension*. IEEE, 2017, pp. 320–323.

[42] Y. Chang, X. Wang, J. Wang, Y. Wu, K. Zhu, H. Chen, L. Yang, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *arXiv preprint arXiv:2307.03109*, 2023.

[43] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *arXiv preprint arXiv:2308.10620*, 2023.

[44] I. Ozkaya, "Application of large language models to software engineering tasks: Opportunities, risks, and implications," *IEEE Software*, vol. 40, no. 3, pp. 4–8, 2023.

[45] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "Unixcoder: Unified cross-modal pre-training for code representation," *arXiv preprint arXiv:2203.03850*, 2022.

[46] OpenAI, "Chatgpt," https://openai.com/blog/chatgpt/, 2022.

[47] C. D. Manning, *Introduction to Information Retrieval*. Syngress Publishing, 2008.

[48] 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*. Association for Computational Linguistics, 2020, p. 1536–1547.

[49] W. Jiao, W. Wang, J.-t. Huang, X. Wang, and Z. Tu, "Is chatgpt a good translator? a preliminary study," *arXiv preprint arXiv:2301.08745*, 2023.

[50] N. M. S. Surameery and M. Y. Shakor, "Use chat gpt to solve programming bugs," *International Journal of Information Technology & Computer Engineering*, vol. 3, no. 1, pp. 17–22, 2023.

[51] B. A. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," 2007, technical report, version 2.3.

[52] Y. Zhang, Z. Qiu, K.-J. Stol, W. Zhu, J. Zhu, Y. Tian, , and H. Li, "Appendix to 'automatic commit message generation: A critical review and directions for future work'," https://github.com/qiuzhiqing999/TSE, 2023.

[53] H. Nikoo, "Writing meaningful commit messages," https://dev.to/yvonnickfrin/a-guide-on-commit-messages-d8n, 2021.

[54] 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*, 2021, pp. 126–136.

[55] D. Roy, S. Fakhoury, and V. Arnaoudova, "Reassessing automatic evaluation metrics for code summarization tasks," in *29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 1105–1116.

[56] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *40th Annual Meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[57] C.-Y. Lin and F. J. Och, "Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics," in *42nd Annual Meeting of the Association for Computational*

Linguistics (ACL-04), 2004, pp. 605–612.

[58] S. Banerjee and A. Lavie, "Meteor: An automatic metric for mt evaluation with improved correlation with human judgments," in *ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, 2005, pp. 65–72.

[59] J. T. Liang, T. Zimmermann, and D. Ford, "Understanding skills for oss communities on github," in *30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY, USA: ACM, 2022, p. 170–182.

[60] Y. Zhang, M. Zhou, K.-J. Stol, J. Wu, and Z. Jin, "How do companies collaborate in open source ecosystems? an empirical study of openstack," in *International Conference on Software Engineering*. New York, NY, USA: ACM, 2020, p. 1196–1208.

[61] E. Smith, R. Loftin, E. Murphy-Hill, C. Bird, and T. Zimmermann, "Improving developer participation rates in surveys," in *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2013, pp. 89–92.

[62] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557–572, 1999.

[63] Z. Liu, Z. Tang, X. Xia, and X. Yang, "Ccrep: Learning code change representations via pre-trained code model and query back," *arXiv preprint arXiv:2302.03924*, 2023.

[64] N. Nachar *et al.*, "The mann-whitney u: A test for assessing whether two independent samples come from the same distribution," *Tutorials in quantitative Methods for Psychology*, vol. 4, no. 1, pp. 13–20, 2008.

[65] C. O. Fritz, P. E. Morris, and J. J. Richler, "Effect size estimates: current use, calculations, and interpretation." *J Exp Psychol Gen*, vol. 141, no. 1, pp. 2–18, 2012.

[66] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.

[67] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[68] "Commit example." https://github.com/oracle/graal/commit/0258b6f84245fc0fca4a952a3003b6dfa56756b3.

[69] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A prompt pattern catalog to enhance prompt engineering with chatgpt," *arXiv preprint arXiv:2302.11382*, 2023.

[70] A. Mockus, "Engineering big data solutions," in *on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014*, J. D. Herbsleb and M. B. Dwyer, Eds. ACM, 2014, pp. 85–99.

[71] F. Tu, J. Zhu, Q. Zheng, and M. Zhou, "Be careful of when: an empirical study on time-related misuse of issue tracking data," in *2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*. ACM, 2018, pp. 307–318.

[72] J. Zhu and J. Wei, "An empirical study of multiple names and email addresses in OSS version control repositories," in *16th International Conference on Mining Software Repositories, MSR 2019, 26-27 May 2019, Montreal, Canada*, M. D. Storey, B. Adams, and S. Haiduc, Eds. IEEE / ACM, 2019, pp. 409–420.

[73] M. Golzadeh, A. Decan, D. Legay, and T. Mens, "A ground-truth dataset and classification model for detecting bots in github issue and pr comments," *Journal of Systems and Software*, vol. 175, p. 110911, 2021.

[74] M. Golzadeh, D. Legay, A. Decan, and T. Mens, "Bot or not? detecting bots in github pull request activity based on comment similarity," in *IEEE/ACM 42nd International Conference on Software Engineering workshops*, 2020, pp. 31–35.

[75] S. Operator, "#491 - url cleanup." 2019.

[76] J. Lin, "Divergence measures based on the Shannon entropy," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 145–151, 1991.

[77] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[78] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *CoRR*, 2022.

[79] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *CoRR*, 2023.