

Commit Messages in the Age of Large Language Models

Cristina V. Lopes

lopes@uci.edu

University of California, Irvine

Irvine, CA, USA

Iris Ma

huaiyao@uci.edu

University of California, Irvine

Irvine, CA, USA

Vanessa I. Klotzman

vklotzma@uci.edu

University of California, Irvine

Irvine, CA, USA

Iftekhar Ahmed

iftekha@uci.edu

University of California, Irvine

Irvine, CA, USA

ABSTRACT

Commit messages are explanations of changes made to a codebase that are stored in version control systems. They help developers understand the codebase as it evolves. However, writing commit messages can be tedious and inconsistent among developers. To address this issue, researchers have tried using different methods to automatically generate commit messages, including rule-based, retrieval-based, and learning-based approaches. Advances in large language models offer new possibilities for generating commit messages. In this study, we evaluate the performance of OpenAI's ChatGPT for generating commit messages based on code changes. We compare the results obtained with ChatGPT to previous automatic commit message generation methods that have been trained specifically on commit data. Our goal is to assess the extent to which large pre-trained language models can generate commit messages that are both quantitatively and qualitatively acceptable. We found that ChatGPT was able to outperform previous Automatic Commit Message Generation (ACMG) methods by orders of magnitude, and that, generally, the messages it generates are both accurate and of high-quality. We also provide insights, and a categorization, for the cases where it fails.

CCS CONCEPTS

- Software and its engineering → Software maintenance tools.

KEYWORDS

Commit message generation, empirical studies, ChatGPT, language models

1 INTRODUCTION

Code is constantly being written and modified on a daily basis. To keep track of these changes, developers use version control systems. In these systems, every alteration is accompanied by a commit message, which is a brief explanation of what the change is. The version control system saves this commit message, along with the “diffs” (differences) between the current version and the previous version of the code. Commit messages and the changes they represent are crucial for understanding and maintaining software. They provide insight into what changes were made and why, which can help developers better understand the design and architectural decisions that were made. Clear and informative commit messages can save software developers time by providing a clear understanding of the

reasoning behind changes. They also aid in tracking the evolution of the codebase over time.

While there is no set rule for what should be included in a commit message, a good commit message should provide context for the changes made, such as the reasons for the change, any related issues, or potential consequences. However, crafting a meaningful commit message takes effort, discipline, and time, which developers may not always have. This can lead to poor quality commit messages, such as messages that are missing, too brief, or simply repeat information found in the diffs. A study by Dyer et al. [14] found that 14% of commit messages in over 23,000 open-source Java projects on SourceForge were empty and 75% had fewer than 16 words. Effective communication among developers is crucial for successful collaborative software development. Commit messages serve as a record of how, by whom, and why the source code of a project has changed. The quality of these messages greatly affects communication among developers, with poor quality messages hindering it.

To improve commit messages, researchers have been exploring the idea of automatically generating them by means of using rule-, retrieval-, and, more recently, learning-based approaches [1, 4, 6, 11, 20, 21, 24, 25, 28–31, 34, 37, 40, 45–47, 54, 56, 57, 59]. Rule-based approaches do not fully account for all situations, and cannot clearly convey the reasoning or intent behind code changes. Retrieval-based approaches tend to produce inaccurate results if there are no similar code changes in the dataset. Learning-based approaches seem to be the most promising ones. Over the past 5 years, there has been a steady stream of work describing supervised machine learning models for Automatic Commit Message Generation (ACMG). All that work has been focused on commit data in isolation: the models were trained on datasets of diffs and the corresponding commit messages. While the results are better than previous approaches, the quality and correctness of the resulting messages still leaves much room for improvement.

The recent emergence of Large Language Models (LLM) trained on extremely large datasets of natural language documents and code is turning our field on its head. The combination of clever architectures (e.g. transformers [3, 55, 60]) and huge amounts of data is creating a generation of pre-trained, relatively general-purpose models that seem to be able to make very accurate predictions of natural language as well as code. LLMs excel at tasks such as completing code and generating code from natural language descriptions [58]. One of the largest models, Codex [5], has been used

Figure 1: Generated Commit Message from ChatGPT

Diff (part of the prompt)
<pre>- a/core/.../CursorToBulkCursorAdaptor.java +++ b/core/.../CursorToBulkCursorAdaptor.java @@ -143,8 +143,7 @@ public final class CursorToBulkCursorAdaptor ... public void close() { maybeUnregisterObserverProxy(); - mCursor.deactivate(); - + mCursor.close(); public int requery(IContentObserver observer, CursorWindow window) {</pre>
ChatGPT's output
<p>Commit message: Fix cursor closing method in CursorToBulkCursorAdaptor</p> <p>Justification: The previous implementation was calling the deactivate() method on the cursor, which is no longer the correct method for closing a cursor in the current version of the Android framework. This change updates the code to use the close() method instead, ensuring that the cursor is properly closed and resources are released.</p>
Original commit message
<p>Call close() instead of deactivate() in CursorToBulkCursorAdaptor.close() This seems like a long-lived copy-and-paste bug. Fixes http://b/issue?id=2015069 This is the master version of change https://android-git.corp.google.com/g/8636 which never made it into Eclair.</p>

in the real-world tool Github Copilot [7] as an assistant for developers that can generate code based on user input. Github Copilot has the ability to produce code in various programming languages, given a specific context known as a prompt, which may include comments, method names, and existing code. Another one of those language models that offers a natural and intuitive interface for humans to interact with is ChatGPT,¹ which was made publicly available at the end of 2022. Since then, it has seen a meteoric rise in interest, some of which related to its capabilities in supporting software engineering tasks.

In a pilot test, we observed that ChatGPT seemed to be able to generate very high-quality commit messages that were descriptive of the changes when presented only with the diffs. Moreover, it also seemed to be able to generate justifications that seemed plausible for the changes. Figure 1 shows an example of a diff² casually entered into a ChatGPT conversation along with the prompt “Generate a commit message for the following diff. Additionally, provide a justification for the changes.” As seen, not only the short message is descriptive, but the justification seems surprisingly plausible, vaguely suggesting that it was due to an API change in the Android framework (“no longer the correct method ... in the current version ...”). ChatGPT’s justification is surprising, because nothing in the input diff refers to the Android framework, much less to an API change. Moreover, the justification provided in the original human-written commit message is different: a simple bug fix. ChatGPT seems to be drawing its prediction from a much broader, and hidden, context than that explicitly provided in the prompt. We posited that maybe the developer forgot to mention that there was an API change, not just a bug, and ChatGPT was able to infer the real reason behind the change.

¹<https://openai.com/blog/chatgpt/>

²https://github.com/allydev/android_frameworks_base/commit/d6dfca8302cc3ccc20113e90e7a65e4aad86fcac

To verify ChatGPT’s justification, we looked at the project where the commit came from, and verified that, indeed, it is using the Android framework. We then looked at Android’s Cursor class³, and observed that the deactivate() method has, indeed, been deprecated. At this point, ChatGPT’s justification was plausible.

But being plausible does not mean that it is true. We then fact-checked ChatGPT’s justification by analyzing the history of the project where this commit was made and the Android framework, cross-checking the dates. It turns out that at the time of this commit (March 2010), the deprecated method was still very much part of Android’s API (level 5, code name ECLAIR); deprecation only happened in July of 2012, several versions later (level 16, code name JELLY_BEAN). ChatGPT’s suggested commit message is high-quality and plausible, but ultimately inaccurate and misleading with respect to the real reason behind the change at the time, as explained by the original developer: a simple bug fix.

The combination of ChatGPT’s very plausible commit messages and our discovery that at least some of them were misleading, or plain wrong, led us to design a more systematic empirical study. The overall objective of the paper is to explore the possibilities of ChatGPT in ACMG and to identify any challenges that need to be addressed to make it a viable solution.

This paper describes and reports the results of our systematic evaluation of ChatGPT for ACMG, comparing it to state-of-the-art research models that have been recently proposed in the literature. As baselines of comparison, we chose NNGen [30], CoDiSum [59], CommitBERT [25], and FIRA [11]. The focus of our work is on qualitative evaluation of the *correctness* of the outputs of all the models with respect to the actual changes, not with respect to the original commit message written by the developers. For ChatGPT, we provide a categorization of the mistakes we were able to identify, and discuss its implications for commit tasks. We aim to answer the following research questions:

- **RQ1:** Is ChatGPT fit for purpose?
- **RQ2:** How does ChatGPT compare to the previous ACMG models?
- **RQ3:** What are the main classes of mistakes in ChatGPT’s generated messages?
- **RQ4:** How do prompts affect the generated commit messages created by ChatGPT?

Our research makes the following contributions:

- As far as we know, this is the first study on the correctness of commit messages with respect to the changes in the code. As powerful AI tools become more prevalent, this change of focus is critical.
- Comparatively, we show that zero-shot ChatGPT outperforms the baselines in all aspects. Moreover, we show that it is capable of generating the rationale behind the changes, which is something the baseline models cannot not do.
- We show that, with respect to the ultimate goal of commit tasks, ChatGPT is a viable model for generating commit messages and rationales that are as good as, and often better, than the human-generated ones. We also show that, when using it as zero-shot predictor, it sometimes makes mistakes, so it must not be blindly trusted.

³<https://developer.android.com/reference/android/database/Cursor>

- We provide an in-depth analysis and a simple categorization of ChatGPT’s mistakes, and suggest prompt engineering mechanisms (*aka* few-shot learning) for dealing with them.

The remainder of the paper is structured as follows. In Section 2, we discuss related work on the different ACMG models proposed. In Section 3, we provide background information on large language models. In Section 4, we provide background information on ChatGPT. In Section 5, we outline our experiment design. Our experiment results are presented in Section 6. In Section 7, we discuss our findings. Section 8 highlights the threats of validity to our study. Finally, in Section 9, we summarize our main findings and conclude the paper.

2 RELATED WORK

This section provides an overview of the different methodologies proposed in the literature for auto-generating commit messages.

Rule-based: DeltaDoc [4] employs templates such as “do X Instead of Z” to generate commit messages based on the control flow of the program between code versions. ChangeScribe [6, 54] proposes templates using method stereotypes [13] and commit stereotypes [12]. Shen [45] used predefined templates to automatically generate commit messages. In general, the template-based have weak capability in describing the rationale and purpose of code changes. These templates are a specific format for the generated commit message, and can include specific metadata from the code revision. However, these approaches do not fully account for all situations or clearly convey the reasoning or intent behind code changes.

Retrieval-based: Retrieval-based methods leverage information retrieval techniques to adopt existing commit messages from code changes. For example, given a code change as a query, Liu [30] uses a nearest neighbor algorithm to select the most similar code change from the training set. The training diff with the highest BLEU-4 score is regarded as the nearest neighbor of the new diff. Similarly, Huang [20, 21] uses both syntax and semantic similarity as the similarity metric. However, retrieval-based techniques are inaccurate when there are no similar code changes in the retrieved database, as they can only output existing commit messages instead of generating new ones.

Learning-based: These methods have the capability to make good predictions for commit messages given source code diffs, and have shown promising results. They have become popular for solving the problem of generating commit messages. Works such as Jiang et al. [24], Loyola et al. [31], Liu et al. [28], Xu et al. [59], and Pavel et al. [37] have used neural machine translation to generate commit messages. Awad et al. [1] used hidden markov models, while Jung [25] and Nie [34] use transformers. Despite their success, these methods tend to favor common words and produce less readable output, as well as suffering from exposure bias and that they generate high-frequency words but ignore low-frequency ones.

Hybrid methods: Liu et al. [29] propose using an abstract syntax tree to represent code changes and integrating both retrieved and generated messages through a hybrid ranking system. This system prioritizes the most accurate message from both retrieved and generated messages for a specific code change. Wang et al. [57] put forward a model that combines the advantages of retrieval-

and learning-based methods for ACMG. Their approach is capable of mitigating the exposure bias. Shia [46, 47]proposes a novel exemplar-based neural commit message generation model, which uses similar commit messages as examples to guide the neural network to generate informative and readable commit messages. Dong et al. [11] proposed a new method for automatically generating commit messages for code changes. They represent the changes using detailed graphs and use machine learning to generate the messages. However, these methods still have limitations, particularly when compared to other methods such as those based on templates, retrieval, or learning. There is ongoing research in the area of combining different models to improve performance. Generating commit messages automatically is a complex task, as it involves converting structured code into natural language. There are many challenges involved, such as selecting the appropriate translations for words with multiple meanings, properly transliterating or translating named entities, and adapting the system to different domains and variations in language.

3 BACKGROUND ON LLMS

Large Language models (LLMs) trained on very large datasets have shown outstanding results in natural language processing tasks [39, 55]. As they become larger, they become more costly to train and update. This can make it difficult to adjust the model’s behavior in real-world applications, particularly when it comes to factual knowledge [17, 38, 50] that plays a critical role in the model’s performance. Petroni et al. [39] used LAMA probing to demonstrate that BERT models [9] function as knowledge bases by storing factual world knowledge. Roberts et al. [9] establishes similar behavior for T5 models. However, since most factual knowledge is constantly changing, the stored information may be outdated or not include new facts, leading to incorrect or poor prediction [26, 35]. The latest trend is to build on pre-trained LLMs and fine-tune them with specific tasks in mind. For example Codex [5] builds on GPT-3 [3] for the purposes of predicting source code given natural language prompts. A version of Codex powers GitHub’s Copilot [7].

Mistakes of LLMs: LLMs have been effective for various tasks, but are also known for producing hallucinated content [23]. Hallucination refers to generating plausible statements that are factually incorrect. They also suffer from temporal misalignment [33]. Temporal misalignment refers to when training and evaluation datasets are drawn from different time periods. Sobania et al. [49] conducted a study evaluating ChatGPT for bug fixing and found that it generated different responses based on the request. This is a result of the model’s abstracted output, which can lead to hallucinations, a common issue with advanced NLP models. A study by Dzri et al. [15] showed that standard benchmarks contain over 60% of false responses from these models, with some amplifying the hallucination. Additionally, LLMs have the potential to have bias representation. Chen et al. [5] found that Codex can be prompted in ways that generate racist, derogatory, and otherwise harmful outputs as code comments and found that code generation models raise further bias and representation issues beyond problematic natural language.

Prompt Engineering: Prompts provide a natural and intuitive interface for humans to interact with and utilize language model systems. It is a widely-used generic method for NLP tasks [3, 43, 44].

Denny et al. [8] conducted a study of prompting GitHub Copilot to solve a series of programming questions. Copilot is able to successfully solve around half of the problems on the first attempt. It is able to solve the remaining 60% of the questions, with changes to the prompt description. Changing the form of the question that is asked of Copilot can influence the correctness of the answer provided. Ross et al. [42] came up with a prototype of the Programmer’s Assistant which integrates a chatbot with the code editor. The prompt needs to be engineered to get specific class of result. LLMs need to be prompted carefully, either manually [41] or automatically [48, 61], as models don’t comprehend the prompts in the same manner as humans do.

4 CHATGPT

ChatGPT is a chatbot developed by OpenAI that has been made publicly available on November 30, 2022. ChatGPT builds on OpenAI’s LLM GPT-3 [3]. GPT-3’s main objective is to make accurate predictions of natural language, and ChatGPT is an impressive demonstration of those accurate predictions. However, ChatGPT *seems* to include much more than GPT-3. Specifically, it *seems* to have also been trained to predict software languages of many kinds (perhaps using Codex), as well as mathematical and scientific languages. *Supposedly*, these additional trainings were made as fine-tunings of GPT-3. Recent news articles⁴ also suggest that a large number of human knowledge workers have been used to generate training data.

The emphasized words of the previous sentences reflect a serious problem that we, as researchers, are facing: we do not know how exactly ChatGPT has been trained, and what techniques might have been used to make it seem like a qualitative leap from the previous generation of language models. Unfortunately, LLMs carry significant costs that no research institution has been, or will be, able to cover. The few papers published by OpenAI about GPT-3, Codex, etc. give some general indications of how it works, but the real power of ChatGPT *seems* to come from the brute-force approach to solving problems: increase the number of machines for training and holding the weights, and increase the number of people for generating training data. For now, OpenAI offers access to their trained models’ output through a human-facing Web interface and a machine-facing Web API; that is what we used in this work.

We do not know if ChatGPT’s training data included datasets of diffs and commit messages, explicitly, and whether its base model was fine-tuned for the specific task of generating commit messages from diffs. It is clear, though, that ChatGPT is capable recognizing the syntax of diffs, explaining them back, and generating commit messages for them.

5 EXPERIMENT DESIGN

5.1 Baselines

Our baselines are relatively small models trained specifically to predict commit messages from diffs, namely: NNGen [30], CoDiSum [59], CommitBERT [25], and FIRA [11]. These models were selected as baselines because they have available datasets and/or source code,

⁴<https://time.com/6247678/openai-chatgpt-kenya-workers/>

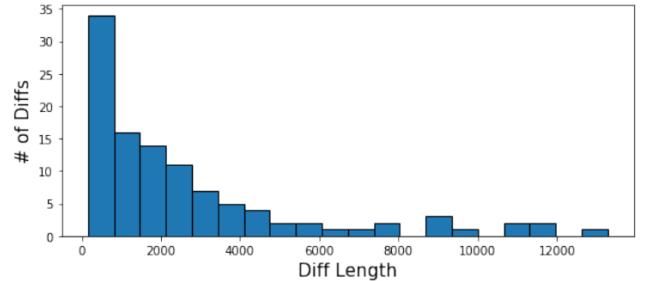


Figure 2: Distribution of diff size, measured in non-blank characters.

which allows us to access or generate data for our qualitative analysis.⁵ Each of these models comes with their own datasets of input (some representation of diffs) and output (commit messages). In assessing the baselines, we used random samples of their own datasets.

A major difference between ChatGPT and the baselines of our study is that the baselines are relatively small models that can be trained by researchers in typical University computing infrastructures, while ChatGPT cannot. Another difference is that we know that these baselines have been specifically trained to generate commit messages from diffs, and we do not know if that happened for ChatGPT. Given these asymmetries of information, it is perhaps unfair to compare them directly. Nevertheless, it is important to study what ChatGPT can do with respect to commit messages, as well as its problems, and compare it to simpler models.

5.2 Dataset

To evaluate ChatGPT, we compiled a dataset of 108 diffs and their original commit messages. For a very large population of code changes, 108 samples places our analysis on a confidence level of $95\pm9.4\%$ [22]. The samples were randomly selected from two prior datasets, namely the dataset associated with Tian et al.’s work [53] and from within the projects used by CommitBERT [25]. Figure 2 shows the histogram of the size of the diffs, measured as the number of non-blank characters.

To obtain the generated commit messages from ChatGPT, we input the prompt “Generate a commit message for the following diff and add a justification for the change:” followed by the specific diff being evaluated. Every prompt with a diff is entered in a freshly created chat session, in order to avoid building up context from diff to diff. With this, we have 108 samples of diffs, their original commit messages, and the commit messages generated by ChatGPT.

5.3 Methods

The following describes the methods we used for answering the three research questions.

⁵We were unable to use CoRec [57] and RACE [46, 47] due to errors in the code and lack of response from the authors. Additionally, attempts at using the Revisiting Learning-based Commit Message Generation (presented at ICSE 2023) were unsuccessful as the authors did not provide the preprint material.

5.3.1 RQ1: Is ChatGPT fit for purpose? One problem of these modern chat bots is that, given the exact same prompt, they do not generate the exact same reply. In order to check whether ChatGPT is capable of functioning as a reliable recommender of commit messages, we performed a competency test consisting of two parts: (1) an expertise test, with a line of questioning about commits, diffs, and examples of those; and (2) a consistency test where we gave it the exact same 10 diff prompts 10 different times using new chat sessions each time (so, no context); we then measured the similarity of its replies using BLEU, ROUGE-L, METEOR, and a BERT-based semantic similarity metric.⁶ We also performed a qualitative manual inspection.

5.3.2 RQ2: How does ChatGPT compare to the previous ACMG models? The quantitative evaluation of the baseline models' accuracy is taken from their original papers. For ChatGPT, we calculate BLEU [36], ROUGE-L [27], and METEOR [2].

For the qualitative evaluation, we take a random sample of 100 generated commit messages from all the baseline models combined (20 each, for a combined confidence level of $95\pm9.8\%$), as well as 108 ChatGPT's messages, and manually analyse them in four dimensions:

- **Semantic similarity:** whether the generated messages convey the same general information as the original commit messages.
- **Quality:** whether the generated messages contain a description of *what* changed and *why* [53].
- **Correctness:** whether the generated messages describe facts inferred from the corresponding diffs.
- **Improvement:** whether the generated messages are qualitative better than the original ones.

The classifications were made by some of the authors, together. We then performed an external validation of our judgements using the Delphi method [19], which relies on using a pool of external experts to arrive at general conclusions. The criteria for expertise consisted of having used source control systems for at least 2 years. The Delphi method was implemented in the form of a survey deployed in a class of 41 computer science master students, of which 10 passed the inclusion criteria for expertise. The survey consisted of 5 questions each containing a randomly selected diff from the 200 samples in our study, along with the original message and the generated message. The experts were asked to judge whether the generated message was semantically similar to the original, whether it had quality, and whether it correctly reflected the diff. The survey included the definitions for these words. One of us, who was not involved in the qualitative evaluation, served as the moderator of this Delphi process. One round of the Delphi method was enough to converge, since for all $5 \times 3 = 15$ judgements, only one of the authors' judgments (related to semantic similarity) differed from the judgement of the majority of external experts.

5.3.3 RQ3: What are the main mistakes in ChatGPT's generated messages? We identify the mistakes ChatGPT makes in its generated messages by manually inspecting the 108 diffs and the original commit messages.

⁶<https://github.com/AndriyMulyar/semantic-text-similarity>

We analyzed the commit messages incrementally, that is, we qualitatively analyzed the commit message, following the open coding protocol [18]. Two of us performed the analysis independently, coding whether there was mistake made by ChatGPT; if a mistake was identified, then we labeled it. This resulted in codes regarding types of mistakes ChatGPT made. In several cases, we also analyzed the project of origin and its history. When doing the analysis, each emerging code was compared with the existing codes to determine if the emerging code was a discrete category or a subset of an existing code. We carried out the whole procedure via continuous comparison throughout the coding sessions and through negotiated agreement [16]. We discussed the rationale for applying specific codes, and reached a consensus during the negotiated agreement process. Two authors agreed on (1) whether it was a mistake, and (2) definition of the mistake.

5.3.4 RQ4: How do prompts affect the generated commit messages created by ChatGPT? The following is an example of our baseline prompt:

```
generate a commit message for the following diff, and add
a justification for the change:
diff --git a/pom.xml b/pom.xml
index 6fdbbe7dc1cd..c62655cb229 100644
--- a/pom.xml
+++ b/pom.xml
@@ -57,7 +57,7 @@
     <okio.version>1.6.0</okio.version>

     <!-- Test Dependencies -->
-    <junit.version>4.11</junit.version>
+    <junit.version>4.12</junit.version>
</properties>

<scm>
```

We experimented with another zero-shot prompt that consisted of a long introductory text about the syntax of diffs, and compared the results with the baseline prompt. We also experimented with introducing omissions in the diffs, to check whether ChatGPT's messages would degrade.

Finally, we experimented with two-shot prompts: for the generated messages that were misleading or wrong, we followed up the conversation with second prompts that were carefully crafted for the type of mistake we identified, according to the categories of RQ2.

6 EXPERIMENT RESULTS

6.1 RQ1: ChatGPT's Fitness for Purpose

The following is the chat log of the expertise test we performed on ChatGPT (CG):

```
Us>> Explain what a commit is in the context of source code
version control.
```

CG>> A commit in the context of source code version control is a record of changes made to a set of files in a repository. Commits are usually made to a local repository, which is then pushed to a remote repository for sharing and collaboration. Each commit represents a new state of the codebase, and contains a message that describes the changes made. Commits serve as a way to keep track of the evolution of the codebase over time, and to allow developers to revert to previous versions if necessary. Commits also allow developers to collaborate on the codebase by merging their changes with those made by others.

Us>> And what is a diff?

CG>> A diff is a representation of the difference between two sets of code, typically two versions of the same file or two different files. A diff is usually shown in a human-readable format that highlights the lines that have been added, removed, or changed between the two versions of the code. The diff is used in version control systems to track changes to code over time and to help resolve conflicts that can arise when multiple people make changes to the same codebase.

Us>> Give me an example of a diff.

CG>> Here's an example of a diff:

```
--- a/file.txt
+++ b/file.txt
@@ -1,3 +1,5 @@

```

This is the original content of file.txt.

-Here's a line that's been removed.

+Here's a line that's been added.

And here's another line that's been added.

The --- a/file.txt and +++ b/file.txt indicate the original and modified versions of the file, respectively. The lines starting with @@ show the line range of the changes, with the - symbol indicating removed lines and the + symbol indicating added lines.

Us>> It seems you got the last line wrong. That line does not start with a `+`

CG>> I apologize, that was an error on my part. Here's a corrected example:

```
--- a/example.txt
+++ b/example.txt
@@ -1,3 +1,3 @@

```

This is an example file.

-It has two lines.

+It has three lines.

And this is the third line.

Us>> Very good. What makes a good commit message?

CG>> A good commit message should be concise and descriptive, explaining the purpose of the changes made in the commit. It should include the following elements:

- A brief summary of the changes, using imperative tense (e.g. "Fix bug" rather than "Fixed a bug").

- Optionally, a more detailed explanation of the changes, providing context and reasoning for the changes made.
- A reference to any relevant issues or tickets, if applicable.

Good commit messages are helpful for other contributors to understand the history of the project and make it easier to revert to a previous state if necessary. A clear and descriptive commit message also makes it easier to maintain a project over time.

Us>> Suggest a commit message for the following diff:

```
diff --git a/pom.xml b/pom.xml
index 6fdb7dc1cd..c62655cb229 100644
--- a/pom.xml
+++ b/pom.xml
@@ -57,7 +57,7 @@
<okio.version>1.6.0</okio.version>
<!-- Test Dependencies -->
- <junit.version>4.11</junit.version>
+ <junit.version>4.12</junit.version>
</properties>
<scm>
```

CG>> "Update JUnit version to 4.12 in pom.xml"

Although ChatGPT made one mistake, it corrected itself when the mistake was pointed out. The conversation made it clear that its training was enough for it to demonstrate knowledge about commits, diffs, and commit messages.

After this, we performed a consistency test of 10 diff prompts, each repeated 10 times, and measured the similarity of ChatGPT's suggested commit messages for each prompt. Table 1 shows the results. The numbers represent the similarity within the 10 instances of the 10 prompts. For example, prompt 1's 10 instances scored an average of 48 on BLEU, 32 on ROUGE-L, 30 on METEOR, and 64 on the BERT-based semantic similarity metric. This last metric is particularly meaningful, as it transforms words into BERT encodings that are then compared in that semantic space. These numbers show a very high syntactic and semantic similarity of ChatGPT's responses to each of the same prompts, a signal that the messages are not completely inconsistent.

We further inspected the messages manually, and confirmed that, in general, the different responses to the same prompt convey the same general idea, even when the sentences and their ordering are different. For example, in 4 out of 10 cases, we observed that it picked a particular point to be the lead short description, and that choice was different; but, if it was not in the lead sentence, that point was present in the longer justification, so the overall semantics of the generated text was the same.

RQ1: Given these results, we conclude that ChatGPT is fit for the purpose of generating commit messages, because it demonstrates knowledge of the material, and its generated messages, although always different, demonstrate acceptable semantic consistency within its non-deterministic behavior.

Table 1: Consistency Test (%Similarity)

Prompt	BLEU	ROUGE-L	METEOR	BERTSim
1	48	32	30	64
2	63	47	48	74
3	55	49	43	74
4	56	41	40	71
5	51	37	34	68
6	46	32	29	64
7	55	41	39	74
8	47	34	32	64
9	46	33	39	67
10	54	42	38	76

Table 2: Quantitive Results (%Similarity)

Model	BLEU	ROUGE-L	METEOR
NNGEN	16.42	NR	NR
CoDiSum	16.55	9.73	12.83
CommitBERT	13.32	NR	NR
FIRA	17.67	21.58	14.93
ChatGPT	22.8	20.8	15.19

Table 3: Qualitative Results (%)

	Sim	Qual	Corr	Improv	Med
Baselines	21	0	13	0	19
ChatGPT	66	100	83	42	381

6.2 RQ2: ChatGPT vs. Baselines

The quantitative results are shown in Table 2. The numbers represent the similarity between each model’s generated messages and the original commit messages in their own datasets. For ChatGPT, we used our dataset of 108 diffs/messages.

According to these results, ChatGPT performs better than the baselines on the BLEU score, slightly better on the METEOR score, and worse than FIRA on the ROUGE-L score. However, all five models are unremarkable, as the scores are relatively low. We will come back to discussing the importance of these metrics after presenting the qualitative results.

Table 3 presents the results of our manual qualitative evaluation of 208 commits and corresponding generated messages, 108 from ChatGPT and 100 from the four baseline models. The columns on this table are: semantic similarity (Sim), quality (Qual), correctness (Corr), and improvement (Improv), as defined in the previous section. The final column, Med, is the median of the length of the generated messages, measured in non-whitespace characters.

The qualitative differences between ChatGPT and the baselines are striking. The generated commit messages from the baseline models are very short (median of 19 characters), and either vague or fail to capture the code changes. Only 21 out of 100 were semantically similar to the original messages. None of them provided a justification for the change (quality) beyond implicit action keywords such as “fix,” which was often incorrect. Worse, only 13 out of 100 were correct, in the sense that they were an accurate summarization of the changes in the diff; 87 were just wrong. Also, none

Table 4: Samples from the baseline models

Original	Generated
Fix TsExtractor tests	Adds lintOptions to Library
Added joscar JAR	Added missing JNA jar
Recycle cached views when clearing the cache.	fix bug in store cached
Bring documentation of Service	fix typo
Change callback defaults for transfers	Fixes typo
Updated version number to match website	Fixes typo
add additional loggin	improve assertion message
fix the title	fix a typo

Table 5: Samples from ChatGPT

Original Internal change	Generated Fix typo in MaskRCNNLosses and improve box loss computation The commit fixes a typo in the variable name "groundturth" in the function signature of call, and changed it to "groundtruth". The commit also improves the computation of the box loss in the _rpn_box_loss method by only considering valid boxes (boxes with at least one coordinate non-zero) in the loss calculation, instead of all boxes. The valid mask is created using the reshaped box_targets, and is used to weight the Huber loss. The loss is then normalized by the sum of non-zero weights and additional normalizer provided by the function caller. This should result in a more accurate and stable box loss calculation.
Original Replaced sort with sorted. ‘sort’ throws an error, so replaced with ‘sorted’ which solves the same purpose	Generated Fix sorting method in various code blocks This diff changes the sorting method used in various code blocks from the “sort” method to the “sorted” method. This is likely done to ensure that the data is always sorted in a consistent manner, as the “sort” method sorts the data in place, while the “sorted” method returns a new sorted list. Additionally, the “sorted” method is a built-in function, which is likely more efficient than a custom sort function.

of them was an improvement over the original commit message. Table 4 shows eight examples of original and generated commit messages of the baseline models.

In contrast, ChatGPT generates complete commit messages and explanations for code changes, as requested by our prompt. The text is grammatically correct, the commit messages are always in good form, and the justifications are always present (quality), resulting in suggestions with a median size of 381 characters. 66% are semantically similar to the original commit messages and 83% of them are correct. Even more surprising is that almost half of them (42%) are better than the original commit messages, and many could be considered better if we overcome some of the problems identified in the next section. Table 5 shows a couple of examples of excellent commit messages generated by ChatGPT.

RQ2: With respect to the baselines, ChatGPT increases the number of correct commit messages 6-fold, and is capable of generating messages that are better than the original ones in 42% of time, something that the baselines cannot do.

After making this large qualitative evaluation of 208 commit messages, we make two final observations:

Poor quality messages. The majority of human-written commit messages we analyzed were short – an observation that has been measured before [14]. Since we looked at the diffs, we found many situations where the commit messages written by developers were incomplete, imprecise, inaccurate, hard to understand, or simply wrong. That is the reason why so many of ChatGPT’s generated messages (42%) were better than the originals – the human-written messages had many problems. That is also one of the motivations behind automatic commit generation research.

Metrics. Referring back to the quantitative results shown in Table 2, we stated that they were unremarkable across the board. They do not correlate with our qualitative analysis, which shows a remarkable qualitative leap of ChatGPT with respect to the baselines. These metrics, which are widely used for measuring the performance of NLP models, do not measure semantic similarity, but simply a very narrow concept of similarity based, for the most part, on equality of words (METEOR includes synonyms, but that is as far as it goes). LLMs like ChatGPT seem to be quite creative with words, and therefore it is unlikely that they predict the exact same words as those chosen by developers on specific situations. These metrics are inappropriate, and this has been pointed out before [10, 32].

These two observations, together, lead us to make a stronger statement about research in ACMG: we cannot rely on non-curated human-written messages as the ground truth. There are cases where the semantic similarity between ChatGPT’s message and the original message is very low, but where ChatGPT is correct and the original message is incorrect, incomplete, or meaningless (see Table 5). Work that uses NLP metrics in a way that optimizes for mimicking the original messages will be misguided by those metrics.

6.3 RQ3: Categories of Mistakes

In spite of its surprisingly good performance, ChatGPT also makes mistakes. As explained before, we followed an open coding protocol to come to a consensus about categories of mistakes. The following are the categories of mistakes that emerged from our open coding protocol:

- **Lack of context:** The message is missing the point, because it lacks project-specific context. For example, links to bug reports or some piece of historical information (e.g. the commit coming after a code review).
- **Truism:** The message (typically the justification) includes statements that are trivially true. For example, “The upgrade ensures that the project has access to the latest bug fixes and features of the library, and helps to keep the dependencies of the project up-to-date.”
- **Perceptual error:** The mistake seems to come from a misunderstanding of the diffs. For example, when the diff is too long and varied, ChatGPT tends to look only at some parts, usually the beginning, and may miss the main change. But it can happen also with very small diffs of just one line change.
- **Hallucination:** The message (typically, the justification) makes too many unsubstantiated statements. For example, it justifies the change as a security issue, when it had

Table 6: Classification of ChatGPT’s Mistakes

Type of mistake	# (%)
Lack of context	38 (35%)
Truism	12 (11%)
Perceptual error	10 (9%)
Hallucination	6 (6%)
Literal interpretation	3 (3%)

nothing to do with security. The motivating example of the introduction is another instance of this mistake.

- **Literal interpretation:** Failure to distinguish code from text, interpreting the words of the text as the changes themselves. This happened for some text (non-code) files: their changed lines described something, for example, a piece of code in documentation, that ChatGPT interpreted as being a change in the code.

Table 6 shows the number of ChatGPT’s generated messages that had these different kinds of problems. Some messages had more than one problem. We noticed that there is a small correlation between lack of context and truism mistakes: when ChatGPT is missing context, it tends to “fill up the space” of the justification with truisms.

RQ3: We identified five major categories of mistakes that seem to explain all of the problems we observed in the generated messages.

6.4 RQ4: Effect of Prompts

In general, the prompts have a significant effect on ChatGPT’s responses. We studied this effect along two dimensions: (1) incomplete zero-shot prompts; (2) two-shot prompts for response improvement.

6.4.1 Incomplete zero-shot prompts. The goal of this part of the study was to analyze how ChatGPT reacted to diffs that were missing seemingly useless information that has been eliminated from the input of some of the baselines. Specifically, NNGen’s input omits the line range of the changes (e.g. “@@ -1,3 +1,5 @@”), and CommitBERT and FIRA use only the added and deleted tokens, throwing away everything else.

We took 42 randomly selected diffs from NNGen’s dataset, and fed them to ChatGPT under our baseline prompt. These diffs were just missing the line ranges, everything else was there. The performance degradation was very visible: in these 42 diffs, the semantic similarity between the generated messages and the original is comparable to the that of the prompts with complete diffs (67%), but the correctness of the generated messages is only 45% (in contrast to 83% on our dataset). Most of the errors are perceptual (26%, in contrast with 9% on our dataset), followed by truisms (24%, in contrast to 11%), followed by hallucinations (21%, in contrast to 6%), followed by lack of context (14%, in contrast to 35%), and finally literal interpretations (7%, in contrast to 3%).

Similarly, we made a few ad-hoc experiments with inputs similar to those used in CommitBERT and FIRA, with only additions and deletions, and nothing else, but it was clear that ChatGPT’s predictions on them were mostly erroneous.

These experiments lead us to the conclusion that ChatGPT did, indeed, “learn” about diffs and is capable of correctly identifying them and of making good predictions of commit messages. But when the supposed diffs do not conform to the form that it “learned,” its performance degrades rapidly.

6.4.2 Two-shot prompts. This part of the study is a feasibility study for an interactive tool that automatically recommends commit messages, but where the problematic recommendations could be corrected by developers. We envision the creation of these messages as an interactive conversation. With that goal in mind, we wanted to find out if, and how, ChatGPT was capable of automatically correcting the problematic commit messages with simple factual feedback from the developer committing the changes.

We selected diffs from each mistake category, for a total of 22 diffs, and we repeated the baseline prompts for them. As expected, ChatGPT gave slightly different responses, but with the same general ideas and the same mistakes. We then prompted it a second time with additional information that might correct the problems. Depending on the type of mistake, that information either pointed out a fact or added context data. Table 7 shows prototypical examples of these second prompts. The following is a concrete example of one of these interactions:

```

US>> Generate a commit message for the following diff and
     provide a justification for the changes:
<the diff>

CG>> Commit Message: Refactor DefaultFuture to improve log
     message for timeout
Justification: The change refactors the DefaultFuture class
     to improve the log message for timeout events by
     removing sensitive data from the request object when
     logging. This change helps to maintain the privacy and
     security of user data by removing it from the log
     messages. The function getRequestWithoutData has been
     added to return a copy of the request object with its
     data field set to null.

Us>> Also, issue #5149, reduce log size. Try again.

CG>> Commit message: Fix #5149, reduce log size in
     DefaultFuture
Justification: This change resolves issue #5149 by reducing
     the size of logs generated by DefaultFuture. The change
     replaces the request object in the timeout message
     with a new request object where the data property is
     set to null, preventing sensitive data from being
     logged. A new method, getRequestWithoutData, has been
     added to return a new request object with data set to
     null.
  
```

The results are shown in Table 8. As shown, after the second prompts, ChatGPT was capable of producing better commit messages that fixed all the problems, but sometimes it introduced new problems that had not been there before.

Before we settled on our second prompts shown in Table 7, we experimented with many approaches for error correction, most of which failed to produce the desired results. Specifically, prompts that tried to say too much failed to lead ChatGPT towards better

Table 7: Examples of Second Prompts

Type of mistake	2 nd Prompt Example
Lack of context	<i>Also, align with latest Spring Data Kay snapshots. And issue #gh-7461. Try again.</i>
Truism	<i>Make it shorter. Try again.</i>
Perceptual error	<i>volatile was also removed. Try again.</i>
Hallucination	<i>Also, issue #5149, reduce log size. Try again.</i>
Literal interpretation	<i>This was just a text file. Try again.</i>

Table 8: Results of ChatGPT’s Mistake Correction

Type of mistake (count)	Fixed	Fixed but new problems	Not fixed
Lack of context (5)	5	0	0
Truism (5)	5	0	0
Perceptual error (5)	2	3	0
Hallucination (4)	2	2	0
Literal interpretation (3)	0	3	0

messages; they seemed to add confusion to its prediction process. The most successful prompts were the very short ones that pointed exactly at what needed to be corrected, added, or removed.

The baseline prompts are always the same, and can be issued automatically upon commit actions. The second prompts, however, require conversation with the developer to gather context and feedback. Given how sensitive ChatGPT is to the exact formulation of prompts, it is not clear that a free-form conversational tool would be successful in practice; developers may prefer to edit the suggested message themselves, and fix them. But this is an empirical question about future tools that falls out of scope of this paper. This part of our study shows that it is possible to lead ChatGPT to correct many of its mistakes, but this requires experience and mastery of the English language.

RQ4: The prompts have a significant effect on ChatGPT’s responses. Small omissions in the diffs lead to a noticeable increase in the number of mistakes. Additionally, in two-shot scenarios designed to produce better commit messages, the exact wording of the second prompt has a significant effect on whether a better message is generated or whether it gets worse. We found that very short commands that address the mistakes directly work better than more verbose prompts that seem to confuse ChatGPT’s predictions.

7 DISCUSSION

We want to come back to the point about metrics. There have been several studies that quantify commits in large datasets [51, 52], and one study that goes qualitatively deeper [53] with respect to the *form* of commit messages. As far as we know, our work is the first study focusing on the *correctness* of commit messages with respect to the respective diffs. We manually inspected almost 108 commits of various sizes and verified whether both the original commit messages and the generated ones truly captured the changes, many times following through to the original projects to better understand the context. This was a time-consuming process, but the findings were enlightening. Independent on our findings about ChatGPT, it was clear that the original commit messages had many problems; except for the trivial changes, it was rare to find a commit message

that accurately summarized the nature of the change. We found many that were truly bad. ACMG work, so far, has focused on measuring and reporting NLP metrics that simply assume that the original commit messages are the gold standard to strive for. Based on what we saw, we believe this is a misguided goal. ACMG should strive to generate commit messages that are much better than those written by developers.

With that more ambitious goal in mind, LLMs such as ChatGPT seem to be the answer to getting there. LLMs are not trained on [bad] examples of commits. Instead, they are trained to predict natural and artificial languages really well. In doing that, they seem to acquire reasoning capabilities that help in solving a large variety of knowledge-based tasks, such as ACMG.

Because generating formally good summaries from diffs is now a reality, the challenge from here on is to detect when the summary is incorrect or incomplete. This ties back with metrics, again. We may have been able to work around poorly-written commit messages from human developers, because, as last resort, we can ask them directly what they meant. But we cannot accept that from automated tools.

8 THREATS TO VALIDITY

The main threat to the validity and repeatability of our findings is the fact that ChatGPT is outside of our control, and cannot be externally kept for archival purposes. For example, on January 30, 2023, during the writing of this paper, OpenAI announced that a new version of ChatGPT had been deployed with “improved factuality and mathematical capabilities.”⁷ To mitigate this, we redid some of our prompts to verify whether some of the problems we identified had been resolved with this update. It appears they have not.

In spite of the high IRR in our classification of ChatGPT’s mistakes, our judgements can be questioned. We mitigate this by making our data available along with the paper. We believe the classifications are non-controversial in the vast majority of cases.

Another threat to validity is the possibility of mistakes in running the scripts to replicate the four ACMG models. To mitigate this threat, we had multiple people check and verify that there were no errors.

9 CONCLUSION

To help programmers write comprehensive commit messages, many automatic message commit models have been proposed. However, these models have limitations. Rule-based models do not cover all scenarios or effectively explain the reasoning behind code changes. Retrieval-based models may be inaccurate if no similar code changes exist in the database, only repeating existing messages instead of generating new ones. Learning-based models often prioritize common words, resulting in less readable messages, and also suffer from exposure bias by generating frequent words and neglecting less frequent ones. ChatGPT, a newly introduced large language model, can generate a recommended commit message by providing a prompt and the related code change.

We show that ChatGPT has much better performance than the ACMG models that have been proposed before. However, some of

the suggested commit messages have problems, such as lack of context and unsubstantiated statements. Experiments using ChatGPT’s dialogue show that it is possible to correct ChatGPT’s mistakes using very simple and direct commands.

Despite its great performance, the question remains whether the mental effort required to verify ChatGPT’s answers outweighs its advantages. Incorporating automated approaches to provide ChatGPT with hints and verifying its responses, for example, by connecting it to the Github repository, could be a useful tool for software developers in their daily tasks. We hope our results and impressions will be helpful for future work with ChatGPT.

LLMs are redefining research in software engineering. Just a few months ago, ACMG looked both an important problem that could benefit from automation, and a hard problem to solve. The machine learning models proposed very recently in the literature were showing a steady, but incremental, progress towards the goal. ChatGPT clearly outperforms them in all dimensions. The reasons for why this happens are worth pondering. Brute force works, and that is something academic researchers cannot do. The question now is not how to develop methods for ACMG, but how to leverage and control LLMs towards specific tasks.

10 DATA AVAILABILITY

All data collected for this study, as well as our statistical analyses, are made available as supplemental material. We intend to make the data and our analysis scripts publicly available upon publication of the paper.

REFERENCES

- [1] Ahmed Awad and Khaled Nagaty. 2019. Commit Message Generation from Code Differences Using Hidden Markov Models. In *Proceedings of the 2019 8th International Conference on Software and Information Engineering* (Cairo, Egypt) (*ICSIE ’19*). Association for Computing Machinery, New York, NY, USA, 96–99. <https://doi.org/10.1145/3328833.3328873>
- [2] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. Association for Computational Linguistics, Ann Arbor, Michigan, 65–72. <https://aclanthology.org/W05-0909>
- [3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. <https://proceedings.neurips.cc/paper/2020/file/1457c0d6fbcb4967418bf8ac142f64a-Paper.pdf>
- [4] Raymond P.L. Buse and Westley R. Weimer. 2010. Automatically Documenting Program Changes. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering* (Antwerp, Belgium) (*ASE ’10*). Association for Computing Machinery, New York, NY, USA, 33–42. <https://doi.org/10.1145/1858996.1859005>
- [5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).
- [6] Luis Fernando Cortes-Coy, Mario Linares Vásquez, Jairo Aponte, and Denys Poshyvanyk. 2014. On Automatically Generating Commit Messages via Summarization of Source Code Changes. , 275–284 pages.
- [7] Arghavan Moradi Dakheel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C Desmarais, Zhen Ming, et al. 2022. GitHub Copilot AI pair programmer: Asset or Liability? *arXiv preprint arXiv:2206.15331* (2022).
- [8] Paul Denny, Viraj Kumar, and Nasser Giacaman. 2022. Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language.

⁷<https://help.openai.com/en/articles/6825453-chatgpt-release-notes>

- <https://doi.org/10.48550/ARXIV.2210.15157>
- [9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. <https://doi.org/10.18653/v1/N19-1423>
- [10] Samanta Dey, Venkatesh Vinayakarao, Monika Gupta, and Sampath Dechu. 2022. Evaluating commit message generation: to BLEU or not to BLEU?. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*. 31–35.
- [11] Jinhao Dong, Yiling Lou, Qihao Zhu, Zeyu Sun, Zhilin Li, Wenjie Zhang, and Dan Hao. 2022. FIRIA: Fine-Grained Graph-Based Code Change Representation for Automated Commit Message Generation. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. 970–981. <https://doi.org/10.1145/3510003.3510069>
- [12] Natalia Dragan, Michael L. Collard, Maen Hammad, and Jonathan I. Maletic. 2011. Using stereotypes to help characterize commits. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. 520–523. <https://doi.org/10.1109/ICSM.2011.6080822>
- [13] Natalia Dragan, Michael L. Collard, and Jonathan I. Maletic. 2006. Reverse Engineering Method Stereotypes. In *2006 22nd IEEE International Conference on Software Maintenance*. 24–34. <https://doi.org/10.1109/ICSM.2006.54>
- [14] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. 2013. Boa: A Language and Infrastructure for Analyzing Ultra-Large-Scale Software Repositories. , 10 pages.
- [15] Nouha Dziri, Sivan Milton, Mo Yu, Osman Zaiane, and Siva Reddy. 2022. On the Origin of Hallucinations in Conversational Models: Is it the Datasets or the Models?. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Seattle, United States, 5271–5285. <https://doi.org/10.18653/v1/2022.naacl-main.387>
- [16] Jane H. Forman and Laura J. Damschroder. 2007. Qualitative Content Analysis.
- [17] Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer Feed-Forward Layers Are Key-Value Memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 5484–5495. <https://doi.org/10.18653/v1/2021.emnlp-main.446>
- [18] Barney G Glaser. 2016. Open coding descriptions. *Grounded theory review* 15, 2 (2016), 108–110.
- [19] Elizabeth A Holey, Jennifer L Feeley, John Dixon, and Vicki J Whittaker. 2007. An exploration of the use of simple statistics to measure consensus and stability in Delphi studies. *BMC medical research methodology* 7 (2007), 1–10.
- [20] Yuan Huang, Nan Jia, Hao-Jie Zhou, Xiang-Ping Chen, Zi-Bin Zheng, and Ming-Dong Tang. 2020. Learning human-written commit messages to document code changes. *Journal of Computer Science and Technology* 35, 6 (2020), 1258–1277.
- [21] Yuan Huang, Qiaoyang Zheng, Xiangping Chen, Yingfei Xiong, Zhiyong Liu, and Xiaonan Luo. 2017. Mining Version Control System for Automatically Generating Commit Comment. In *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '17)*. IEEE Press, Markham, Ontario, Canada, 414–423. <https://doi.org/10.1109/ESEM.2017.56>
- [22] Peter J Huber. 2011. Robust statistics. In *International encyclopedia of statistical science*. Springer, 1248–1251.
- [23] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Andrea Madotto, and Pascale Fung. 2022. Survey of Hallucination in Natural Language Generation. *Comput. Surveys* (nov 2022). <https://doi.org/10.1145/3571730>
- [24] Siyuan Jiang, Amer Armaly, and Collin McMillan. 2017. Automatically Generating Commit Messages from Diffs Using Neural Machine Translation. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2017)*. IEEE Press, Urbana-Champaign, IL, USA, 135–146.
- [25] Tae-Hwan Jung. 2021. CommitBERT: Commit Message Generation Using Pre-Trained Programming Language Model.
- [26] Angeliki Lazaridou, Adhi Kuncoro, Elena Gribovskaya, Devang Agrawal, Adam Liska, Tayfun Terzi, Mai Gimenez, Cyprien de Masson d'Autume, Tomas Kociský, Sebastian Ruder, et al. 2021. Mind the gap: Assessing temporal generalization in neural language models. *Advances in Neural Information Processing Systems* 34 (2021), 29348–29363.
- [27] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, 74–81. <https://aclanthology.org/W04-1013>
- [28] Qin Liu, Zihé Liu, Hongming Zhu, Hongfei Fan, Bowen Du, and Yu Qian. 2019. Generating Commit Messages from Diffs Using Pointer-Generator Network. In *Proceedings of the 16th International Conference on Mining Software Repositories (Montreal, Quebec, Canada) (MSR '19)*. IEEE Press, Los Alamitos, CA, USA, 299–309. <https://doi.org/10.1109/MSR.2019.00056>
- [29] S. Liu, C. Gao, S. Chen, N. Lun Yiu, and Y. Liu. 5555. ATOM: Commit Message Generation Based on Abstract Syntax Tree and Hybrid Ranking. <https://doi.org/10.1109/TSE.2020.3038681>
- [30] Zhongxin Liu, David Lo, Xin Xia, Zhenchang Xing, Ahmed E. Hassan, and Xinyu Wang. 2018. Neural-machine-translation-based commit message generation: how far are we?. In *ASE'18 - Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. Gordon Fraser and Christian Kastner (Eds.). Association for Computing Machinery (ACM), United States of America, 373–384. <https://doi.org/10.1145/3238147.3238190> Automated Software Engineering Conference 2018, ASE 2018 ; Conference date: 03-09-2018 Through 07-09-2018.
- [31] Pablo Loyola, Edison Marrese-Taylor, and Yutaka Matsuo. 2017. A Neural Architecture for Generating Natural Language Descriptions from Source Code Changes. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 2: Short Papers*. Regina Barzilay and Min-Yen Kan (Eds.). Association for Computational Linguistics, 287–292. <https://doi.org/10.18653/v1/P17-2045>
- [32] Congbo Ma, Wei Emma Zhang, Mingyu Guo, Hu Wang, and Quan Z Sheng. 2022. Multi-document summarization via deep learning techniques: A survey. *Comput. Surveys* 55, 5 (2022), 1–37.
- [33] Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Hannaneh Hajishirzi, and Daniel Khashabi. 2022. When Not to Trust Language Models: Investigating Effectiveness and Limitations of Parametric and Non-Parametric Memories. *arXiv preprint arXiv:2212.10511* (2022).
- [34] Lun Yiu Nie, Cuixun Gao, Zhicong Zhong, Wai Lam, Yang Liu, and Zenglin Xu. 2021. CoreGen: Contextualized Code Representation Learning for Commit Message Generation. *Neurocomputing* 459 (2021), 97–107.
- [35] Yasumasa Onoe, Michael J. Q. Zhang, Eunsol Choi, and Greg Durrett. 2022. Entity Cloze By Date: What LMs Know About Unseen Entities. <https://doi.org/10.48550/ARXIV.2205.02832>
- [36] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (Philadelphia, Pennsylvania) (ACL '02)*. Association for Computational Linguistics, USA, 311–318. <https://doi.org/10.3115/1073083.1073135>
- [37] Nicolae-Teodor Pavel and Traian Rebedea. 2021. A Sketch-Based Neural Model for Generating Commit Messages from Diffs. *arXiv:2104.04087 [cs.CL]*
- [38] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language Models as Knowledge Bases?. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 2463–2473. <https://doi.org/10.18653/v1/D19-1250>
- [39] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick S. H. Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander H. Miller. 2019. Language Models as Knowledge Bases?. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, 2463–2473. <https://doi.org/10.18653/v1/D19-1250>
- [40] Sarah Rastkar and Gail C. Murphy. 2013. Why did this code change? , 1193–1196 pages.
- [41] Laria Reynolds and Kyle McDonell. 2021. Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems (Yokohama, Japan) (CHI EA '21)*. Association for Computing Machinery, New York, NY, USA, Article 314, 7 pages. <https://doi.org/10.1145/3411763.3451760>
- [42] Steven I Ross, Michael Muller, Fernando Martinez, Stephanie Houde, and Justin D Weisz. 2023. A Case Study in Engineering a Conversational Programming Assistant's Persona. *arXiv preprint arXiv:2301.10016* (2023).
- [43] Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczęsła, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M Rush. 2022. Multitask Prompted Training Enables Zero-Shot Task Generalization. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=9Vrb9D0W14>
- [44] Timo Schick and Hinrich Schütze. 2020. Exploiting cloze questions for few shot text classification and natural language inference. *arXiv preprint arXiv:2001.07676* (2020).
- [45] Jinfeng Shen, Xiaobing Sun, Bin Li, Hui Yang, and Jiajun Hu. 2016. On Automatic Summarization of What and Why Information in Source Code Changes. *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)* 1 (2016), 103–112.
- [46] Ensheng Shi, Yanlin Wang, Wei Tao, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2022. RACE: Retrieval-Augmented Commit Message

- Generation. <https://doi.org/10.48550/ARXIV.2203.02700>
- [47] Ensheng Shia, Yanlin Wangb, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2022. ECMG: Exemplar-based Commit Message Generation. <https://doi.org/10.48550/ARXIV.2203.02700>
- [48] Taylor Shin, Yasaman Razeghi, Robert IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. 4222–4235. <https://doi.org/10.18653/v1/2020.emnlp-main.346>
- [49] Dominik Sobania, Martin Briesch, Carol Hanna, and Justyna Petke. 2023. An Analysis of the Automatic Bug Fixing Performance of ChatGPT. *arXiv preprint arXiv:2301.08653* (2023).
- [50] Liyan Tang, Tanya Goyal, Alexander R Fabbri, Philippe Laban, Jiacheng Xu, Semih Yahvuz, Wojciech Kryściński, Justin F Rousseau, and Greg Durrett. 2022. Understanding factual errors in summarization: Errors, summarizers, datasets, error detectors. *arXiv preprint arXiv:2205.12854* (2022).
- [51] W. Tao, Y. Wang, E. Shi, L. Du, S. Han, H. Zhang, D. Zhang, and W. Zhang. 2021. On the Evaluation of Commit Message Generation Models: An Experimental Study. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSM-E)*. IEEE Computer Society, Los Alamitos, CA, USA, 126–136. <https://doi.org/10.1109/ICSM-E52107.2021.00018>
- [52] Wei Tao, Yanlin Wang, Ensheng Shi, Lun Du, Shi Han, Hongyu Zhang, Dongmei Zhang, and Wenqiang Zhang. 2022. A large-scale empirical study of commit message generation: models, datasets and evaluation. *Empirical Software Engineering* 27, 7 (2022), 198.
- [53] Yingchen Tian, Yuxia Zhang, Klaas-Jan Stol, Lin Jiang, and Hui Liu. 2022. What makes a good commit message? In *Proceedings of the 44th International Conference on Software Engineering*, 2389–2401.
- [54] Mario Linares Vásquez, Luis Fernando Cortes-Coy, Jairo Aponte, and Denys Poshyvanyk. 2015. ChangeScribe: A Tool for Automatically Generating Commit Messages. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* 2 (2015), 709–712.
- [55] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
- [56] Bei Wang, Meng Yan, Zhongxin Liu, Ling Xu, Xin Xia, Xiaohong Zhang, and Dan Yang. 2021. Quality Assurance for Automated Commit Message Generation. In *28th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2021, Honolulu, HI, USA, March 9-12, 2021*. IEEE, "United States of America", 260–271. <https://doi.org/10.1109/SANER50967.2021.00032>
- [57] Haoye Wang, Xin Xia, David Lo, Qiang He, Xinyu Wang, and John Grundy. 2021. Context-Aware Retrieval-Based Deep Commit Message Generation. *ACM Trans. Softw. Eng. Methodol.* 30, 4, Article 56 (jul 2021), 30 pages. <https://doi.org/10.1145/3464689>
- [58] Frank F. Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A Systematic Evaluation of Large Language Models of Code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming* (San Diego, CA, USA) (MAPS 2022). Association for Computing Machinery, New York, NY, USA, 1–10. <https://doi.org/10.1145/3520312.3534862>
- [59] Shengbin Xu, Yuan Yao, F. Xu, Tianxian Gu, Hanghang Tong, and Jian Lu. 2019. Commit Message Generation for Source Code Changes.
- [60] Biao Zhang, Barry Haddow, and Alexandra Birch. 2023. Prompting Large Language Model for Machine Translation: A Case Study. <https://doi.org/10.48550/ARXIV.2301.07069>
- [61] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910* (2022).

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009