Context Conquers Parameters: Outperforming Proprietary LLM in Commit Message Generation

Aaron Imani
University of California, Irvine
Irvine, USA
aaron.imani@uci.edu

Iftekhar Ahmed
University of California, Irvine
Irvine, USA
iftekha@uci.edu

Mohammad Moshirpour
University of California, Irvine
Irvine, USA
mmoshirp@uci.edu

***Abstract*—Commit messages provide descriptions of the modifications made in a commit using natural language, making them crucial for software maintenance and evolution. Recent developments in Large Language Models (LLMs) have led to their use in generating high-quality commit messages, such as the Omniscient Message Generator (OMG). This method employs GPT-4 to produce state-of-the-art commit messages. However, the use of proprietary LLMs like GPT-4 in coding tasks raises privacy and sustainability concerns, which may hinder their industrial adoption. Considering that open-source LLMs have achieved competitive performance in developer tasks such as compiler validation, this study investigates whether they can be used to generate commit messages that are comparable with OMG. Our experiments show that an open-source LLM can generate commit messages comparable to those produced by OMG. In addition, through a series of contextual refinements, we propose OMEGA, a commit message generation approach that uses a 4-bit quantized 8B open-source LLM. OMEGA produces state-of-the-art commit messages, surpassing the performance of GPT-4 in practitioners' preference.**

***Index Terms*—commit message generation, large language model, llama3, gpt4**

I. Introduction

Commit Messages (**CM**) play a crucial role in documenting changes in version control systems, facilitating maintenance and evolution of the software [1]. Researchers have leveraged Natural Language Processing techniques to develop automatic methods for Commit Message Generation (**CMG**), with the goal of improving the quality of human-written messages [2], [3]. In addition to detailing the changes ("What" information) and their rationale ("Why" information) [2], researchers have identified additional expected criteria from practitioners' perspective [3]. Thus, CMG can be regarded as a complex reasoning task requiring a comprehensive and precise understanding of the commit context, as well as the ability to formulate a CM that meets all the quality criteria developers expect.

Given this complexity, CMG methods have significantly advanced with the introduction of Large Language Models (**LLM**) trained using reinforcement learning from human feedback (RLHF) [4], such as GPT-4 [5]. The enhanced reasoning capabilities of these models have resulted in state-of-the-art performance in CMG [3], [6]. Notably, Li et al. utilized GPT-4, a proprietary LLM with 1.76 trillion parameters [7], to outperform traditional state-of-the-art CMG approaches [3]. They introduced Omniscient Message Generator (**OMG**), a ReAct [8] agent that produces high-quality CMs by leveraging six contextual pieces of information about a commit. GPT-4 is employed in OMG to generate three out of the six contextual pieces of information, and it also serves as the reasoning engine in the ReAct chain. Although OMG produces CM with state-of-the-art quality, its reliance on a proprietary LLM introduces certain organizational and environmental risks.

Third-party proprietary LLM APIs have been introducing privacy risks for companies [9], [10] and research has been done to investigate LLMs' privacy and security implications [11]. For instance, a developer at Samsung accidentally shared sensitive internal source code with ChatGPT [12], leading Samsung to ban the use of all proprietary chatbots due to the difficulty in accessing and deleting shared information [9]. OMG requires sharing sensitive source code information, including the bodies of affected methods and classes before and after the commit, with a third-party API (GPT-4). This introduces privacy risks, making its adoption by the industry problematic and limiting its practical application.

Additionally, researchers have shown that making requests to proprietary LLMs can lead to annual carbon emissions greater than the emission during training such LLMs [13]. OMG makes 12 LLM requests (4 in preparing commit context + 7 Thoughts in ReAct (number of available tools to the Agent) + 1 Initial ReAct Thought) to generate a CM. A study on 24 popular Java repositories has shown that on average, a developer pushes 3 commits daily [14]. Assuming this number holds in industrial cases, a company with 1,000 developers using OMG would generate over a million CMs for a single project annually, resulting in 12 million calls to a proprietary LLM. This makes the wide adoption of OMG a sustainability threat that introduces environmental risks. According to [15], each commit contains approximately 632 tokens on average (excluding additional context required by OMG). With GPT-4's pricing at $10 per million input tokens, this results in 7.584 billion tokens for the adorementioned company and a cost exceeding $75,000. Such expenses present a barrier to the industrial adoption of OMG. These sustainability and financial limitations underscore the need to shift away from a proprietary LLM-based approach.

Open-source LLMs (**OLLM**) offer a cost-effective and privacy-conscious alternative, as they mitigate the privacy concerns associated with proprietary models. Deployable on

1558-1225/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSE55347.2025.00048

local GPUs due to their smaller number of trained parameters, OLLMs are more sustainable for adoption in LLM-based automations [16]. Software engineering researchers have attempted to use OLLMs to address several development tasks. While they have shown promising results in some areas [17], [18], a performance gap has been noted when compared to proprietary LLMs [19], [20].

However, with the advent of new OLLMs that achieve performance on par with proprietary LLMs across various benchmarks and leaderboards [21], the likelihood of their successful adoption for a wide range of tasks is increasing. Since prior work has not investigated the application of OLLMs in generating CMs that meet practitioners' expectations and are able to achieve performance similar to the state-of-the-art CMG technique [3], our study aims to take the first step in replacing GPT-4 in OMG with an OLLM and to assess the feasibility of generating CMs comparable to those produced by OMG. To address this goal, we formulated our first research question.

RQ1: Can an OLLM generate CMs comparable to a state-of-the-art LLM (GPT-4)?

To address our first research question, we experimented with using an OLLM instead of GPT-4 to generate LLM-derived commit context and produce high-quality CMs. We utilized automated machine translation evaluation metrics and practitioner surveys to measure the quality of the generated CMs.

Based on our results, although the OLLM produced CMs with quality comparable to OMG in various aspects, it did not meet practitioners' expectations in one key aspect: Comprehensiveness [3]. This indicates that the OLLM-generated CM missed details that were covered by OMG. To address this shortcoming, we propose our second research question.

RQ2: How can we bridge the comprehensiveness gap between the CMs produced by an OLLM and the CMs generated by a state-of-the-art LLM (GPT-4)?

To answer this RQ, we introduced **C**hange-Based **M**ulti-Intent **M**ethod **S**ummarization (**CMMS**) to provide refined contextual information that bridge the comprehensiveness gap by refining the commit context. Through automated evaluation and a second survey, we examined the refined context's effectiveness in addressing RQ2.

While the improved prompt with the enhanced commit context leads to CMs even closer to those generated by OMG, running the OLLM for inference requires at least an NVIDIA A6000 GPU with 48GB of VRAM. This high resource demand can limit the practical usefulness of our CMG approach for individual developers or smaller teams with limited budgets. Furthermore, adopting an OLLM with fewer trained parameters results in a more sustainable CMG approach by minimizing its carbon emission [16], aligning with one of the primary objectives of this study. Thus, we aimed to explore the possibility of generating comparable CMs using a smaller OLLM (**SLM**) that can run on a local GPU with as little as 8GB of VRAM [22], **reducing the GPU VRAM requirement by 84%**. This investigation is formulated as the third research question of this study.

RQ3: Can a smaller OLLM (SLM) produce CMs comparable to a state-of-the-art LLM (GPT-4)?

To address this research question, we initially employed the same prompt and context as we used to answer RQ2 in an attempt to achieve comparable results. However, as anticipated, the automated scores for the generated CMs by our tested SLMs were considerably lower compared to those produced by the larger OLLM. The poor scores, despite the enhanced context, led us to hypothesize that the SLM is not capable of correctly understanding the underlying changes in a diff. Having validated our hypothesis through an analysis study, we developed two commit diff augmentation techniques, Diff Narrator and **F**ine-grained **I**nteractive **D**iff **EX**plainer **(FIDEX)**, designed to clarify the changes in a diff. As with previous research questions, we used automated metrics and a third practitioner survey to evaluate the effectiveness of our diff augmentation techniques in closing the gap between the commit messages generated by the SLM and those produced by the OLLM.

In summary, our study makes the following contributions:

1) We demonstrate that replacing GPT-4 with an OLLM using the same commit context as OMG produces comparable CMs in all human CM quality evaluation criteria except comprehensiveness.
2) We introduce a new method summarization approach called **C**hange-based **M**ulti-Intent **M**ethod **S**ummarization (**CMMS**) for software engineering tasks that rely on code changes.
3) We propose two augmentation techniques for commit diff, Diff Narrator and **F**ine-grained **I**nteractive **D**iff **EX**plainer **(FIDEX)** that boost SLM's performance in the CMG.
4) We propose the state-of-the-art CMG approach, l**O**cal **M**essag**E** **G**ener**A**tor (**OMEGA**), that employs a 4-bit quantized SLM with 8B trained parameters that runs on a local GPU with as little as 8GB VRAM to generate CMs that are preferred by practitioners over those generated by OMG.

The remainder of this paper is structured as follows. In Section II, we review related research pertinent to our work. In Section III, we detail our methodology for addressing all research questions. Section IV presents the results of our experiments and surveys. In Section V, we highlight potential threats to the validity of our findings and the measures taken to mitigate them. Finally, Section VI concludes our study and outlines potential future work.

II. Related Work

A. Commit Message Generation

Over the past few years, several studies have aimed to improve the state-of-the-art in CMG by exploring various methods to represent the changes in a commit, such as the diff [23], Abstract Syntax Tree (AST) paths [24], [25],

issue states [26], and modification embedding [27]. However, these CMG methods did not account for the impact of bot-generated and uninformative CMs during training, which has been shown to render their reported performance inaccurate [15]. Accordingly, researchers proposed filtering the adopted CM datasets to include only good practice CMs [28] when training the deep learning models. However, these methods relied on the quality of human-written CM, which has been reported to lack the required quality [2], [3], [28], and did not align their generated CM with practitioners' expectations of a good CM [3].

To address these shortcomings in previous CMG methods, Li et al. conducted surveys and data mining to understand practitioners' expectations for a good CM [3]. They proposed an LLM-based CMG approach called OMG that uses the ReAct prompting framework [8] with GPT-4 to meet the identified expectations. Based on their findings, the commit diff, which was the primary artifact used in traditional CMG methods, is not enough to generate a CM that aligns with developers' expectations. Hence, they utilized six different contextual pieces of information about a commit, resulting in CMs that surpassed the quality of those generated by the previous state-of-the-art, FIRA [24], as determined through human evaluation. Despite achieving superior results, the authors did not consider using an LLM that addresses privacy and sustainability concerns. Instead, they employed the most advanced proprietary LLM available during the development of OMG. Our study builds upon OMG in terms of the commit context provided to the model and employing an LLM. However, we seek a different goal. Our objective is to generate high-quality CMs without relying on state-of-the-art proprietary LLMs.

B. OLLMs in Software Engineering

Researchers have investigated the effectiveness of OLLMs in solving various software engineering tasks and compared them with proprietary LLMs, yielding both positive and negative results across different areas [29].

Yin et al. evaluated OLLMs in identifying software vulnerability. They reported that while OLLMs demonstrate some capability in specific areas, they still require further improvement to be truly effective in addressing software vulnerability-related tasks [19]. Pan et al. investigated the effectiveness of LLMs in code translation [20]. Among the evaluated models, including OLLMs and GPT-4, the best performing OLLM, StarCoder, achieved a 14.5% successful translation rate compared to 47.3% by GPT-4.

On the other hand, OLLMs have demonstrated comparable results with GPT-4 in certain areas. In a study by Liu et al, Vicuna 13B equipped with the author's proposed online log analysis approach, LogPrompt, showed comparable performance with GPT-4 [17]. Zhong and Wong [18] inspected the reliability and robustness of the LLM-generated code. They found that although Meta Llama2 achieved a low API misuse rate, its compilation rate was reported significantly lower than the other models in the study. However, instruction-tuned Deepseek-Coder 6.7B achieved comparable results in terms of the balance between compilation rate and API misuse rate. Munley et al. explored various LLMs' capabilities in generation of a validation and verification test suite for high-performance computing compilers from a standard specification. Their results indicated the instruction-tuned Deepseek-Coder 33B produced the most passing tests followed by GPT-4-Turbo [30].

Given the varied performance of OLLMs across different software engineering tasks, it is essential to investigate their ability to generate CMs. Our study aims to fill this gap in existing research through investigating the potential of OLLMs in producing high-quality CMs that meet practitioners' expectations.

III. Methodology

In this section, we present our methodology in answering our research questions. Figure 1 presents the changes we made to various aspects of OMG in answering each research question.

A. Base Commit Context

In addition to the commit diff, we utilized six different contextual pieces of information about a commit that were proposed and utilized by OMG [3]. Specifically, for each commit, we provided the following contextual information to the OLLM/SLM: 1) Associated issues on the version control system or issue tracking service 2) Associated pull requests 3) Relative importance of changed files 4) The software maintenance activity type of the commit 5) Multi-Intent Method Summaries of changed methods [31] 6) Summary of changed classes. Since the latter three components (4-6) of the commit context are generated by an LLM, we refer to them as the *LLM-derived commit context*. We used this context as the basis for our study. However, we made specific refinements when addressing our RQs, which we detail in Section III.F.

One of the contextual refinements we propose in this work is altering how method summaries were generated for the affected methods. Therefore, it is important to discuss the original approach adopted by OMG. OMG employs the Multi-Intent Method Summarization (**MMS**) technique proposed by Geng et al. [31] to summarize the methods affected by a commit,

MMS is an LLM-based method comment generation approach that uses few-shot prompting to generate the summaries for a method from five different aspects (Developer's intents) as follows: 1) **What** describes the functionality of a method 2) **Why** Explains the reason why a method is provided or the design rationale of the method 3) **How-to-use** Describes the usage or the expected set-up of using a method 4) **How-it-is-done** Describes the implementation details of a method 5) **Property** Asserts properties of a method including pre-conditions or post-conditions of a method.

Later in this section (See Section III.F), we discuss how MMS is employed by OMG, why it is not fit for the CMG task, and how we can overcome its limitations.

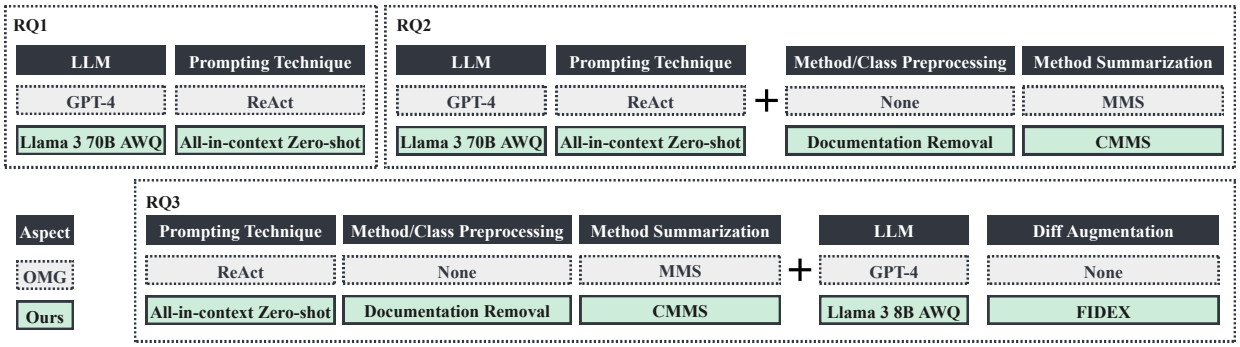

Fig. 1. Overall Methodology. Modified aspects of OMG are represented using Aspect. For each changed aspect, OMG shows the value for OMG, while Ours represents ours.

B. Datasets

Since our work builds on OMG, our dataset should include practitioner-evaluated CMs generated by OMG. This ensures our ground truth CMs have high human evaluation scores [3]. The dataset includes 381 commits from 32 Apache projects in Java. We used this dataset to compare our CMs generated using our approach with those generated by OMG using automated evaluation metrics and practitioner surveys.

Additionally, we used the same dataset to evaluate the performance of our candidate models in producing *LLM-derived commit context* (as discussed in Section III.A). Specifically, for the software maintenance activity type classifier, we used the same dataset of 1,151 commits in Java manually labeled with three maintenance activities (Corrective, Perfective, and Adaptive). To evaluate the performance of candidate OLLMs/SLMs in class summarization, similar to the approach taken by Li et al. [3], due to budget constraints and the high cost of using GPT-4, we sampled 384 class-summary pairs (confidence level 95%, margin of error 5%) from the class summary dataset used by OMG. Lastly, to examine our candidate OLLMs/SLMs in generating method summaries, we used the test set utilized by OMG. All these datasets are provided in the supplementary [32].

C. Evaluation Metrics

1) Automated Metrics: OMG-generated CMs have been evaluated by practitioners and achieved high human evaluation scores [3]. Therefore, we postulated that if we make our CMs similar to those generated by OMG, they would achieve acceptable results when evaluated by practitioners. This approach allowed us to use the similarity to OMG-generated CMs as an initial quality assurance measure before conducting human evaluations. Accordingly, we used standard evaluation metrics that are used to compare CMs with state-of-the-art machine-generated CMs [3], [24], [27]. Specifically, we used BLEU, METEOR, and ROUGE-L to measure the similarity between the CMs generated by an OLLM and SLM with those generated by OMG. Additionally, following the common practices in CMG research [3], we reported automated metrics by comparing our CMs with human-written CMs.

2) Human Metrics: In order to evaluate our CMs by practitioners, we opted for using the four human evaluation metrics proposed by Li et al. in our surveys. These metrics were developed through careful study of CMG literature and discussions among researchers [3]. The metrics are: 1) **Rationality**, which assesses whether a CM provides a logical explanation for the code change and identifies the software maintenance activity type. 2) **Comprehensiveness**, which evaluates whether the message summarizes what has been changed and includes relevant important details. 3) **Conciseness**, which measures the brevity of a CM. 4) **Expressiveness**, which examines the grammatical correctness and fluency of the CM.

D. Experimental Setup

We utilized a Linux server equipped with an NVIDIA A6000 GPU with 48GB of VRAM to run the OLLM inference engine. The 48GB of GPU VRAM limited our experiments to OLLMs with up to 20 billion trained parameters in full precision or up to a 4-bit quantized 70 billion parameter OLLM [33]. Therefore, we had to select a quantization method to quantize OLLMs with more than 20 billion trained parameters. Among the state-of-the-art quantization methods, we chose Activation-aware Weight Quantization (**AWQ**) due to its minimal impact on model's perplexity and the inference speedup it provides for the quantized LLM [34]. For the remainder of this paper, a quantized model refers to an OLLM that has been quantized to 4-bit using the AWQ technique.

To efficiently manage and execute an OLLM, we chose VLLM [35] due to its efficient memory management, compatibility with our server, and ease of deploying a wide range of OLLMs. VLLM provides an OpenAI-compatible API, facilitating its use with LLM app developments such as Langchain [36], which we used to develop the LLM agents utilized in this study. However, at the time of our experiments, VLLM lacked the embedding inference capability. Consequently, we could not use the original class summarization approach employed by Li et al., which relied on embedding the changed classes and using retrieval-based question answering to summarize them. Instead, we utilized zero-shot prompting to summarize the affected classes. Lastly, to ensure consistent output, we

set the temperature to 0 when using the deployed OLLM for inference, similar to previous work [3].

E. Survey

Overall, we conducted three practitioner surveys to answer our research questions.

Commits Sampling: We adopted a similar approach as Li et al. while conducting our surveys to keep the workload manageable for participants. Specifically, evaluating CMs for the entire commit dataset, which involves assessing 762 candidate CMs for 381 commits from four perspectives, would be impractical for participants. Therefore, we randomly sampled 15 commits for each survey, resulting in 30 commit messages (15 CMs generated using our approach and 15 CMs generated using the approach we are comparing with) for participants to compare and evaluate. This is the same total number of CMs that were evaluated by the survey participants in the evaluation of OMG, ensuring a fair workload to achieve a high completion rate [3].

Participant Recruitment: We adopted the snowball sampling approach to recruit the participants for our surveys [37]. Specifically, we began by distributing the survey to our industry contacts with at least two years experience in Java development and sent periodic reminders to encourage participation. Additionally, we asked them to share the survey with other developers who have relevant programming backgrounds. This approach ensured that participants possessed the necessary knowledge to accurately assess the CMs generated for Java projects. The majority of survey participants were male, with 20% identifying as female. Participants had Java programming experience ranging from 2 to 20 years, and all were based in North America.

Survey Design: All three surveys were designed to comparatively evaluate CMs written for a sample of 15 commits. For the first and last surveys, we presented two candidate CMs for each commit (CM #1 and CM #2). We randomly shuffled the questions to eliminate any bias towards an option due to an observable pattern in the candidate CMs. The survey was hosted on QuestionPro [38], and participants were provided with definitions of the human evaluation metrics. Following the definitions, we presented the commits with all the commit context along with the two candidate CMs. For each human evaluation metric, we asked the participants to select their preferred CM or choose "Identical" if there was no clear preference. The Identical option was provided to see if the CMs generated by the OLLM/SLM are "comparable" to those produced by OMG. Additionally, for each commit, we asked the participants to select their overall preferred CM.

The second survey was designed differently, as its purpose was to validate the comprehensiveness of the CMs after making LLM-derived commit refinements. For each commit, we presented the CMs before and after the refinements. Similar to the other surveys, we randomly shuffled the questions. For each commit, we asked the participants to choose a candidate CM that is more comprehensive. We did not provide an "Identical" option for this survey, as our goal was to assess whether the enhanced context produced more comprehensive CMs, not just comparable ones.

F. Answering RQs

In the following subsections, we detail the steps taken to address each research question.

RQ1. *Can an OLLM generate CMs comparable to a state-of-the-art LLM (GPT-4)?*

OLLM Selection: To answer RQ1, we needed to compile a list of candidate models. Given the reliance of OMG on code summarization and the necessity of code understanding to generate high-quality CMs, candidate models had to demonstrate strong performance on code-related tasks by scoring high on relevant benchmarks. We selected four OLLMs as candidates to answer RQ1. All selected models held leading rankings in the EvalPlus leaderboard [39] and the Big Code Models leaderboard [40] at the time of compiling the candidate list. EvalPlus is a code synthesis evaluation framework designed to benchmark the functional correctness of LLM-synthesized code. The Big Code Models leaderboard evaluates the performance of base multilingual code generation models on the HumanEval benchmark and MultiPL-E. These benchmarks are commonly referenced by researchers when selecting LLMs [41]. We chose a quantized instruction-tuned Llama3 70B [42] and a quantized instruction-tuned DeepSeek-Coder 33B [43]. We selected instruction-tuned Llama3 70B since its score on the HumanEval benchmark was comparable to the top-3 models on the Big Code Models leaderboard [42]. We also selected the instruction-tuned DeepSeek-Coder 33B because it was ranked as one of the top models in the EvalPlus benchmark.

In addition, we used the AutoAWQ library to quantize CodeFuse-DeepSeek-33B and OpenCodeInterpreter-DS-33B [44], as AWQ quantized versions of these models were not available on Huggingface. When sorting the models on the Big Code Models leaderboard by their performance in generating correct Java code, CodeFuse-DeepSeek-33B and OpenCodeInterpreter-DS-33B were ranked among the top models.

In order to ensure the quality of the *LLM-derived commit context*, similar to Li et al. [3], we evaluated the candidate OLLMs performance on class summary generation, MMS, and classifying software maintenance activity types. Based on the evaluation results, we selected the top performing model, quantized instruction-tuned Llama3 70B, for our experiments to answer RQ1 and RQ2.

Prompting Method: To address RQ1, we aimed to assess the feasibility of replacing GPT-4 with an OLLM in the original implementation of OMG. While the prompting method adopted by OMG, ReAct [8], does not align with one of our study's primary objectives, ensuring the sustainability of the CMG approach, we opted to begin our experiments with this method. This choice allowed us to isolate the impact of replacing the LLM while keeping other aspects of OMG unchanged. However, we observed poor automated scores

when comparing the CMs generated using ReAct by our selected OLLM to those generated by OMG. This led us to question the capability of the OLLM in generating useful thoughts to reason about each contextual piece of information about a commit, as noted by other researchers [45].

To address the poor performance, we avoided using more advanced prompting techniques to ensure the results did not stem from improvements in the adopted prompting approach. Hence, we utilized the simplest prompting method, zero-shot prompting [46]. Instead of expecting the OLLM to request each piece of commit context as in ReAct, we provided all the available context about a commit along with CMG instructions in one prompt. This change in prompting technique increased the BLEU score achieved by the OLLM by 118%. The high automated scores (METEOR and ROUGE-L above 30) from zero-shot prompting suggested that the generated CMs resembled those produced by OMG. To verify their quality, we conducted a survey with practitioners to compare the CMs from the selected OLLM with those from OMG using human metrics. The survey results are discussed in Section IV.

The survey verified our assumption. However, practitioners preferred OMG-generated CMs for their comprehensiveness, which led to our second research question.

RQ2. *How can we bridge the comprehensiveness gap between the CMs produced by an OLLM and the CMs generated by a state-of-the-art LLM (GPT-4)?*

To answer RQ2, we initially expanded our zero-shot prompt to ask the OLLM to comprehensively detail all the changes that occurred in the commit without missing anything. However, this not only failed to increase the automated metrics but also moderately decreased them.

Therefore, since a change in the prompt did not address RQ2, we shifted our focus to the provided context. Specifically, we investigated the effectiveness of the *LLM-derived commit context*, namely, the summaries of changed classes and methods, and the software maintenance activity type. Our CMs were structured similarly to OMG; they included a header with the software maintenance activity type and a brief subject, followed by the body. Consequently, the software maintenance activity type contributed only one word to the CM, i.e., one of refactor, fix, style, or feat [3]. Therefore, we concentrated on the class and method summaries, positing that improvements in these should help the OLLM write more comprehensive CMs.

According to the literature, mismatched code comments cause the models to produce summaries that reflect the comments rather than the actual code functionality [47]. Hence, we added a preprocessing step to the class and method bodies passed to the OLLM for summarization by removing all the documentation (comments and Javadocs).

Furthermore, we changed the way the MMS was done for modified methods (methods that exist in both pre- and post-commit states). Originally, OMG generated these summaries separately for the pre-commit and post-commit bodies of affected methods and provided these as the summaries of the affected methods. However, we hypothesize this approach is not suitable for a code change-based task like CMG, as it does not ask the OLLM to generate summaries based on how the changes in the diff impact each of the five different aspects (as discussed in Section III.A) of affected methods. Instead, this approach assumes LLM's ability in generating different summaries with slight changes to the method bodies.

In order to validate our assumption, we randomly sampled 55 commits from the 280 commits in which the changes affected a method (confidence level 90%, margin of error 10%). Two authors independently compared the generated summaries for the pre- and post-commit bodies of the affected methods to see if the changes are correctly captured (yes/no) in the summaries for any of the five aspects. We observed that in 96% of the commits (inter-rater agreement of 88%), there was no conceptual difference in the generated method aspects by the OLLM for the method bodies before and after the commit. This confirmed our hypothesis.

CMMS: To address the identified shortcoming of MMS, generating semantically identical summaries for affected methods before and after the commit, we introduce **C**hange-based **M**ulti-Intent **M**ethod **S**ummarization (**CMMS**). For each modified method, we first applied the approach proposed by Geng et al. [31] to generate multi-intent summaries for the original method body. We then utilized a Python script to list the upcoming changes by comparing the method before and after the commit. These changes could include Additions (e.g., "*int a = 3*" will be added after line 12), Removals (e.g., line 12 will be removed), or Replacements (e.g., "*farthest.addNormalized(dn.getRdn(ii));*" will replace "*farthest.add(dn.getRdn(ii));*" in line 12). Finally, we provided the following inputs to the OLLM: 1) the original method with line numbers, 2) the list of upcoming changes, and 3) the multi-intent summaries for the original method. We then asked the model to explain how each of the five aspects for the original method body would be affected by these upcoming changes to the original method. For instance, changes to a method's input arguments affect the *How-to-use* aspect of it.

We conducted an ablation study to examine the effectiveness of each refinement (documentation removal and CMMS) separately. Both refinements improved all automated metrics, using OMG as the reference. We report the automated evaluation results in Section IV. Since the OMG-generated CMs were perceived as more comprehensive by the participants in our first survey, this increased similarity to OMG-generated CMs led us to infer that the produced CMs had become more comprehensive. To validate this assumption, we conducted a second survey and asked participants to compare the CMs produced with the old commit context to those resulting from the enhanced commit context. The survey results verified our hypothesis. The survey results are reported in Section IV.

RQ3. *Can a smaller OLLM (SLM) produce CMs comparable to a state-of-the-art LLM (GPT-4)?*

SLM Selection: To maintain a fair comparison between the performance of a selected SLM and the quantized Llama3

70B, we only considered the quantized versions of any SLM. Given the acceptable performance of the quantized Llama3 70B, we included the lighter version of the Llama3 family, the quantized instruction-tuned Llama3 8B, as one of the candidate SLMs. Additionally, we considered the quantized versions of conversation-tuned CodeQwen1.5 7B [48] and instruction-tuned Mistral v0.3 7B [49]. At the time of gathering the candidate SLMs, these three models had the highest rankings among OLLMs with under 10 billion trained parameters in the RepoQA benchmark, which evaluates LLMs' capability in long-context code understanding tasks [50].

Similar to our approach for OLLM selection in RQ1, we evaluated the candidate SLMs on class summarization, MMS, and software maintenance activity classification tasks. Initially, we selected the conversation-tuned CodeQwen1.5 7B due to its superior performance in generating the *LLM-derived commit context*. However, after observing its output in an initial CMG experiment, we noticed that the model often repeated the same sentences to fill the allowed maximum tokens. While we could have used a repetition or frequency penalty to mitigate this issue, best practice from the relevant research dictates that the model should not get penalized for reusing tokens [51]. Hence, we used the second-best model, quantized instruction-tuned Llama3 8B, for the remaining experiments without any penalties.

CMG Experiments: Given the improved comprehensiveness achieved in RQ2 and the overall good performance of our zero-shot prompting in RQ1, we experimented with the selected SLM under zero-shot prompting using the enhanced *LLM-derived commit context*. However, the automated scores were considerably lower than those achieved by the selected OLLM. Given the identical commit context, we posit that the lower automated metrics are due to the model's inability to correctly and comprehensively understand the changes introduced by the commit. Dong et al. found a similar issue in learning-based CMG approaches by observing that their poor quality lies in strong attention weights for marks (+/-/white spaces line prefixes) in a diff [52]. Nonetheless, they did not evaluate LLMs' understanding of a git diff and its impact on the generated CM. Accordingly, to validate our hypothesis, we randomly sampled 38 commits and asked the selected SLM to explain the changes in the diff of those commits. Next, two authors independently evaluated the SLM's answers and marked each answer as correct if all the changes in the diff were correctly covered, otherwise incorrect. We found out that in 82% of the cases, the LLM's response was not correct. The inter-rater agreement for this analysis study was 100%.

FIDEX: In light of these findings, we realized the necessity of a diff augmentation approach that helps the SLM correctly understand the changes in a diff. This augmentation approach should address specific considerations. Firstly, it should be able to accurately comprehend all the changes that occurred in the diff without any errors (*C1*). Additionally, research suggests that explicit prompting boost the performance of LLMs in inferential reasoning tasks [53], such as ours, which is inferring the changes in a diff during CMG. Hence, the diff augmentation approach should be able to remove this inference step for the SLM in the CMG task by explicitly stating the differences between the old and new versions (*C2*). Furthermore, relevant studies in CMG suggest that providing fine-grained details of the changes in a diff can boost the performance [54]. Therefore, the diff augmentation approach should provide details about the differences (*C3*).

We did not find a similar approach that addresses these considerations.Therefore, we propose **F**ine-grained **I**nteractive **D**iff **EX**plainer **(FIDEX)**, a hybrid LLM-based approach to produce a detailed explanation of a diff, highlighting all differences between the pre- and post-commit versions of affected Java files. FIDEX is a hybrid approach since it leverages a deterministic solution to understand the changes in a diff (C1) and uses an LLM based solution while explaining the differences between the old and new versions (C2 & C3).

Specifically, since we observed the SLM's inaccuracy in understanding the changes in a diff, we devised a deterministic solution to minimize the hallucinations by FIDEX in understanding the changes (C1). Particularly, we developed a Python script named *Diff Narrator*. Given a commit diff, Diff Narrator outputs a *Diff Narrative*, which is a numbered list of *Change Items*. Change Items are basic units of changes in a diff and can be either of the following cases: 1) *Addition Chunk* Consecutive lines with '+' mark 2) *Removal Chunk* Consecutive lines with '-' mark 3) *Replacement Chunk* A Removal chunk immediately followed by an Addition chunk.

To explain the differences between the old and the new version of affected Java files (C2 & C3), FIDEX prompts an LLM. We adopted the role-playing prompting technique in FIDEX, which has been shown to outperform zero-shot prompting in several reasoning benchmarks [55]. Specifically, FIDEX consists of two phases: Prompt Construction and Output Construction. Figure 2 illustrates the FIDEX approach as a conversation between the user and an LLM.

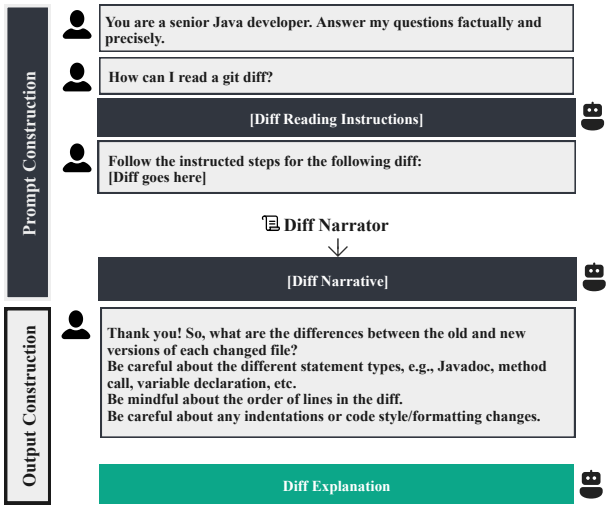

Fig. 2. Fine-grained Interactive Diff Explainer (FIDEX). Values wrapped in square brackets are hard-coded. Except the last message, other messages are fed into the LLM's memory.

```
diff --git a/maven-mboot2/src/main/java/jar/JarMojo.java b/maven-
mboot2/src/main/java/jar/JarMojo.java
index 0ecfceb9a..c1cf57026 100644
--- a/maven-mboot2/src/main/java/jar/JarMojo.java
+++ b/maven-mboot2/src/main/java/jar/JarMojo.java
@@ -26,7 +26,7 @@ public class JarMojo

        Map includes = new LinkedHashMap();

-       addDirectory( includes, "**/**", "**/package.html", "", basedir );
+       addDirectory( includes, "**/**", "**/package.html,**/.svn/**", "", basedir );

        createJar( jarFile, includes );
    }
```

File maven-mboot2/src/main/java/jar/JarMojo.java has been modified in this commit.
Following changes have been made to the file:

1.
Following line(s)

```
        addDirectory( includes, "**/**", "**/package.html", "", basedir );
```

is replaced with with following line(s)

```
        addDirectory( includes, "**/**", "**/package.html,**/.svn/**", "", basedir );
```

Based on the provided diff, the differences between the old and new versions of the JarMojo.java file are:

1. The line `addDirectory( includes, "**/**", "**/package.html", "", basedir );` has been replaced with `addDirectory( includes, "**/**", "**/package.html,**/.svn/**", "", basedir );`.

The change is in the third argument of the `addDirectory` method. The old version includes only `**/package.html`, while the new version includes `**/package.html` and `**/.svn/**`. This suggests that the new version of the code is including the `.svn` directory in the package.

Fig. 3. Example of a diff, its Diff Narrative, and its FIDEX-generated diff explanation using quantized Llama3 8B (From top to bottom). The highlighted parts show how diff explanation outlines the difference between the old and new version in fine detail.

In the Prompt Construction phase, we prepare the role prompts to be passed to the LLM. The prompts start with a Role-Setting prompt [55], where we define the LLM as a senior Java developer and instruct it to answer questions factually and precisely. Following this, we include two rounds of User-LLM interactions. In the first round, the user asks the LLM to provide instructions on how to read a diff. The LLM's response includes predefined instructions for reading a diff. In the second round, the user asks the LLM to follow these instructions and describe all the changes in the input diff. To generate the LLM's response for this prompt, we use the Diff Narrative produced by Diff Narrator.

Lastly, the Output Construction stage is where the final interaction between the user and LLM takes place. The user asks the LLM to describe the differences between the old and new versions of each affected file (C2), while considering specific cautions to ensure accuracy and comprehensiveness. The cautions are designed to warn the LLM about different fine-grained statement types (C3), such as Javadocs, method declarations, etc., to avoid confusion between documentation and code changes [56]. Additionally, the LLM is instructed to be mindful of the order of changes to highlight reordering changes and to pay attention to the sequence of lines in the diff. The final consideration is asking the LLM to differentiate between code style or formatting changes, which helps identify stylistic changes and correctly differentiate between different software maintenance activity types in the commit [3]. Figure 3 provides an example of a raw diff, the *Diff Narrative* produced by the *Diff Narrator* (in grey), and a diff explanation that is generated by the selected SLM through using the FIDEX.

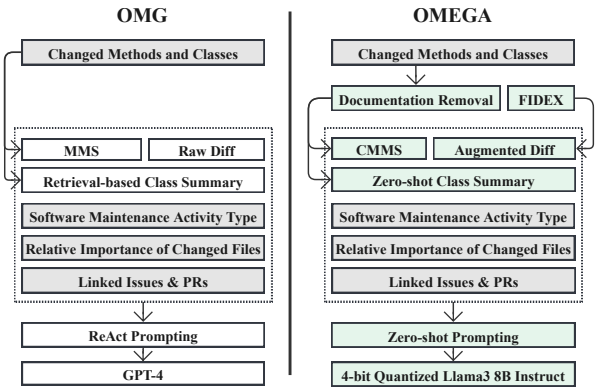

Fig. 4. Differences between OMG and OMEGA

We augmented the raw diff with the FIDEX-generated diff explanation to observe its impact on the similarity of the generated CMs to those generated by OMG. In addition, we used the *Diff Narrative* as a diff augmentation to determine if the additional details provided by FIDEX help the SLM write CMs more similar to those produced by OMG. Lastly, we conducted a final practitioner survey to measure the effectiveness of CMG using the selected SLM. Figure 4 highlights the differences between OMG and our final CMG approach, OMEGA.

IV. Results

In this section, we address our research questions by presenting the results of both automated and human evaluations.

RQ1. *Can an OLLM generate CMs comparable to a state-of-the-art LLM (GPT-4)?*

Automated Evaluation*:* To ensure the quality of the LLM-derived commit context, similar to Li et al. [3], we evaluated the candidate OLLMs performance on class summarization, MMS, and classifying software maintenance activity types. Table I presents the results of automated evaluation of candidate OLLMs in generating LLM-derived commit context. Among the candidate OLLMs, the quantized instruction-tuned Llama3 70B scored highest in classifying software maintenance activity type and class summarization, leading to its selection for experiments in RQ1 and RQ2.

As discussed in the Section III, we experimented with ReAct and zero-shot prompting techniques to answer RQ1. Table II presents the automated evaluation results for these two prompting approaches. Based on the automated scores, using zero-shot prompting resulted in a BLEU score that was 2.18 times higher than that achieved by adopting ReAct. Therefore, we used the CMs generated through zero-shot prompting technique in our first survey to assess their quality by practitioners.

TABLE I
Automated Evaluation Results for Generating *LLM-derived commit context*. SMA stands for Software Maintenance Activity*. We used the value reported by Li et al. to avoid incurring additional costs△. Bold models are the selected candidates.

Model	Candidate For	Class Summarization			Method Summarization			SMA* Classification Accuracy
		BLEU	METEOR	ROUGE-L	BLEU	METEOR	ROUGE-L	
gpt-4-turbo-2024-04-09		1.74	18.85	15.58	3.87	30.11	22.21	51%△
CodeFuse-DeepSeek-33B	RQ1-2	2.09	17.07	16.81	15.50	**38.11**	**37.36**	35%
Deepseek-Coder 33B Instruct	RQ1-2	2.01	18.05	18.26	8.42	36.04	28.90	36%
OpenCodeInterpreter-DS-33B	RQ1-2	1.06	20.21	16.47	**16.11**	37.42	37.01	36%
Llama3 70B Instruct	RQ1-2	**2.51**	**20.58**	**18.26**	9.64	35.75	30.66	**50%**
CodeQwen 1.5 7B Chat	RQ3	2.16	**20.05**	**18.57**	**19.46**	**38.91**	**39.58**	37%
Mistral 7B Instruct v0.3	RQ3	1.58	17.59	15.17	9.90	34.70	31.59	34%
Llama3 8B Instruct	RQ3	**2.18**	16.90	16.76	13.41	35.66	33.37	**46%**

TABLE II
Automated Evaluation Results for RQ1 and RQ2 using instruction-tuned AWQ-quantized Llama3 70B. Before and after values for commit enhancements are separated by slash. Rows with bold context were used in the survey conducted for the RQ.

RQ	Prompt	Commit Context	Reference OMG			Reference Human		
			BLEU	METEOR	ROUGE-L	BLEU	METEOR	ROUGE-L
1	ReAct	Same as OMG	5.2	20.33	27.22	2.29	17.47	11.92
	Zero-shot	**Same as OMG**	11.36	30.37	31.46	1.26	17.39	10.03
2	Zero-shot	OMG - Documentation	10.54 / 14.12	29.84 / 35.78	27.64 / 32.76	1.05 / 1.09	14.98 / 17.04	7.59 / 8.76
		MMS replaced by CMMS	10.32 / 14.08	29.95 / 36.44	27.54 / 31.83	1.06 / 0.89	15.57 / 17.05	7.76 / 8.41
		Refined	10.54 / 14.19	29.84 / 36.44	27.64 / 32.06	1.05 / 0.95	14.98 / 16.38	7.59 / 8.16

Human Evaluation: In our first survey, 10 practitioners participated. The CMs were randomly sampled using all-in-context zero-shot prompting with the original OMG commit context. Figure 5 illustrates the results of this survey. OLLM-generated CMs were preferred in 39% of responses, while OMG-generated CMs were selected as the overall preferred CM in 29% of responses. In 52% of the responses, the CM generated by the selected OLLM was perceived as more concise. Overall, there was no aspect in which OMG was selected by the majority of responses over the selected OLLM (quantized instruction-tuned Llama3 70B). However, OLLM-generated CMs were selected in fewer responses in terms of comprehensiveness (OLLM:25% of responses and OMG:33%). This observation led to the formulation of RQ2, which we addressed through LLM-derived commit context enhancements.

> An OLLM can produce CMs that are comparable overall to those generated by a state-of-the-art LLM except in terms of comprehensiveness.

RQ2. *How can we bridge the comprehensiveness gap between the CMs produced by an OLLM and the CMs generated by a state-of-the-art LLM (GPT-4)?*

Automated Evaluation: Table II presents the automated evaluation results for our ablation study (See Section III), highlighting the impact of each contextual refinement in bridging the identified comprehensiveness gap from our first survey. Removing documentation from affected method and class bodies significantly improved all automated scores, increasing the BLEU score of the CMs from 10.54 to 14.12, a 34% improvement. Replacing MMS with CMMS without documentation removal also led to similar improvement, although the BLEU score when comparing CMs to human-written ones degraded by 16%. Given the effectiveness of each standalone enhancement, we combined them, resulting in a BLEU score

TABLE III
Automated Evaluation Results for RQ3 using instruction-tuned AWQ-quantized Llama3 8B. The last row presents the automated scores of OMEGA, which was used for survey 3.

Prompt	Commit Context	Diff Augmentation Technique	Reference OMG			Reference Human		
			BLEU	METEOR	ROUGE-L	BLEU	METEOR	ROUGE-L
Zero-shot	Refined	None	10.78	31.18	28.16	0.86	14.72	7.38
		Diff Narrator	12.19	32.26	29.81	0.89	15.09	7.99
		FIDEX (Used in OMEGA)	**13.01**	**33.13**	**30.85**	**1.22**	**16.08**	**8.26**

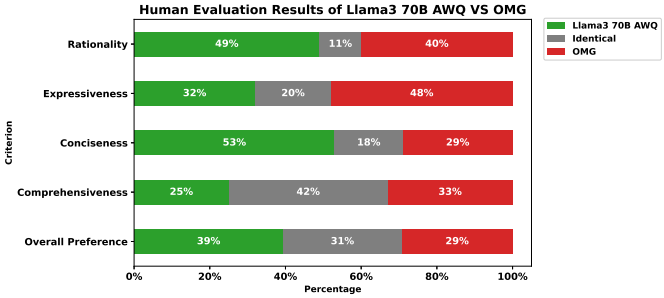

Fig. 5. Human Evaluation Results of Llama3 70B AWQ VS OMG

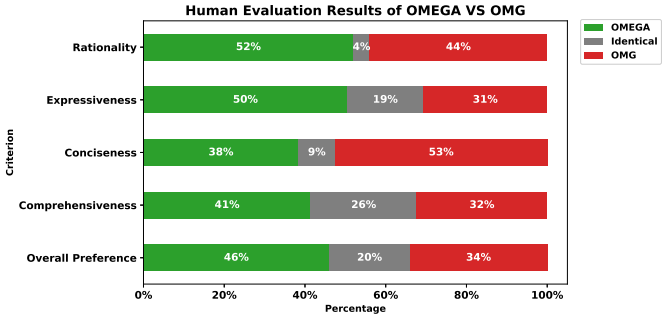

Fig. 6. Human Evaluation Results of OMEGA VS OMG

of 14.19 when compared to OMG, a 35% higher score than the score achieved by CMs used for our first practitioner survey.

Human Evaluation: Our second survey was designed to assess the effectiveness of our strategy to bridge the identified comprehensiveness gap in the first survey. Hence, 10 participants compared 15 randomly sampled CMs that were generated before the LLM-derived context enhancements with those generated by the enhanced context. Based on the survey results, 71% of responses found the CM generated through the enhanced context more comprehensive.

> Replacing MMS with CMMS, along with a documentation removal step before summarizing affected methods and classes, mitigates the comprehensiveness gap.

RQ3. *Can a smaller OLLM (SLM) produce CMs comparable to a state-of-the-art LLM (GPT-4)?*

Automated Evaluation: Similar to RQ1, we evaluated the candidate SLMs performance on class summarization, MMS, and classifying software maintenance activity types. Table I presents the automated scores achieved by each candidate SLM in generating LLM-derived commit context. Although the quantized conversation-tuned CodeQwen 7B was the best performing model based on the automated scores, we chose the second best SLM, the quantized instruction-tuned Llama3 8B, after observing the problem of repeated sentences by the quantized conversation-tuned CodeQwen 7B in CMG (See Section III for details).

We present the automated scores for our experiments in RQ3 in Table III. As shown in the table, each augmentation to the raw diff consistently improves all automated scores. Two key observations can be made from this table. Firstly, the FIDEX-generated diff explanation enhances the BLEU score when compared to human-written CMs by 42%. Secondly, the difference between various diff augmentation approaches compared to the raw diff is more significant than the differences among these methods themselves. We posit that this is because providing the Diff Narrative gives sufficient context for simple diffs, and the CMs for those diffs do not significantly change with the FIDEX approach. Nevertheless, appending the diff with our FIDEX-generated diff explanation improves the BLEU score by 21%, which is only 8% lower than the score achieved by the selected OLLM without diff augmentation (RQ2).

Human Evaluation: Our last survey aimed to evaluate the performance of OMEGA by assessing the impact of augmenting commit diff with FIDEX-generated explanations compared to OMG. A total of 22 practitioners participated in this survey. The results are presented in Figure 6. We conducted a Chi-Square test [57] on the survey results for each criterion. The average p-value of 0.0069 indicates a rejection of the null hypothesis, suggesting significant differences across the criteria. According to the findings, the selected SLM outperforms GPT-4 in generating CMs that meet practitioners' expectations, except in conciseness. Figure 7 illustrates an example of a CM evaluated in the survey. The evaluation results for this CM reveal a trade-off between comprehensiveness and conciseness, as perceived by the participants. Comprehensiveness and conciseness are often viewed as opposing qualities [3]. While we did not explicitly ask developers about their preferences, a Spearman's correlation analysis [58] of the survey results indicates a strong positive correlation between comprehensiveness and preferred messages (Coefficient = 0.84, p-value = 0.00). However, explicit developer feedback is required to validate this observation.

Overall, SLM-generated CMs were preferred in 46% of responses, while OMG-generated CMs were preferred in 34% of responses. Notably, the preference for SLM-generated CMs in all metrics, except conciseness, was higher than that for the quantized instruction-tuned Llama3 (OLLM). This highlights the effectiveness of the LLM-derived commit context enhancements (RQ2) and the FIDEX-generated diff summaries (RQ3) in enabling a quantized OLLM to produce superior CMs compared to GPT-4.

> Augmenting the commit diff with an explanation generated by FIDEX enables an SLM with just 0.005% of the trained parameters to outperform GPT-4 in CMG.

V. Threats to Validity

We adopted measures to mitigate potential threats to the validity of our work, which we outline in this section.

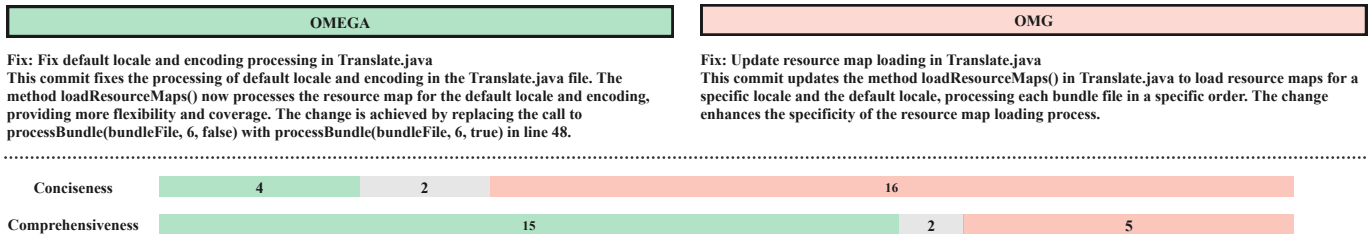

Fig. 7. An Example of the trade-off between perceived comprehensiveness and conciseness of commit messages generated by OMEGA against OMG

Construct Validity Relying on automated machine translation metrics for CM evaluation has been shown not to align with human preferences[3]. To minimize our reliance on these metrics, we used them solely as a preliminary step to ensure the generated CMs' similarity to OMG-written ones. The quality of the CMs were ultimately assessed by practitioners using state-of-the-art human evaluation metrics [3]. There is a possibility that survey participants may have misunderstood the human evaluation criteria when assessing the candidate CMs. To mitigate this risk, we provided definitions for each human evaluation criterion at the beginning of the survey. Lastly, our choice of quantization method may have hampered the selected model's performance. However, among the existing quantization methods at the time of conducting our study, our selected approach was the one with the minimal impact on the model's performance [34].

Internal Validity To ensure that our results solely stem from the contextual refinements we made and are not influenced by the selection of LLM inference parameters, we set the temperature to 0 and avoided any repetition or frequency penalties. This approach makes the OLLMs' responses reproducible and deterministic.

External Validity Similar to OMG, our study is limited to Apache projects written in the Java programming language. However, given the multilingual programming datasets used to train our selected OLLMs, we posit that our results are generalizable to other programming languages as well. Additionally, the choice of OLLM/SLMs may affect the generalizability of our findings. However, to mitigate this threat, we tested our approach with multiple candidate OLLM/SLMs before selecting the best performing one.

VI. Conclusion

This study investigated the feasibility of generating state-of-the-art CMs that meet practitioners' expectations using OLLMs. Our results show that an OLLM can produce CMs comparable to those generated by proprietary models, though with poorer comprehensiveness performance. By refining the LLM-derived commit context, we bridged this performance gap. Additionally, our findings demonstrate that even a smaller LLM (SLM) can perform as well as, if not better, compared to a large proprietary LLM or large OLLM.

Our findings encourage researchers to explore ways to carefully curate task-specific contextual information for LLMs to achieve better results rather than relying solely on larger proprietary LLMs with poorly devised contexts. Our approach to generating high-quality CMs using an SLM offers significant advantages for the industry. The lower hardware requirements of SLMs make them adaptable for companies and practitioners with limited computational resources and facilitate the avoidance of sharing sensitive information with external providers. In the future, we plan to develop an Integrated Development Environment (IDE) extension for OMEGA and conduct a usability study to evaluate its effectiveness. Additionally, we aim to explore previously unexamined contextual information, such as changes that incorporate context from other files and modules. We provide the source code and datasets that were used in our experiments in the supplementary [32].

References

[1] S. Rebai, M. Kessentini, V. Alizadeh, O. B. Sghaier, and R. Kazman, "Recommending refactorings via commit message analysis," *Information and Software Technology*, vol. 126, p. 106 332, 2020.

[2] Y. Tian, Y. Zhang, K.-J. Stol, L. Jiang, and H. Liu, "What makes a good commit message?" In *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 2389–2401.

[3] J. Li, D. Faragó, C. Petrov, and I. Ahmed, "Only diff is not enough: Generating commit messages leveraging reasoning and action of large language model," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, 2024.

[4] D. M. Ziegler, N. Stiennon, J. Wu, *et al.*, *Fine-tuning language models from human preferences*, 2020. arXiv: 1909.08593.

[5] OpenAI, J. Achiam, S. Adler, *et al.*, "GPT-4 Technical Report," Tech. Rep., Mar. 2023.

[6] A. Eliseeva, Y. Sokolov, E. Bogomolov, Y. Golubev, D. Dig, and T. Bryksin, "From commit message generation to history-aware commit message completion," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2023, pp. 723–735.

[7] *GPT-4 - Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/GPT-4.

[8] S. Yao, J. Zhao, D. Yu, *et al.*, "ReAct: Synergizing Reasoning and Acting in Language Models," Oct. 2022.

[9] *Samsung Bans ChatGPT Among Employees After Sensitive Code Leak*. [Online]. Available: https://tinyurl.com/samsung-leaks.

[10] *Amazon's Q has 'severe hallucinations' and leaks confidential data in public preview, employees warn*. [Online]. Available: https://www.platformer.news/amazons-q-has-severe-hallucinations/.

[11] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, "A survey on large language model (LLM) security and privacy: The Good, The Bad, and The Ugly," *High-Confidence Computing*, vol. 4, no. 2, p. 100 211, 2024.

[12] OpenAI, *Chatgpt*, Accessed: 2024-07-28, 2024. [Online]. Available: https://www.openai.com/research/chatgpt.

[13] A. A. Chien, L. Lin, H. Nguyen, V. Rao, T. Sharma, and R. Wijayawardana, "Reducing the Carbon Impact of Generative AI Inference (today and in 2035)," in *Proceedings of the 2nd Workshop on Sustainable Computer Systems*, ser. HotCarbon '23, New York, NY, USA: Association for Computing Machinery, 2023.

[14] M. Ferreira, D. Gonçalves, M. Bigonha, and K. Ferreira, "Characterizing commits in open-source software," in *Proceedings of the XXI Brazilian Symposium on Software Quality*, 2023.

[15] Y. Zhang, Z. Qiu, K.-J. Stol, *et al.*, "Automatic commit message generation: A critical review and directions for future work," *IEEE Transactions on Software Engineering*, vol. 50, no. 4, pp. 816–835, 2024.

[16] A. Faiz, S. Kaneda, R. Wang, *et al.*, "LLMCarbon: Modeling the end-to-end Carbon Footprint of Large Language Models," Sep. 2023.

[17] Y. Liu, S. Tao, W. Meng, *et al.*, "Interpretable online log analysis using large language models with prompt strategies," in *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*, 2024, pp. 35–46.

[18] L. Zhong and Z. Wang, *Can chatgpt replace stackoverflow? a study on robustness and reliability of large language model code generation*, 2024. arXiv: 2308.10335.

[19] X. Yin, C. Ni, and S. Wang, *Multitask-based evaluation of open-source llm on software vulnerability*, 2024. arXiv: 2404.02056.

[20] R. Pan, A. R. Ibrahimzada, R. Krishna, *et al.*, "Lost in translation: A study of bugs introduced by large language models while translating code," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024.

[21] J. Liu, J. L. Tian, V. Daita, *et al.*, "RepoQA: Evaluating Long Context Code Understanding," Jun. 2024.

[22] *Llama 3 performance and cost benchmarks — Parseur®*. [Online]. Available: https://parseur.com/blog/blog-llama3-performance-cost.

[23] E. Shi, Y. Wang, W. Tao, *et al.*, *Race: Retrieval-augmented commit message generation*, 2022. arXiv: 2203.02700.

[24] J. Dong, Y. Lou, Q. Zhu, *et al.*, "Fira: Fine-grained graph-based code change representation for automated commit message generation," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 970–981.

[25] S. Liu, C. Gao, S. Chen, L. Y. Nie, and Y. Liu, "ATOM: Commit Message Generation Based on Abstract Syntax Tree and Hybrid Ranking," *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1800–1817, 2022.

[26] W. Tao, Y. Wang, E. Shi, *et al.*, "On the Evaluation of Commit Message Generation Models: An Experimental Study," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2021, pp. 126–136.

[27] Y. He, L. Wang, K. Wang, Y. Zhang, H. Zhang, and Z. Li, "COME: Commit Message Generation with Modification Embedding," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2023, 2023, pp. 792–803.

[28] W. Tao, Y. Zhou, Y. Wang, H. Zhang, H. Wang, and W. Zhang, "KADEL: Knowledge-Aware Denoising Learning for Commit Message Generation," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 5, Jun. 2024.

[29] X. Hou, Y. Zhao, Y. Liu, *et al.*, *Large language models for software engineering: A systematic literature review*, 2024. arXiv: 2308.10620.

[30] C. Munley, A. Jarmusch, and S. Chandrasekaran, *Llm4vv: Developing llm-driven testsuite for compiler validation*, 2024. arXiv: 2310.04963.

[31] M. Geng, S. Wang, D. Dong, *et al.*, "Large language models are few-shot summarizers: Multi-intent comment generation via in-context learning," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024.

[32] *OMEGA Repository*. [Online]. Available: https://github.com/aaron-imani/omega.

[33] J. Dai, *GPU-Benchmarks-on-LLM-Inference*. [Online]. Available: https://github.com/XiongjieDai/GPU-Benchmarks-on-LLM-Inference.

[34] J. Lin, J. Tang, H. Tang, *et al.*, "AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration," Jun. 2023.

[35] W. Kwon, Z. Li, S. Zhuang, *et al.*, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[36] *LangChain*. [Online]. Available: https://www.langchain.com/.

[37] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014.

[38] *QuestionPro*. [Online]. Available: https://www.questionpro.com.

[39] J. Liu, C. S. Xia, Y. Wang, and L. ZHANG, "Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 21 558–21 572.

[40] L. Ben Allal, N. Muennighoff, L. Kumar Umapathi, B. Lipkin, and L. von Werra, *A framework for the evaluation of code generation models*, https://github.com/bigcode-project/bigcode-evaluation-harness, 2022.

[41] X. Du, M. Liu, K. Wang, *et al.*, "Evaluating large language models in class-level code generation," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024.

[42] *Meta Llama 3*. [Online]. Available: https://llama.meta.com/llama3/.

[43] D. Guo, Q. Zhu, D. Yang, *et al.*, *Deepseek-coder: When the large language model meets programming – the rise of code intelligence*, 2024. arXiv: 2401.14196.

[44] T. Zheng, G. Zhang, T. Shen, *et al.*, *Opencodeinterpreter: Integrating code generation with execution and refinement*, 2024. arXiv: 2402.14658.

[45] M. Xing, R. Zhang, H. Xue, Q. Chen, F. Yang, and Z. Xiao, *Understanding the weakness of large language model agents within a complex android environment*, 2024. arXiv: 2402.06596.

[46] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, *Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing*, 2021. arXiv: 2107.13586 [cs.CL].

[47] Z. Song, X. Shang, M. Li, R. Chen, H. Li, and S. Guo, "Do not have enough data? an easy data augmentation for code summarization," in *2022 IEEE 13th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*, 2022, pp. 1–6.

[48] Q. Team, *Code with codeqwen1.5*, Apr. 2024. [Online]. Available: https://qwenlm.github.io/blog/codeqwen1.5/.

[49] A. Q. Jiang, A. Sablayrolles, A. Mensch, *et al.*, *Mistral 7b*, 2023. arXiv: 2310.06825.

[50] J. Liu, J. L. Tian, V. Daita, *et al.*, *Repoqa: Evaluating long context code understanding*, 2024. arXiv: 2406.06025.

[51] J. A. Galindo, A. J. Dominguez, J. White, and D. Benavides, "Large language models to generate meaningful feature model instances," in *Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume A*, 2023, pp. 15–26.

[52] 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 (ICSE)*, 2023, pp. 794–805.

[53] S. R. Moghaddam and C. J. Honey, *Boosting theory-of-mind performance in large language models via prompting*, 2023. arXiv: 2304.11490.

[54] C. Wang, L. Zhang, and X. Zhang, "Multi-grained contextual code representation learning for commit message generation," *Information and Software Technology*, vol. 167, p. 107 393, 2024.

[55] A. Kong, S. Zhao, H. Chen, *et al.*, *Better zero-shot reasoning with role-play prompting*, 2024. arXiv: 2308.07702.

[56] L. Fan, J. Liu, Z. Liu, D. Lo, X. Xia, and S. Li, *Exploring the capabilities of llms for code change related tasks*, 2024. arXiv: 2407.02824.

[57] D. C. Howell, *Chi-square test: Analysis of contingency tables.* 2011.

[58] L. Myers and M. J. Sirois, "Spearman Correlation Coefficients, Differences between," in *Wiley StatsRef: Statistics Reference Online*, John Wiley & Sons, Ltd, 2014.