Benchmarking LLM Commit Message Generation through a Developer-centric Pairwise Preference Framework

Lucas Aguiar
State University of Ceará (UECE)
Fortaleza, Ceara, Brazil
lucas.aguiar@aluno.uece.br

Matheus Freitas
State University of Ceará (UECE)
Fortaleza, Brazil
fernandes.matheus@aluno.uece.br

Matheus Paixao
State University of Ceará (UECE)
Fortaleza, Ceara, Brazil
matheus.paixao@uece.br

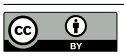
Rafael Carmo
Federal University of Ceará (UFC)
Fortaleza, Ceara, Brazil
carmorafael@virtual.ufc.br

Abstract

Commit Messages are crucial for software maintenance, as they serve both as a means of communication between developers and as a way to document software evolution. However, writing good commit messages is time-consuming and non-trivial. Solutions that employ Large Language Models (LLMs) have been proposed to assist developers in this task, demonstrating good performance in automatically generating commit messages. However, benchmarking these models remains a significant challenge. Research has demonstrated that developer-written commit messages in literature datasets vary in quality and often contain low-quality messages. Furthermore, traditional evaluation metrics, such as BLEU and ROUGE, which compare a generated message to a developer-written message, fail to capture the quality that developers perceive as important. To address this gap, we propose an evaluation framework based on the Elo Rating system that measures commit quality through direct pairwise comparisons, reflecting genuine developer preferences. In our empirical study, we evaluated commit messages written by developers and also three automated solutions: OMG (LLM-based), FIRA (traditional Neural Network-based), and NNGen (Retrieval-Based). Our analysis revealed a significant misalignment between developer preferences and traditional metrics: developers preferred the LLM-based solution, while traditional metrics favored the Neural Network solution (RQ1). The Elo scores offered valuable insights into developers' preferences, revealing not only a strong inclination towards the detailed messages generated by OMG over developers', but also predicting that OMG would be preferred over 80% of the time against developer messages in certain criteria (RQ2). Our findings indicate that developer-centric frameworks, such as our Elo-based approach, are essential for more accurate benchmarking of Large Language Models (LLMs) in code-related tasks.

Keywords

Automatic Code Summarization, Commit Message Generation, Quality Evaluation, Elo Rating System



LLM4Code '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2412-1/2026/04
https://doi.org/10.1145/3786181.3788728

ACM Reference Format:
Lucas Aguiar, Matheus Freitas, Matheus Paixao, and Rafael Carmo. 2026. Benchmarking LLM Commit Message Generation through a Developer-centric Pairwise Preference Framework. In *3rd International Workshop on Large Language Models For Code (LLM4Code '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3786181.3788728

1 Introduction

Modern software systems are characterized by immense complexity, making code comprehension a dominant activity in developers' workflows, accounting for approximately 58% of their time [27]. Automatic code summarization, which generates natural language descriptions of source code [11], has emerged as a key technique to alleviate this burden [23]. These summaries benefit developers during development, including facilitating cooperation and modification of others' code [31], accelerating code comprehension for developers [12], automatically generating documentation [20], and, most importantly for this work, generating summaries about source code changes [7].

High-quality commit messages are crucial for software maintenance and evolution [13, 28]. They are concise summaries of code changes that serve as a documentation of software evolution [28] and as a means of communication between developers [13]. Studies have shown that a good commit message should include an explanation of the changes, the "Why", and a summary of the changes, the "What" [7, 21, 26]. In addition, commit messages should be brief and grammatically correct [14]. However, developers often overlook commit messages, which frequently lack essential information due to various factors, such as time constraints or lack of motivation [10, 13, 16, 17, 19].

This scenario has spurred significant research into Automatic Commit Message Generation (ACMG) solutions, including retrieval-based models, traditional machine learning models, and, more recently, Large Language Models (LLMs) [14, 23]. However, evaluating the quality of these solutions presents a major hurdle [29]. The field has historically relied on automated metrics from machine translation [18, 29], such as BLEU [22], ROUGE [15], and METEOR [4], which measure n-gram overlap, or simply lexical similarity, between a generated message and a reference, in this case, a developer-written commit message.

The core problem with this approach is twofold. First, the "ground truth" is often flawed. Studies indicate that a large percentage of developer-written commit messages are of poor quality [26]. Consequently, a model that generates a genuinely superior message may be penalized for deviating from a suboptimal reference [14, 30]. Second, these metrics fail to capture the semantic and structural qualities that developers deem essential, such as the quality of the explanation for a change ("Why") or the quality of the summary of changes ("What") [14].

This misalignment was highlighted in the paper that introduced the OMG model [14]. In their work, developers were asked to evaluate the quality of commit messages on a scale from 0 (poor) to 4 (excellent). The data analysis showed that the LLM-based approach scored higher than developer and Neural Network solution messages, but performed the worst on automated metrics because its messages were more detailed than the developer references. This discrepancy highlights the pressing need for more effective methods to benchmark LLMs on ACMG tasks.

This paper addresses this challenge by proposing and validating an evaluation framework for ACMG based on the Elo Rating system. We expand on the Likert scale evaluation employed by Li et al. [14] by reducing the burden on the evaluator. Instead of relying on a subjective number, we adopt a more direct approach: pairwise preference selection with Elo ratings for each solution.

Popularized in chess, the Elo Rating is a method for calculating the relative skill of players through direct pairwise comparisons. This paradigm has gained new relevance for evaluating generative models [3, 5]. Platforms like Chatbot Arena [8] demonstrate the effectiveness of this type of rating by creating a public leaderboard[1] for general-purpose LLMs based on thousands of human preference votes. By adapting this proven methodology, we frame ACMG evaluation as a "competition". We select four commit message sources to have "matches" against each other, and a developer judge decides the winner based on established quality criteria. This approach provides a more faithful proxy for genuine developer preference.

In this study, we seek to answer the following research questions:

- RQ1: How does the ranking of ACMG methods produced by the Elo-based framework compare to rankings produced by traditional automated metrics?
- RQ2: What is the difference in Elo Rankings for different commit message quality criteria?

The main contribution of this work is a validated, reference-free evaluation methodology for ACMG solutions that more accurately reflects developers' preferences. By adopting this approach, we shift away from lexical-similarity metrics that penalize ACMG solutions generated by LLMs, as their messages are more detailed than those of humans [30]. This way, we provide the foundation for an empirical analysis that demonstrates how automated metrics for ACMG are misaligned with what developers find useful.

This paper is structured as follows: Section 2 outlines the fundamental concepts necessary for understanding the work; Section 3 details the methodology employed in this study; Section 4 presents the results obtained from the methodology; Section 5 discusses the potential threats to validity; and finally, Section 6 summarizes the conclusions, contributions, and future directions of the research.

[1] https://lmarena.ai/leaderboard/

2 Background and Related Work

2.1 Evaluating ACMG

Historically, the literature has benchmarked Automatic Commit Message Generation (ACMG) models using metrics borrowed from machine translation [25, 29], such as BLEU [22], ROUGE [15], and METEOR [4]. These metrics operate on a simple principle: measuring the lexical overlap (n-gram similarity) between a generated message and a single developer-written message used as a "ground truth" reference.

The fundamental flaw in this paradigm is its reliance on a reference that is itself often unreliable. A high-quality commit message should clearly explain not only what was changed but also why the change was necessary [26]. However, developers frequently neglect this documentation due to time constraints or a lack of motivation, resulting in poor messages. Studies confirm this, indicating that as many as 44% of developer-written commit messages in widely-used datasets are of suboptimal quality [26]. In addition, 14% of commit messages across 23,000 projects were empty [10], and message quality in 17 popular Apache projects declined over time [13]. Consequently, when using traditional evaluation metrics, a model that generates a genuinely superior, more detailed message will be unfairly penalized for deviating from a poor-quality reference.

Recent research has empirically demonstrated this misalignment. Tao et al. [25] highlighted the lack of standardization in BLEU implementations and confirmed their imperfect correlation with human judgment. The most compelling evidence of this paradox comes from Li et al. [14]'s work on the OMG model, a state-of-the-art LLM-based approach. In their evaluation, human evaluators scored messages generated by OMG higher on the Likert scale for comprehensiveness and rationality. Yet, when evaluated using automated metrics, such as BLEU and ROUGE, the model performed poorly precisely because its outputs were more detailed and informative than the human references.

2.2 Elo Rating

The Elo Rating system, originally developed for chess [2], is a robust method for calculating the relative skill levels of players in competitive, zero-sum games. Its principles have been successfully adapted for evaluation in computer science [24], particularly for ranking generative models [3, 5].

The Elo system operates on the basic idea that each player receives a numerical rating indicating their skill level at a given moment. When two players compete, the difference in their ratings is used to figure out the expected score for each player. Therefore, a higher-rated player is expected to score more wins against a lower-rated opponent when playing multiple games. The system changes dynamically: after each game, points are redistributed, with the winner gaining points and the loser losing them. The amount of exchanged points depends on how different the actual result was from what was expected. Unexpected results, like a lower-rated player beating a higher-rated one, yield a bigger shift in points, while a predictable result yields fewer points.

Given the characteristics of the Elo Rating system, it is possible to calculate the ranking of players in a competition without every player having to face each other [2]. For example, given a competition match between two players, A and B, each with their

respective Elo scores R_A and R_B, the expected outcome of a match between them is calculated as follows:

$$E_A = \frac{1}{1 + 10^{\frac{(R_B - R_A)}{400}}} \tag{1}$$

$$E_B = \frac{1}{1 + 10^{\frac{(R_A - R_B)}{400}}} \tag{2}$$

The variables E_A and E_B are the expected outcome of each respective player winning, assuming a value between 0 and 1. The constant 400 is a factor that adjusts the sensitivity to differences in the scores of the players in the match. A 400-point advantage in the rating equates to a 10:1 probability in favor of the higher-rated player, while a match where players have equal scores equates to a 50:50 probability of victory for both [5].

After obtaining the expected probability of victory, it is possible to calculate the updated scores R'_A and R'_B by verifying the player who won the match and using the following equations:

$$R'_A = R_A + K(S_A - E_A) \tag{3}$$

$$R'_B = R_B + K(S_B - E_B) \tag{4}$$

S_A and S_B represent the match's outcome, and can take a binary value of 1 or 0 in a match where there are no draws, being 1 if the player was the winner and 0 otherwise. The K-factor serves as a hyperparameter to adjust the pace at which the score changes, thereby altering the speed at which the score converges or mitigating the impact of unusual cases on the score.

In summary, the Elo system provides a dynamic and iterative method for establishing a performance hierarchy. By applying the predictive scoring (Equations 1 and 2) and the update mechanism (Equations 3 and 4) over a series of matches, the system processes the binary outcomes of each match and adjust participant scores. This repeated adjustment, moderated by the K-factor, allows the player ratings to converge to a score that accurately represents each player's performance. The final calculated Elo rank, therefore, represents a stable and relative measure of skill derived from these various matches, without requiring a complete round-robin tournament structure.

3 Methodology

This section outlines the methodology for evaluating and ranking Automatic Commit Message Generation (ACMG) solutions using the Elo Rating System. The process, illustrated in Figure 1, comprises three main phases: (1) **Data and Experiment Preparation**, where we detail data collection and model selection; (2) the **Empirical Experiment**, involving developer evaluators; and (3) the **Elo Rating calculation** to rank the solutions. All the code and data necessary to replicate this study are available in our replication package [1].

3.1 Data and Experiment Preparation

Our study utilizes a dataset derived from the OMG paper [14], which contains commit messages from 32 Apache projects, like Mesos and Cassandra. The original dataset provided 381 commit messages and

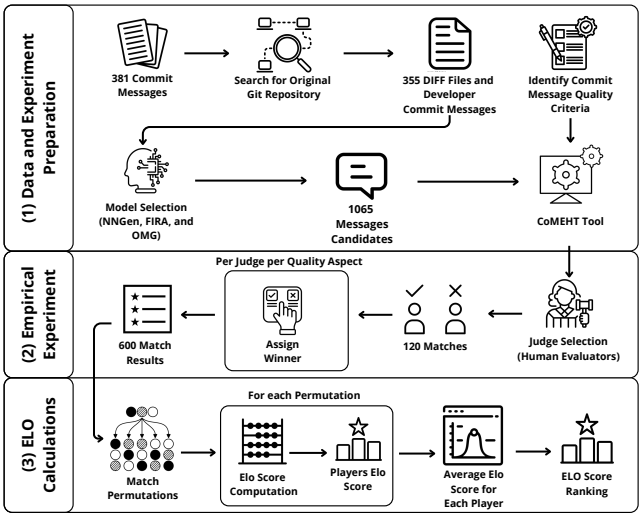


Figure 1: Overview of the Research Methodology

Table 1: Number of commit messages across all projects in the dataset.

Project	URL	# Commit Messages
ambari	https://github.com/apache/ambari	1
ant	https://github.com/apache/ant	19
archiva	https://github.com/apache/archiva	6
aries	https://github.com/apache/aries	1
beam	https://github.com/apache/beam	15
Cassandra	https://github.com/apache/cassandra	16
cocoon	https://github.com/apache/cocoon	11
cxf	https://github.com/apache/cxf	15
directory-server	https://github.com/apache/directory-server	16
flink	https://github.com/apache/flink	13
Geronimo	https://github.com/apache/geronimo	12
hadoop	https://github.com/apache/hadoop	1
ignite	https://github.com/apache/ignite	7
isis	https://github.com/apache/causeway	4
jclouds	https://github.com/apache/jclouds	11
jena	https://github.com/apache/jena	13
JMETER	https://github.com/apache/jmeter	22
karaf	https://github.com/apache/karaf	3
Lenya	https://github.com/apache/lenya	10
logging-log4j2	https://github.com/apache/logging-log4j2	20
maven	https://github.com/apache/maven	13
mesos	https://github.com/apache/mesos	1
poi	https://github.com/apache/poi	12
qpid	https://github.com/apache/qpid	3
storm	https://github.com/apache/storm	5
synapse	https://github.com/apache/synapse	5
tomcat	https://github.com/apache/tomcat	43
tomee	https://github.com/apache/tomee	16
usergrid	https://github.com/apache/usergrid	6
wicket	https://github.com/apache/wicket	35

their corresponding SHAs, but lacked the code changes (diffs) and repository URLs.

To obtain the necessary context, we both manually searched for the original repository and programmatically retrieved the original commits and their code changes using the commit SHA. This process successfully recovered 355 commits and their respective diffs across 30 projects. Commits that could not be recovered were

affected by either destructive modifications to the repository history or by the original repository URL not being found. The final dataset content is detailed in Table 1.

We evaluated four sources of commit messages, hereafter referred to as players: the original developer-written messages and messages generated by three distinct ACMG models. The models were selected to represent a variety of approaches:

- **OMG** [14]: A state-of-the-art LLM-based ACMG solution that employs a ReAct prompting.
- **FIRA** [9]: An ACMG solution that employs traditional Neural Networks, which represent code changes with graphs.
- **NNGen** [17]: A classical retrieval-based model that employs the Nearest Neighbor algorithm to retrieve the most closely related message from its index of existing commit messages.

Messages from OMG and FIRA were extracted directly from the dataset provided by Li et al. [14]. For NNGen, we used the original authors' publicly available implementation to generate message candidates for our set of code changes. The final experimental dataset included the 355 original developer-written messages and 1,065 corresponding messages generated by the three ACMG models (355 messages per model).

The criteria we chose to evaluate the quality of commit messages were based both on literature and a survey conducted by Li et al. [14], which explores the information developers expect from effective commit messages. The selected evaluation criteria for this study are the following:

- **Rationality**: the message needs to provide a clear and logical explanation for the code changes, explaining the reasoning behind them, and the nature of the change. Popularly known as the "Why".
- **Comprehensiveness**: the message should provide a summary of the changes, including all relevant details necessary for understanding the modifications. Popularly known as the "What".
- **Conciseness**: the message should convey information succinctly, ensuring readability and quick comprehension.
- **Expressiveness**: the message content should be grammatically correct.
- **Judge's Selection**: overall preference for a commit message, mitigating limitations of the other previous criteria, and ultimately, is the one that the judge would use from the available candidates.

Evaluating the five distinct criteria for each pairwise comparison of commit messages could be a daunting process. To streamline this process and alleviate the cognitive burden, we developed a tool called the Commit Message Evaluation Helper Tool, or CoMEHT (pronounced as COMET).

CoMEHT, as shown in Figure 2, serves as a Graphical User Interface, offering structured and more readable information. For each **Match**, CoMEHT displays the number of files changed and illustrates the changes introduced by the commit by showing the BEFORE (2) and AFTER (3) versions of the selected file (1). We explicitly show the current criterion (6) to be evaluated in the commit message pair (4) and (5). At the bottom, we present the buttons (7) where the judge will click to select the message that he/she believe to be the best for the current criterion. Clicking "A is better" will compute a win for Player A on the current criterion. Otherwise, "B is better" will compute a win for Player B. It is important to note that the evaluator cannot identify which commit message sources they are evaluating.

3.2 Empirical Experiment

To avoid bias in judge selection, we selected five Developers with varying experience: one with 2 years, two with 4 years, and two with more than 9 years of experience in software development. They also had various fields of expertise: one tester, three backend developers, and one full-stack developer. All of them reported some degree of familiarity with the Java language, which was the language of the evaluated projects.

The experiment was then performed in two steps. In the first step, we conducted a presentation that allowed the judges to explore CoMEHT's UI while we explained the evaluation criteria. Additionally, we selected 8 matches for them to evaluate, enabling them to familiarize themselves with the process. This step took about 30 minutes, and the played matches were both disregarded in our analysis and removed from the final dataset.

In the second step, the judges evaluated 120 matches, selecting the better message from each pair based on their preferences across the five criteria. The number of matches is the result of a previously conducted pilot, which showed that 120 matches were enough for the Elo Ranking to converge between four players. All players had the same number of matches, and each pair played 20 games. On average, the second step took approximately 2 hours and 50 minutes to complete, and all 600 match results (120 matches for each of the 5 criteria) were used to compute the Elo Ranking.

3.3 Elo Ranking Calculation

To transform the raw pairwise comparison data from the empirical experiment into a stable and meaningful ranking, we implemented a rigorous calculation process based on the Elo Rating system. This process was specifically designed to address the inherent challenges of the Elo system, particularly its sensitivity to the sequence of comparisons.

Following the best practices for using Elo Rating, our primary step was to mitigate the potential for order bias [5]. To achieve this, we generated 100 unique permutations of the complete set of 600 match results. For each shuffled sequence, we calculated the Elo scores for the four players (Developer, FIRA, NNGen, and OMG). In these calculations, we utilized a K-factor of 16, a value that, in the literature, yielded more consistent and accurate rankings in evaluating LLMs [5]. All players received a common initial score of 1400.

This procedure resulted in 100 distinct Elo scores for each player across each of the five quality criteria. The final, definitive Elo score for each player and each criterion was then computed by averaging these 100 scores. This averaging step is critical as it ensures that the final ranking is robust and represents a stable measure of quality, independent of the random order in which the matches were processed.

While we opted for the Elo Rating system for this phase, for reasons explained in Sections 1 and 2.2, our methodology is not limited to it. We recognize that other rating systems are available,

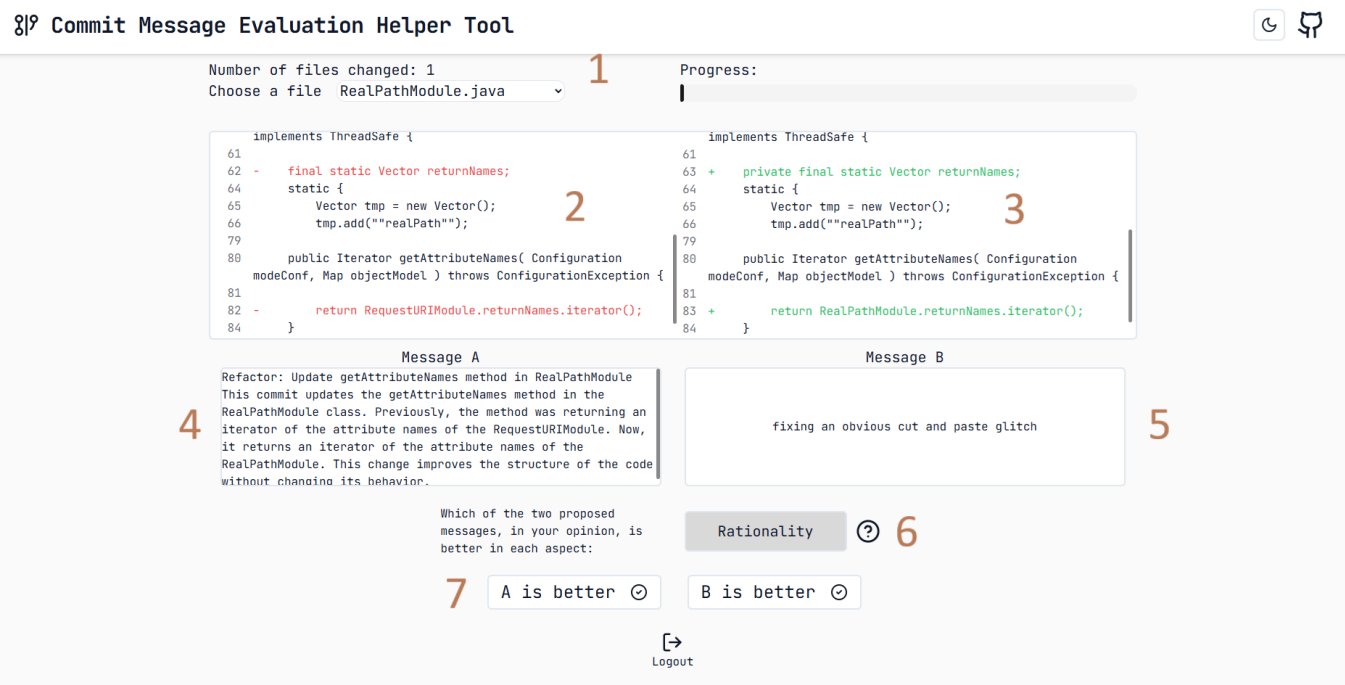


Figure 2: CoMEHT: An auxiliary platform designed to assist judges in evaluating commit message sources. It presents important information in an organized manner while continuously saving the match results.

such as the Bradley-Terry method [6]. Our framework can be easily adapted with minimal modifications to accommodate any new and improved rating systems for evaluating ACMG.

4 Results

In this section, we present the results of our empirical study. The analysis is structured to answer our two research questions: first, we compare the rankings produced by our Elo-based framework against those from traditional automated metrics (RQ1); second, we examine the Elo scores for the commit quality criteria to derive insights into developers' preferences for commit messages (RQ2).

4.1 RQ1: How does the ranking of ACMG methods produced by the Elo-based framework compare to rankings produced by traditional automated metrics?

To answer our first research question, we established a developer-centric benchmark by evaluating four commit message sources (OMG, FIRA, NNGen, and Developer) using our Elo-based framework. The analysis of the aggregated Elo scores, shown in Figure 3, reveals a consistent hierarchy of developer preference across the five quality criteria.

The results indicate a clear preference for the LLM-based OMG model, which ranked first in four of the five criteria: Comprehensiveness, Expressiveness, Rationality, and the holistic Judge's Personal Selection, surpassing even developer-written messages. The messages from FIRA and Developer sources were ranked closely,

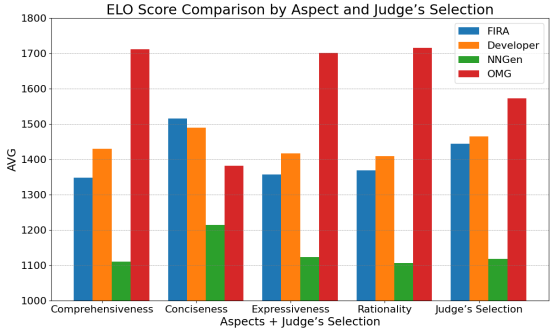


Figure 3: Player's average Elo-score between all judges and 100 iterations.

occupying the middle ground, with FIRA notably outperforming Developer messages in Conciseness. Lastly, the retrieval-based NNGen was consistently the least preferred solution across all criteria. The final ranking from the Judge's Personal Selection, which represents the Judges' ultimate choice, is shown in Table 2

Next, we calculated the scores for the traditional automated metrics. We employed BLEU [22], specifically B-Norm [25], ROUGE-L [15], and METEOR [4]. For the three models, we used the developer-written messages as the ground truth reference. This allowed us to generate results from the perspective of conventional evaluation methods, as shown in Table 2.

Table 2: Player ranking through different metrics.

	Metrics			
Source	**Elo**	**BLEU**	**ROUGE-L**	**METEOR**
OMG	1st (1573.1)	2nd (0.009)	3rd (0.128)	1st (0.210)
Developer	2nd (1464.6)	-	-	-
FIRA	3rd (1443.8)	1st (0.011)	1st (0.314)	2nd (0.205)
NNGen	4th (1118.4)	3rd (0.005)	2nd (0.247)	3rd (0.136)

The results show that FIRA achieved the highest position in both BLEU and ROUGE-L with 0.011 and 0.314, respectively. These automated metrics calculate the closeness of the generated messages to the original messages, indicating that FIRA and Developer messages have highly related content.

However, despite being the clear winner in our Elo Framework, the OMG solution ranked last in ROUGE-L and second in BLEU. This occurs because OMG messages, while rated higher in informational content by Judges (being preferred in both Rationality and Comprehensiveness), are more verbose and detailed than the typically brief human references, leading to a penalty for their lack of similarity.

The automated metric METEOR, which didn't penalize OMG's longer messages, failed to accurately reflect the difference in preference between OMG and FIRA, yielding similar scores for both: 0.205 and 0.210, respectively. In contrast, Elo allows us to properly estimate the difference between these two models: OMG has a 65% chance of being preferred overall over developer messages, as reported in Figure 4.

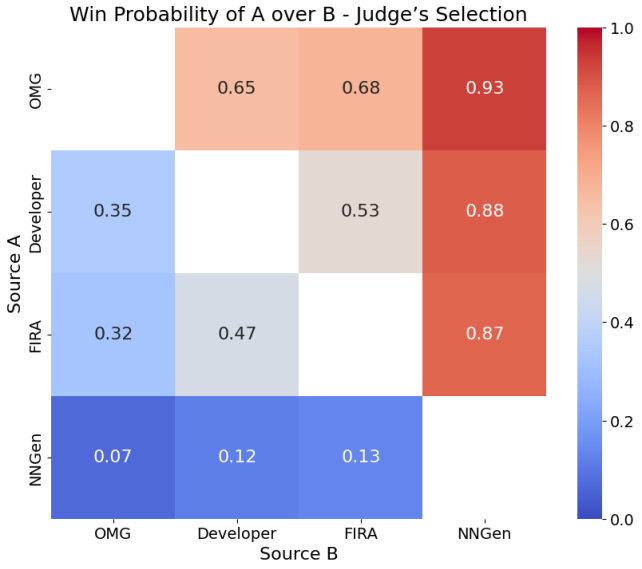


Figure 4: Win probability of each player against the other in the criterion of Judge's Selection

Response for RQ1: There is a fundamental misalignment between traditional automated metrics and genuine developer preference. The Elo-based framework reveals that developers prioritize the detailed nature of LLM-generated messages. In contrast, metrics like ROUGE-L penalize the very characteristics that make these messages valuable. This demonstrates that human-centric, reference-free methods, such as our Elo-based Framework, are better suited for benchmarking modern ACMG solutions.

4.2 RQ2: What is the Difference in ELO Rankings for Different Commit Message Quality Criteria?

The evaluation of Comprehensiveness, which assesses a message's ability to provide a complete summary of all relevant changes, the "what", found OMG as the undisputed winner, achieving the highest Elo score among all sources. Figure 5 quantifies this dominance by detailing the win probabilities from direct comparisons. OMG demonstrated massive superiority, showing an 83% win probability against Developer-written messages, 89% against FIRA, and a near-total 97% against NNGen. Developer-written messages ranked second, consistently being more comprehensive than FIRA, with a 62% win probability in their matchups.

In the Rationality criterion, which assesses the message's ability to explain the "why" behind the code change, the OMG model ranked first. The quantitative results, presented in Figure 6, underscore OMG's superiority, achieving an expected win probability of 85% against Developer-written messages and 88% against FIRA. The second position was occupied by Developer messages, which held a slight edge over FIRA (56% win probability), suggesting a very similar perceived preference between the two.

The Conciseness criterion was the only aspect where OMG did not rank first, revealing a clear trade-off between informational depth and brevity. Figure 7 illustrates this reversal in preference, with FIRA emerging as the top performer, achieving a 68% win probability against OMG and 54% against Developer messages. Developer messages were also rated as significantly more concise than OMG's (65% win probability), indicating that OMG's verbosity was perceived as a drawback.

Response for RQ2: Our findings reveal that the Elo rankings differed across quality criteria, highlighting a critical trade-off in developer preferences. The LLM-based model, OMG, emerged as the dominant performer, ranking first in Comprehensiveness, Rationality, and Expressiveness. However, this trend was completely inverted for the Conciseness criterion, where OMG was rated poorly for its lengthy messages, while FIRA and Developer-written messages were favored. This discrepancy demonstrates that no single model excelled across all attributes, underscoring an inherent tension between depth of information and brevity. Crucially, despite OMG's low Conciseness score, it regained the top rank in the holistic Judge's Personal Selection, indicating that developers prioritize context-rich, in-depth messages over pure brevity.

5 Threats to Validity

We acknowledge the potential limitations in our study, which we report as follows:

External Validity: Our primary limitation is the scope of the empirical study. The evaluation was conducted with five judges, all of whom were familiar with Java. While the participants were experienced professionals from diverse software development backgrounds and varying experience levels, the small sample size limits

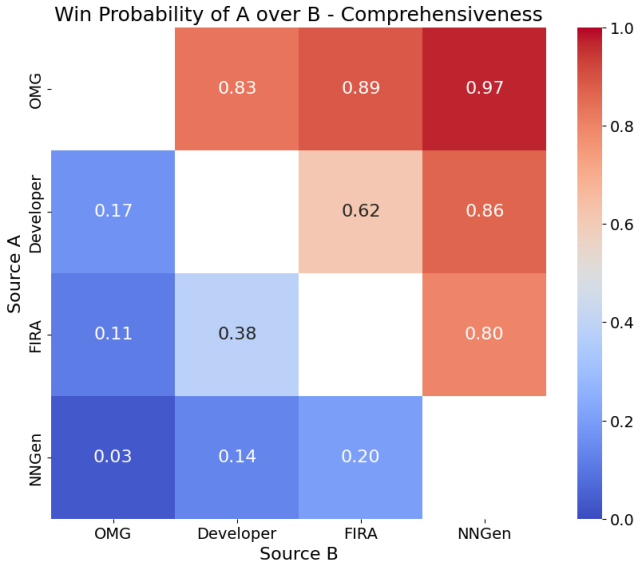


Figure 5: Win probability of each player against the other in the criterion of comprehensiveness

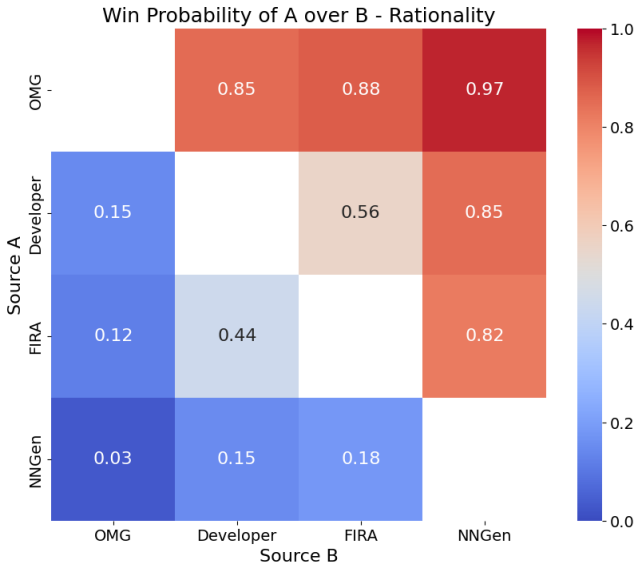


Figure 6: Win probability of each player against the other in the criterion of rationality

the generalization of our findings to the broader developer community. Furthermore, our dataset consisted of 120 code changes, which we determined through a pilot. While the results showed stable rankings at 120 matches, it is possible that with more matches, the rankings would change. Second, the dataset's scope is limited to Java-based projects from the Apache Software Foundation. This lack of diversity limits the generalization of our findings, as the identified preferences and source performance may not be representative across other programming languages and projects.

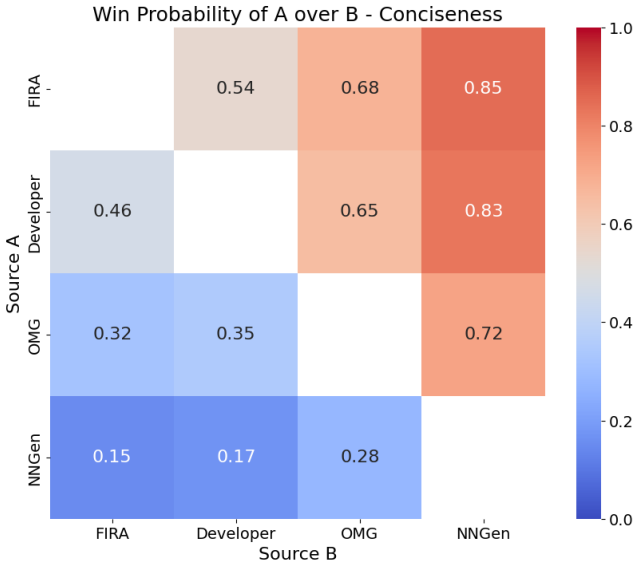


Figure 7: Win probability of each player against the other in the criterion of conciseness

Construct Validity: First, our definition of commit message quality was based on five criteria derived from the literature. Although well-grounded, this set of criteria is not exhaustive and may not capture all the aspects of quality that developers perceive. Second, the Elo framework itself has inherent limitations. It provides a relative ranking of preference, not a measure of absolute quality.

Internal Validity: A potential threat arises from the subjectivity of human evaluation. To mitigate this, we selected five participants with diverse backgrounds and an average of 5 years of software development experience. The judges' consensus on most criteria (e.g., Rationality, Comprehensiveness) suggests this was effective. Another threat is the Elo system's sensitivity to the sequence of comparisons. We mitigated this by generating 100 unique permutations of the 600 match results and averaging the final Elo scores, ensuring the final ranking is robust and stable. Finally, the Elo calculation is sensitive to the K-factor parameter. We addressed this by adopting a K-factor of 16. This value yielded more consistent rankings when evaluating generative models in the literature [5]. Still, other K factors can be explored as we balance the number of matches needed with the best representation of the player's performance.

6 Conclusion

Evaluating the quality of Automatic Commit Message Generation (ACMG) systems, particularly those powered by LLMs, is a significant challenge. Traditional metrics such as BLEU and ROUGE are often misaligned with developer preference, as they rely on lexical similarity to Developer-written references of varying quality. This paper addressed this gap by proposing and validating a novel, reference-free evaluation framework based on the Elo Rating system, designed to benchmark ACMG solutions through direct pairwise comparisons that capture genuine developer preference.

Our empirical study revealed a fundamental misalignment between our Developer-centric Elo rankings and those produced by traditional automated metrics (RQ1). The LLM-based OMG model, which ranked highly across most developers' criteria, performed poorly on metrics such as ROUGE-L and BLEU, which penalize its informative verbosity. Furthermore, our granular, criterion-based analysis provided key insights into the nature of developer preferences (RQ2). We found that while developers value conciseness, they were willing to trade it for informational completeness.

The primary contribution of this work is a validated, reference-free evaluation methodology for ACMG that serves as a more faithful and insightful proxy for developer preference than traditional metrics. Our findings provide strong empirical evidence that the current paradigm for evaluating code summarization tasks may be focusing on the wrong target. By shifting the focus from lexical similarity to Developer-centric, preference-driven assessment, frameworks like the one presented in this paper can pave the way for developing LLM-powered tools that are not just technically proficient but genuinely more useful to the software engineers who rely on them.

For future work, we envision creating a unified platform to evaluate ACMG solutions. We also plan to explore various rating systems, like the Bradley-Terry method [6]. Ultimately, we aim to gather a larger sample of evaluators to better capture the nuances of developer preferences. Additionally, we will investigate the potential of using LLMs to replace human judges in the evaluation process, thereby streamlining the assessment.

Acknowledgments

This work received partial funding from CNPq-Brazil, Universal grant 404406/2023-8

References

[1] Lucas Almeida Aguiar, Francisco Matheus Fernandes Freitas, Matheus Paixao, and Rafael Augusto Ferreira do Carmo. 2026. Scripts, Software and Dataset for "Benchmarking LLM Commit Message Generation through a Developer- centric Pairwise Preference Framework". doi:10.5281/zenodo.18369469

[2] Elo Arpad. 1978. The rating of chessplayers, past and present. *Arco Pub* 216 (1978).

[3] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862* (2022).

[4] 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*. 65–72.

[5] Meriem Boubdir, Edward Kim, Beyza Ermis, Sara Hooker, and Marzieh Fadaee. 2023. Elo uncovered: Robustness and best practices in language model evaluation. *arXiv preprint arXiv:2311.17295* (2023).

[6] Ralph Allan Bradley and Milton E. Terry. 1952. Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. *Biometrika* 39, 3/4 (1952), 324–345. http://www.jstor.org/stable/2334029

[7] Raymond PL Buse and Westley R Weimer. 2010. Automatically documenting program changes. In *Proceedings of the 25th IEEE/ACM international conference on automated software engineering*. 33–42.

[8] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference. arXiv:2403.04132 [cs.AI] https://arxiv.org/abs/2403.04132

[9] Jinhao Dong, Yiling Lou, Qihao Zhu, Zeyu Sun, Zhilin Li, Wenjie Zhang, and Dan Hao. 2022. FIRA: 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. doi:10.1145/3510003.3510069

[10] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N Nguyen. 2013. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 422–431.

[11] David Gros, Hariharan Sezhiyan, Prem Devanbu, and Zhou Yu. 2020. Code to comment" translation" data, metrics, baselining & evaluation. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 746–757.

[12] Sonia Haiduc, Jairo Aponte, Laura Moreno, and Andrian Marcus. 2010. On the use of automated text summarization techniques for summarizing source code. In *2010 17th Working conference on reverse engineering*. IEEE, 35–44.

[13] Jiawei Li and Iftekhar Ahmed. 2023. Commit message matters: Investigating impact and evolution of commit message quality. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 806–817.

[14] Jiawei Li, David Faragó, Christian Petrov, and Iftekhar Ahmed. 2024. Only diff is not enough: Generating commit messages leveraging reasoning and action of large language model. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 745–766.

[15] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[16] Shangqing Liu, Cuiyun Gao, Sen Chen, Lun Yiu Nie, and Yang Liu. 2020. Atom: Commit message generation based on abstract syntax tree and hybrid ranking. *IEEE Transactions on Software Engineering* 48, 5 (2020), 1800–1817.

[17] Zhongxin Liu, Xin Xia, Ahmed E Hassan, David Lo, Zhenchang Xing, and Xinyu Wang. 2018. Neural-machine-translation-based commit message generation: how far are we?. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 373–384.

[18] Pablo Loyola, Edison Marrese-Taylor, and Yutaka Matsuo. 2017. A neural architecture for generating natural language descriptions from source code changes. *arXiv preprint arXiv:1704.04856* (2017).

[19] Walid Maalej and Hans-Jörg Happel. 2010. Can development work describe itself?. In *2010 7th IEEE working conference on mining software repositories (MSR 2010)*. IEEE, 191–200.

[20] Paul W McBurney and Collin McMillan. 2014. Automatic documentation generation via source code summarization of method context. In *Proceedings of the 22nd International Conference on Program Comprehension*. 279–290.

[21] Mockus and Votta. 2000. Identifying reasons for software changes using historic databases. In *Proceedings 2000 international conference on software maintenance*. IEEE, 120–130.

[22] 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 of the Association for Computational Linguistics*. 311–318.

[23] Weisong Sun, Yun Miao, Yuekang Li, Hongyu Zhang, Chunrong Fang, Yi Liu, Gelei Deng, Yang Liu, and Zhenyu Chen. 2024. Source code summarization in the era of large language models. *arXiv preprint arXiv:2407.07959* (2024).

[24] Lei Tan, Yuqing Lin, and Li Liu. 2014. Quality ranking of features in software product line engineering. In *2014 21st Asia-Pacific Software Engineering Conference*, Vol. 2. IEEE, 57–62.

[25] Wei Tao, Yanlin Wang, Ensheng Shi, Lun Du, Shi Han, Hongyu Zhang, Dongmei Zhang, and Wenqiang Zhang. 2021. On the evaluation of commit message generation models: An experimental study. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 126–136.

[26] 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.

[27] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E Hassan, and Shanping Li. 2017. Measuring program comprehension: A large-scale field study with professionals. *IEEE Transactions on Software Engineering* 44, 10 (2017), 951–976.

[28] Shengbin Xu, Yuan Yao, Feng Xu, Tianxiao Gu, Hanghang Tong, and Jian Lu. 2019. Commit message generation for source code changes. In *IJCAI*.

[29] Xuejun Zhang, Xia Hou, Xiuming Qiao, and Wenfeng Song. 2024. A review of automatic source code summarization. *Empirical Software Engineering* 29, 6 (2024), 162.

[30] Yuxia Zhang, Zhiqing Qiu, Klaas-Jan Stol, Wenhui Zhu, Jiaxin Zhu, Yingchen Tian, and Hui Liu. 2024. Automatic commit message generation: A critical review and directions for future work. *IEEE Transactions on Software Engineering* (2024).

[31] Yuxiang Zhu and Minxue Pan. 2019. Automatic code summarization: A systematic literature review. *arXiv preprint arXiv:1909.04352* (2019).