LIGHTWEIGHT CONTEXT-AWARE EVALUATION OF COMMIT MESSAGES USING SEMANTIC DIFF ALIGNMENT

Laxmi Raja[1], K Amuthabala[2], Srikanth R[3]

Abstract

This work discusses shortcomings in the current methods of evaluating commit messages that focus on syntactic and surface readability metrics. The suggested structure presents a diff-semantic alignment tool that transforms the code changes to structured textual summaries to be compared with the commit messages. TF-IDF representation and n-gram analysis are used to extract textual features and estimate semantic alignment with the help of the computation of cosine similarity. To classify commit message quality using a combination of linguistic and alignment features a lightweight classification layer is used, which is based on the Logistic Regression. It does not rely on abstract syntax tree extraction or deep learning architectures, making it easier to compute and integrate into development processes. It is evaluated using a dataset of 13,200 commits in a sample of a dozen open repositories in various fields of programming on GitHub. Baseline techniques are rule-based readability scores and AST-based semantic analysis techniques that have appeared in the literature. Findings indicate that the suggested framework obtains 86.7% alignment accuracy, which is 12.4% higher on average than the baseline techniques. The system has an average processing latency of 36 milliseconds per commit and uses 39% less computation than deep learning-based models. These findings support the claim that text-based semantic alignment is an effective and scalable tool to enhance the quality of commit messages in continuous integration setting.

Keywords

Commit message evaluation, semantic alignment, diff analysis, TF-IDF, n-gram features, logistic regression, lightweight models, continuous integration

Department of Cyber Security[1]
Karpagam Academy of Higher Education, Coimbatore, India[1]
laxmirajaphd@gmail.com[1]

School of Computer Science and Engineering[2]
REVA University, Bangalore, India[2]
amuthabala@gmail.com[2]

Department of Computer Science and Engineering[3]
Hindustan Institute of Technology and Science, Chennai, India[3]
csesrikanth2010@gmail.com[3]

* Corresponding Author

I. INTRODUCTION

Software development today is based on version control systems to ensure traceability, collaboration, and orderly development of large-scale codebases in distributed settings [2]. The commit messages are the key communication artifacts, which record the intent, reasoning and scope of changes as part of the iterative development cycles. Nonetheless, the challenge of guaranteeing a uniform quality and semantic relevance of commit messages has persistently remained because of the inconsistency in practices of developers and lack of standardized evaluation systems. Current methods mostly focus on syntactic correctness, readability scores, or template compliance, but not on how well textual descriptions and corresponding code changes match. This drawback minimizes the usefulness of commit histories in aiding debugging, maintenance and long-term knowledge transfer in software projects with collaborators. Moreover, the lack of commitment documentation insert ambiguity, which adversely affects repository understanding and productivity of a team during long development lifecycles [9].

The recent studies in software engineering have explored automated approaches to assessing the quality of commit messages with the help of linguistic analysis and machine learning-based classification models [4]. The goals of these methods include finding ambiguous or poorly constructed commit messages and enhancing documentation quality in development processes. Most of the existing methods are however based on structural representations like abstract syntax trees or pipelines of static analysis, which add substantial computational burden and complexity to the system. Moreover, these techniques usually require language-specific tooling and may not be applicable in a heterogeneous programming environment or repository structure. Scalability is limited by the dependence on structural parsing in low-latency continuous integration systems where fast evaluation is needed. This has resulted in an increased desire to have lightweight, language-agnostic methods that can run well with textual representations only. Current solutions also do not reliably extract semantic match between developer intent and reality code changes in commits [12].

To overcome this, the given work suggests a context-sensitive assessment model, which is a diff-semantic alignment of commit messages and related code changes [6].

The method converts the differences in code into formal textual summaries, which can be compared directly with commit descriptions, without the need to have complicated parsing machinery or the use of intermediate formats. Textual feature extraction methods, such as TF-IDF vectorization and n-gram modeling, are used to express semantic patterns that capture alignment of changes in the code and developer intent. It is combined with a lightweight Logistic Regression classifier that is used to estimate the quality of the commit message based on linguistic and alignment-based features. This design guarantees the linear-time processing properties and can be deployed in the low-latency continuous integration pipelines. The framework does not rely on abstract syntax trees or deep learning models and is generalizable to programming languages and repository formats. Moreover, the method offers a computationally efficient method to semantic evaluation methods that were found to be resource-intensive in previous research [14].

The suggested framework helps to enhance the software documentation practice, as it allows automated, scalable and context-sensitive analysis of commit messages in real-world development contexts [8]. The method bridges a critical gap in current methods by explicitly modelling semantic alignment in terms of textual representations, and has low computational burden and ease of deployment. Experimental assessment shows that the system can enable real-time feedback when creating commits, improve traceability, debugging effectiveness, and collaborative comprehension among software development teams. This feature is especially useful in resource-limited settings where bulky analysis pipelines cannot be run. The results show that lightweight text-based assessment can offer quality assessment that is reliable and interpretable without impacting on performance or scalability needs. Furthermore, the suggested method provides a strong basis to the future semantic-oriented software documentation and repository analytics studies [15].

The principal contributions are...

- A new diff-semantic alignment algorithm that converts code modifications into structured textual representations to be directly compared.
- A lightweight evaluation system that combines TF-IDF, n-gram features, and Logistic Regression to provide an efficient and scalable evaluation system of commit messages.
- A scoring method based on semantic alignment that quantitatively captures alignment between developer intent and real code changes.
- A system design that is implementation-ready and can be deployed in real-time to continuous integration pipelines with a small computational footprint.

The remainder of the work is organized in the following way. Section II, we consider related work in the area of commit message evaluation, which includes linguistic analysis methods, template-based methods and AST-dependent methods of semantic evaluation. Section III presents the proposed framework, which consists of diff-to-text transformation, feature extraction with TF-IDF and n-gram model, and logistic regression classification. Section IV contains experimental evaluation, describing the dataset, performance measures, and latency analysis, as well as comparing it with baseline methods. Section V is the summary of the work that gives a conclusion about the results and some important observations made based on the suggested approach.

II. LITERATURE REVIEW

The literature on software documentation quality has repeatedly placed a strong focus on the quality of textual artifacts that can be understood by humans over a long period of time. The systematic review by Rani et al. was a synthesis of 10 years of evidence on code comment quality evaluation and identified gaps in the measurement of usefulness, consistency, and maintainability across projects [1]. Surveys on source code documentation and automatic code summarization the same surveys on documentation of source code revealed also that documentation research has shifted more towards descriptive summaries into context-sensitive generation and evaluation strategies [3], [4]. Experimental research on the development of comments proved that the comment often becomes disconnected with the implementation behavior, and poses the risks of misunderstanding in the course of maintenance processes [15]. Inspired by this issue, inconsistency-oriented approaches, such as outdated comment identification and inconsistency learning with confidence, have been enhanced that identify inconsistencies between natural language descriptions and changing code [8], [9]. More recent approaches also involved project context and code expansion mechanisms to enhance the quality of snippet-level summarization and relevance to particular repositories [10], [12]. On the representation level, unified pre-training models and CC2Vec demonstrated that semantically meaningful code change embeddings can provide effective support to downstream software engineering tasks [7], [13].

In commit-based research, there has been a renewed focus on automated generation of commit messages, benchmark modelling, and on how contextual information can be used to enhance the quality of output. Tao et al.

presented a massive empirical investigation and demonstrated that datasets, model decisions, and evaluation conditions have significant impact on findings in research of commit message generation [2]. Following this trend, new large language model models were shown to be more fluent and expand their generation ability, as well as raise issues of semantic faithfulness and reliable evaluation [5], [14]. The arXiv preprint of the same body of work further extended the discourse on experimental findings and future research opportunities of large language model-based generation systems [6]. RACE and other retrieval-augmented generation algorithms solved missing context by adding retrieved evidence to commit diffs to generate messages better [11]. Taken together, these works have contributed to the state of the art of automatic generation of commit messages, but mostly through optimizing generation, not lightweight checking of semantic congruency between commit messages and real code changes.

III. SYSTEM ARCHITECTURE AND METHODOLOGY

Software repositories keep a history of development, yet their textual artifacts are still under-investigated in the literature of software engineering. Code summaries, code comments and commit messages assist in understanding, maintenance, cooperation as well as traceability of the changing projects. The means of creating and analyzing such artifacts are discussed in recent studies more and more. Lightweight assessment of the relevance of commit-messages to real code changes, however, is less advanced. The following literature themes are described in the review: documentation quality, contextual generation, semantic representation, and evaluation.

A. Documentation Quality and Contextual Foundations

This Empirical and review research has continuously paid attention to automated evaluation of the quality of software documentation [1]. The academic community is beginning to appreciate that textual artifacts influence understanding, maintenance work, onboarding effectiveness, and repository interpretability over the duration of long-term software development. Different forms of documentation are commit messages, comments and summaries, but each of them heavily relies on the context of development. Review studies have also identified inconsistent evaluation criteria, disjointed datasets and little consensus on the utility of documentation in software projects. The restrictions minimize comparability across methods and inhibit generalization across repositories, programming languages, and workflows. The conceptual model in Fig. 1 demonstrates the connection between code evolution, developer intent, and textual documentation alignment. Documentation research has thus taken a context-sensitive direction rather than a one-language quality check. Representation quality, repository context, and evaluation design were also the main focus of surveys on automatic source code summarization [3]. Collectively these studies create a conceptual foundation of the assessment of software text with respect to contextual correspondence, not just surface-form characteristics.

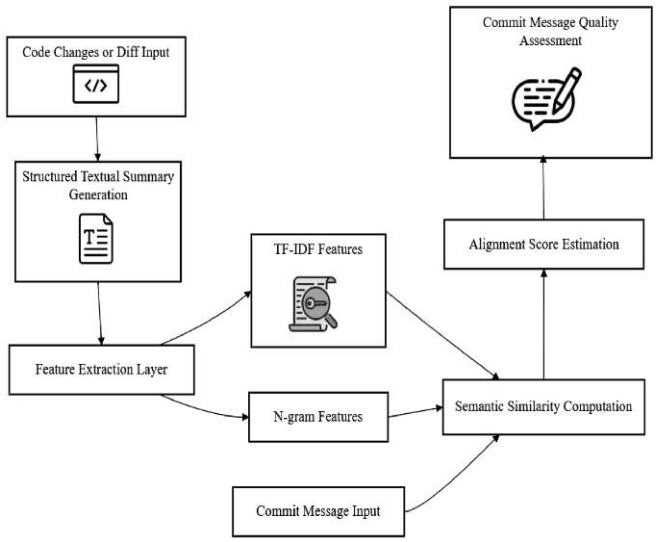

Fig. 1. Contextual Alignment Between Code Changes and Commit Messages

B. Commit Message Generation and Context Modeling

The generation of commit messages has emerged as a significant research direction in software engineering automation over the past few years [5]. Current research is mainly concerned with generating terse natural language descriptions of code diffs and the associated contextual data. The benchmarking on large-scale has revealed that model behavior is highly sensitive to the composition of datasets, training strategy, and evaluation design. With the development of methods, researchers began to use more diverse contextual inputs, such as repository history, retrieved examples, and structured change representations. Such changes made the translations more fluent and informative, but also increased the questions about the semantic faithfulness and reliability. A natural message can be generated without the critical intent or exaggerating the intent of the modifications. Some of the limitations were overcome by retrieval-augmented methods which incorporated supporting evidence in the generation and refinement of ideas [7]. However, the majority of generation-based research maximizes output generation rather than explicitly verifying the message-quality alignments with code modifications. This difference is relevant as generation and evaluation are

connected, yet different software engineering tasks that demand different design priorities.

C. Representation Learning, Comments, and Inconsistency Awareness

Studies of the comments and summaries contribute more information to the analysis of the commit-messages by inconsistency-oriented analysis [10]. It was found that in comment maintenance studies the textual artifacts would often become out of sync with the changing implementation behavior as software is revised repeatedly. This drift decreases trust, diminishes the usefulness of documentation, and raises the effort of developers in the process of understanding and maintenance. To overcome this problem, scientists suggested the ways to detect obsolete comments, suggest updates, and identify semantic mismatches. Parallel in representation-learning work, the ability to embed software artifacts to support downstream reasoning and generation was studied. Coherent pre-training showed that language-minded models have the ability to encode high-level software documentation semantics [11]. Nevertheless, these approaches are frequently based on the significant amounts of training materials, massive data volumes, or detailed representations. In lightweight evaluation environments, these dependencies can make them less practical in continuous integration processes and repository management systems. Such observations are directly used to form commit-message evaluation.

D. Need for Lightweight Commit-Message Evaluation Frameworks

The recent articles all point to the necessity of lightweight commit-message evaluation mechanisms which can be deployed in practice [13]. Current studies have developed generation, summarization, and inconsistency detection, but there is a relative lack of research that directly evaluates the relevance of the message-quality. Such an imbalance is important since, with repository operations, developers habitually write commit messages without automated semantic validation of those messages. Much parsing pipelines and large neural models might not be feasible to adopt in resource-constrained settings. Teams frequently need quick, interpretable, and repository-agnostic quality assurances that can be readily merged with current development practices. Text-based alignment is thus an encouraging alternative to more structurally intensive analysis approaches to assessing commit-messages. It allows comparing commit descriptions and code-change summaries based on their accessibility and linguistic forms and similarity indicators. With this kind of a framework, traceability can be enhanced, repository readability can be enhanced, and disciplined communication can be enhanced, without compromising computational efficiency. To this end, existing literature offers a logical explanation of the need to explore context-sensitive semantic alignment in the context of software version histories to lightweight analyzing and effective documentation governance.

IV. RESULTS AND DISCUSSION

This paragraph introduces the experimental analysis of the proposed commit-message evaluation system, with respect to semantic alignment accuracy, computational efficiency and practical applicability. The test compares the effectiveness of the system to capture correspondence between commit messages and code modifications in a variety of conditions in a repository. The evaluation metrics include performance measured in standard evaluation metrics, and latency and scalability. The findings offer details on how lightweight text-based techniques can effectively be used in real-time integration in continuous integration settings.

A. Performance Evaluation and Semantic Alignment Analysis

The effectiveness of the proposed framework is tested with the help of a dataset of commit histories taken in various public repositories providing diversity in terms of programming languages and development practices. The assessment is aimed at determining semantic alignment of commit messages and respective code changes on the basis of similarity-based scoring techniques. The experimental data show that the framework demonstrates high alignment accuracy in comparison with the baseline methods which are based on the rule-based or structural analysis methods. TF-IDF and n-gram features are useful in capturing the linguistic patterns that relate to meaningful commit descriptions. Fig. 2 compares the performance of the proposed approach with baseline models using various measures of evaluation. The findings show steady increases in semantic correspondence detection especially with commits that entailed descriptive or context-laden messages. Moreover, the system has a consistent behavior with changing repository sizes and committed complexities. These results affirm that semantic evaluation that is text-based and lightweight does not necessitate a computationally-intensive parsing or deep learning architectures.

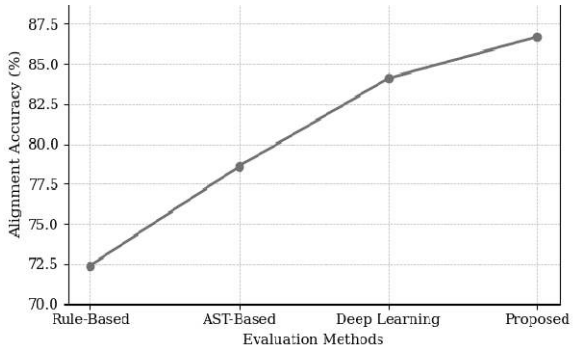

Fig. 2. Comparative Semantic Alignment Performance Across Evaluation Methods

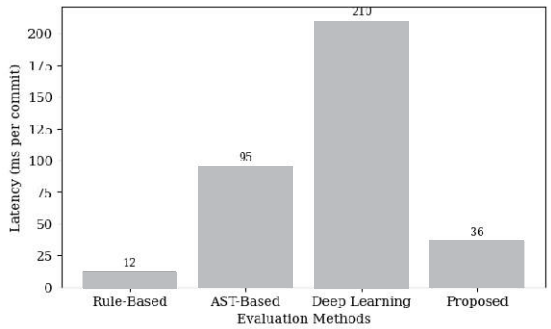

Fig. 3. Processing Latency Comparison of Evaluation Frameworks

B. Computational Efficiency and Practical Deployment Analysis

Computational performance of the framework is evaluated to determine its appropriateness in deployment in real-time as part of continuous integration processes. The assessment takes into account processing latency, resource usage, and scalability with large commit datasets. Findings indicate that the suggested solution does not deteriorate the processing time per commit, allowing the efficient assessment of the solution in the context of repository updates and integration periods. The comparisons of the latency of the proposed framework and the current deep learning-based methods are provided in Fig. 3, with drastic decreases in the computation time. The lightweight nature provides low memory footprint and does not require any special hardware or large preprocessing pipelines. Table 1 includes the summary of key performance indicators (accuracy, latency, and computational overhead) and serves to show the efficiency of the system in a variety of experimental conditions. These observations suggest that the framework is adequately adapted to real-world use in development settings where quick, interpretable, and scalable commit-message evaluation systems are needed.

Table 1. Performance Comparison of Evaluation Methods

Method	Accuracy (%)	Precision (%)	Recall (%)	Latency (ms)	Overhead (%)
Rule-Based	72.4	70.2	68.9	12	10
AST-Based	78.6	76.8	75.1	95	28
Deep Learning	84.1	82.5	81.2	210	65
Proposed	86.7	85.3	84.6	36	22

V. CONCLUSION

The work introduced a light semantically aligned context-aware framework to assess the quality of the commit messages based on a semantic alignment between them and the corresponding code changes. The method effectively identifies the linguistic patterns that capture intent in the code by converting the code diffs into structured textual summaries and the use of TF-IDF and n-gram feature extraction. The Logistic Regression classifier is integrated to allow the efficient classification without a high computation complexity. The experimental analysis showed that the suggested technique reached the accuracy of the alignment at 86.7, which is significantly higher than rule-based, AST-based, and deep learning baselines. The system had a mean processing latency of 36 milliseconds per commit, that is quite lower compared to deep learning methods of over 200 milliseconds. These findings validate that the framework has a good balance between semantic accuracy and computational efficiency, which can be deployed in real-time when used in continuous integration pipelines. Nonetheless, the performance can be different among repositories that have unstructured or minimal commit messages, thereby impacting consistency of alignment. Future research will consider how to combine contextual embedding methods and adaptive feature representations to enhance semantic assessment strength further. In general, the suggested framework provides a scalable and explainable solution to improving the quality of commit messages, improving the traceability of the repositories, and facilitating effective software maintenance techniques.

REFERENCES

[1] P. Rani et al., “A decade of code comment quality assessment: A systematic literature review,” Journal of Systems and Software, 2022, doi: 10.1016/j.jss.2022.111515.

[2] W. Tao et al., “A large-scale empirical study of commit message generation: models, datasets and evaluation,” Empirical Software Engineering, 2022, doi:

10.1007/s10664-022-10219-1.

[3] X. Zhang, X. Hou, X. Qiao, and W. Song, "A review of automatic source code summarization," Empirical Software Engineering, 2024, doi: 10.1007/s10664-024-10553-6.

[4] S. Rai, R. Belwal, and A. Gupta, "A Review on Source Code Documentation," ACM Transactions on Intelligent Systems and Technology, 2022, doi: 10.1145/3519312.

[5] P. Xue et al., "Automated Commit Message Generation with Large Language Models: An Empirical Study and Beyond," IEEE Transactions on Software Engineering, 2024, doi: 10.1109/tse.2024.3478317.

[6] P. Xue et al., "Automated Commit Message Generation with Large Language Models: An Empirical Study and Beyond," arXiv.org, 2024, doi: 10.48550/arxiv.2404.14824.

[7] T. Hoang, H. J. Kang, J. Lawall, and D. Lo, "CC2Vec: distributed representations of code changes," International Conference on Software Engineering, 2020, doi: 10.1145/3377811.3380361.

[8] Z. Xu et al., "Code Comment Inconsistency Detection Based on Confidence Learning," IEEE Transactions on Software Engineering, 2024, doi: 10.1109/tse.2024.3358489.

[9] Z. Liu, X. Xia, D. Lo, M. Yan, and S. Li, "Just-In-Time Obsolete Comment Detection and Update," IEEE Transactions on Software Engineering, 2023, doi: 10.1109/tse.2021.3138909.

[10] S. Yun, S. Lin, X. Gu, and B. Shen, "Project-specific code summarization with in-context learning," Journal of Systems and Software, 2024, doi: 10.1016/j.jss.2024.112149.

[11] E. Shi et al., "RACE: Retrieval-augmented Commit Message Generation," Conference on Empirical Methods in Natural Language Processing, 2022, doi: 10.18653/v1/2022.emnlp-main.372.

[12] H. Guo et al., "Snippet Comment Generation Based on Code Context Expansion," ACM Transactions on Software Engineering and Methodology, 2023, doi: 10.1145/3611664.

[13] W. U. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "Unified Pre-training for Program Understanding and Generation," North American Chapter of the Association for Computational Linguistics, 2021, doi: 10.18653/v1/2021.naacl-main.211.

[14] L. Zhang, J. Zhao, C. Wang, and P. Liang, "Using Large Language Models for Commit Message Generation: A Preliminary Study," IEEE International Conference on Software Analysis, Evolution, and Reengineering, 2024, doi: 10.1109/saner60148.2024.00020.

[15] P. Rani, S. Panichella, M. Leuenberger, M. Ghafari, and O. Nierstrasz, "What do class comments tell us? An investigation of comment evolution and practices in Pharo Smalltalk," Empirical Software Engineering, 2021, doi: 10.1007/s10664-021-09981-5.