Leveraging Artificial Intelligence for Automated Software Documentation Assessment: A Machine Learning Approach to Code Comment Evaluation

Aritra Maji[1,*]

[1]*Computer Science & Engineering Department, Indian Institute of Technology Kharagpur, West Bengal- 721302*

Abstract

This research investigates an innovative approach for improving binary classification systems designed to evaluate software documentation quality through the application of advanced AI methodologies. Our study demonstrates the effectiveness of augmenting traditional datasets with artificially generated training samples, specifically incorporating 1,400 synthesized code-comment pairs categorized as either "Useful" or "Not Useful" from diverse GitHub repositories into a foundational C programming language corpus containing 9,000 annotated instances. Through the utilization of sophisticated Large Language Model architectures, our experimental framework achieves notable performance enhancements: a 8.0% precision improvement in Support Vector Machine (SVM) implementations, advancing from 0.75 to 0.81, alongside a 3.02% recall enhancement in Artificial Neural Network (ANN) configurations, progressing from 0.694 to 0.715. These empirical findings validate the efficacy of AI-driven data augmentation strategies in refining automated comment evaluation systems, presenting considerable opportunities for advancing software engineering practices and development workflow optimization. The investigation establishes a foundation for incorporating synthetic data generation methodologies into machine learning applications within software analysis domains.

Keywords

Software Documentation Assessment, Machine Learning Data Augmentation, Support Vector Machines, Artificial Neural Networks, Natural Language Processing

1. Introduction

Software documentation through inline comments represents a fundamental component of modern programming practices, serving as a critical mechanism for code comprehension, collaborative development, and sustainable software maintenance workflows, as established by De et al. (2005) [1]. Nevertheless, the manual assessment of comment effectiveness presents significant challenges, primarily due to the resource-intensive nature and inherent subjectivity of human evaluation processes, as documented by Haouari et al. (2011) [2]. In response to these constraints, our investigation examines the application of advanced AI technologies for automating documentation quality evaluation, building upon the theoretical foundations proposed by Ebert et al. (2023) [3], thereby contributing to the enhancement of code review methodologies and the acceleration of software development workflows.

The strategic integration of high-quality documentation within software development lifecycles offers substantial advantages to development teams, including expedited debugging processes, comprehensive technical documentation, and the establishment of solid architectural foundations for future enhancement cycles, as demonstrated by Majumdar (2020) [4]. Our research presents a detailed analysis of the proposed methodology, experimental framework, and the potential paradigm shift that AI-driven approaches could introduce to software engineering practices, consistent with findings reported by Roehm et al. (2012) [5]. The subsequent sections examine current literature on automated comment evaluation and describe our novel approach to dataset augmentation through Large Language Model implementations.

Forum for Information Retrieval Evaluation, December 17-20, 2025, India

*Corresponding author.

aritra.maji.1729@gmail.com (A. Maji)
0009-0000-9218-2670 (A. Maji)

1.1. Related Work: Automated Documentation Analysis and Contemporary Challenges

The domain of automated software comprehension has established itself as a prominent research field within the software engineering community. Multiple frameworks and tools have been developed to extract meaningful insights from software artifacts, including execution traces and code structural characteristics [1, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]. Academic investigations have produced diverse methodologies for analyzing and evaluating inline documentation, with particular emphasis on quality assessment through comparative analysis of documentation-code relationships. In the context of documentation quality evaluation, researchers [18, 19, 20, 21, 22, 23, 24, 15] have implemented approaches including lexical similarity metrics (such as Levenshtein distance computations) and textual length analysis to identify and eliminate superficial or uninformative documentation. Rahman et al. [25] developed methods for distinguishing valuable from non-valuable code review feedback within collaborative development platforms, utilizing characteristics derived from empirical studies involving Microsoft development teams [26].

Novice developers frequently depend on existing documentation to understand program logic and execution flow. Nevertheless, not every piece of documentation provides meaningful assistance for code comprehension, thereby requiring systematic evaluation of source code documentation relevance before utilization. Multiple research endeavors have concentrated on automated categorization of source code documentation regarding quality assessment. For example, Omal et al. [27] observed that variables affecting software maintainability can be structured into hierarchical frameworks. These researchers established quantifiable characteristics through metric-based evaluations for individual factors, facilitating software property assessment that can subsequently be aggregated into unified maintainability indices. Fluri et al.[28] investigated the correlation between source code modifications and corresponding documentation updates across multiple software versions. Their analysis of three open-source projects, including *ArgoUML, Azureus*, and *JDT Core*, revealed that 97% of documentation modifications occur within identical revisions as their associated source code changes. Yu Hai et al.[29] developed a categorization system for source code documentation utilizing four quality levels - inadequate, acceptable, good, and exceptional. The combination of fundamental classification algorithms resulted in improved categorization performance. Additional research published in [30] introduced an automated classification framework called "CommentProbe" for evaluating documentation quality within C programming language codebases. While researchers have addressed source code documentation from multiple perspectives[30, 4, 22, 21, 24, 15], automated quality assessment of source code documentation remains an active area requiring continued investigation.

Following the emergence of sophisticated language models [31], comparative analysis of documentation quality assessment between established models such as GPT 3.5 or LLaMA and human evaluation becomes crucial. The IRSE track at FIRE 2024 [32, 14] extends the methodological frameworks established in [30, 16, 33, 21] to examine various vector space representations [34] and feature sets for binary classification and assessment of documentation in the context of code understanding. This research initiative also evaluates predictive model performance through the integration of AI-generated labels for quality assessment of code and documentation segments extracted from open-source software repositories.

Code comments are used to clarify logic, design decisions, and develop challenges [35]. However, manual evaluation remains inconsistent, time-consuming, and subjective [4]. Automated classification, labeling comments as "Useful" or "Not Useful," offers a more efficient approach to streamline code review [30]. This study examines how Generative AI can enhance these classification models [3], potentially transforming comment quality assessment. By prioritizing essential comments, resource management can improve. This introduction sets up a discussion on how Large Language Models (LLMs) are advancing code comment classification and software development practices [1].

1.2. IMPACT OF LLM ON THE QUALITY OF COMMENTS

Leveraging Large Language Models (LLMs) represents a major advancement in evaluating the quality of code comments [3]. These models move beyond syntactic comprehension, capturing the deeper semantics of the code and generating insightful comments that streamline assessment processes. By doing so, they significantly enhance the relevance and clarity of comments across the Software Development Life Cycle (SDLC). Beyond mere classification, LLMs redefine developer interaction with code, fostering clearer communication and strengthening collaboration. This transformative impact underscores the essential role LLMs are set to play in the future of code comment quality evaluation.

The application of Generative AI within the IRSE@FIRE-2024 task [33] is set to transform code quality evaluation, streamlining the Software Development Life Cycle (SDLC) and promoting more effective resource distribution and collaborative development efforts among teams.

The subsequent sections are organized as follows: Section 2 provides an overview of comment classification and the foundations of Generative AI. Section 3 describes the task setup and dataset used. Our methodology is detailed in Section 4. In Section 5, we present the results, while Section 6 offers a comparative analysis of our models and embeddings against established approaches in code comment quality assessment, underscoring their unique contributions. Lastly, Section 7 concludes with a summary of our findings and discusses possible avenues for future research.

2. Related Work

Automated program understanding is a recognized research area among professionals in the software domain. Various tools have been developed to facilitate the extraction of knowledge from software metadata, encompassing components such as runtime traces and structural attributes of code [1, 6, 7, 8, 9, 10, 11, 12]. Researchers have developed various methods to mine and evaluate code comments, focusing on analyzing comment quality through code-comment pair comparisons. In assessing code comment quality, authors [18, 19, 20, 21, 22, 23, 24, 15] employ techniques such as word similarity measures (e.g., Levenshtein distance) and comment length analysis to filter out trivial and non-informative comments. Rahman et al. [25] detect useful and non-useful code review comments (logged-in review portals) based on attributes identified from a survey conducted with developers of Microsoft [26].

New programmers often rely on existing comments to comprehend code flow. However, not all comments contribute effectively to program comprehension, necessitating a relevancy assessment of source code comments prior to their use. Numerous researchers have focused on the automatic classification of source code comments in terms of quality evaluation. For instance, Omal et al. [27] noted that factors influencing software maintainability can be organized into hierarchical structures. The authors defined measurable attributes in the form of metrics for each factor, enabling the assessment of software characteristics, which can then be consolidated into a single index of software maintainability. Fluri et al.[28] examined whether the source code and associated comments are changed together along the multiple versions. They investigated three open source systems, such as *ArgoUML, Azureus*, and *JDT Core*, and found that 97% of the comment changes are done in the same revision as the associated source code changes. Yu Hai et al.[29] classified source code comments into four classes - unqualified, qualified, good, and excellent. The aggregation of basic classification algorithms further improved the classification result. Another work published in [30] in which author proposed an automatic classification mechanism "CommentProbe" for quality evaluation of code comments of C codebases. We see that people worked on source code comments with different aspects[30, 4, 22, 21, 24, 15], but still, automatic quality evaluation of source code comments is an important area and demands more research.

With the advent of large language models [31], it is important to compare the quality assessment of code comments by the standard models like GPT 3.5 or llama with the human interpretation, like [36, 37]. The IRSE track at FIRE 2024 [32, 14] builds upon the methodologies proposed in [30, 16, 33, 21] to investigate various vector space models [34] and features for binary classification and evaluation of comments in relation to code comprehension. This track also assesses the performance of the predictive model by incorporating GPT-generated labels for the quality of code and comment snippets extracted from open-source software.

3. Experimental Framework and Data Specifications

This section describes the IRSE@FIRE-2024 research challenge [33], which centers on advancing binary classification systems for automated software documentation quality assessment. The experimental framework involves the strategic incorporation of synthetically generated documentation-code pairs to achieve improved classification performance. The foundation consists of an initial corpus containing 9000 annotated documentation-code pairs in the C programming language, with 5480 instances categorized as "Useful" and 3520 instances labeled "Not Useful", supplemented by additional pairs synthesized through Large Language Model (LLM) architectures, each receiving appropriate quality labels.

The expected deliverables encompass two distinct classification model variants: an enhanced version incorporating the augmented synthetic training data and a baseline version utilizing only the original dataset. The foundational corpus comprises 9000 documentation instances extracted from GitHub repositories, each containing the documentation text, associated code context, and corresponding utility classification (Table 1).

Table 1
Sample Data Instance

Comment	Code	Label

/* Swap two values */	void swapValues(int *x, int *y) {
	int temp;
	temp = *x; *x = *y;
	*y = temp;}
		Useful

/* Compute the sixth power */	int result = computeSixthPower(value);	Not Useful
/*Simple variable declaration */	int x = 10;	Not Useful

Ground truth establishment was achieved through independent evaluation by 12 expert annotators, achieving substantial inter-annotator agreement (Cohen's kappa coefficient of 0.746). The comprehensive annotation process encompassed the evaluation of 15,000 documentation instances.

Research participants were additionally required to construct a supplementary dataset comprising labeled documentation-code pairs sourced from GitHub repositories using LLM technologies. This augmented dataset constitutes a mandatory component of the experimental submission.

In essence, the primary objective involves refining automated documentation quality classification models through the strategic integration of synthetically generated training instances, thereby achieving enhanced predictive accuracy and system effectiveness.
For further details, please refer to the task description provided at IRSE@FIRE-2024 [1].

4. Research Methodology

Our experimental framework integrates multiple sophisticated approaches, encompassing Support Vector Machine (SVM) architectures for classification tasks and Artificial Neural Network (ANN) implementations featuring varied activation functions to model intricate data relationships [38]. Furthermore, we employ Large Language Model (LLM) capabilities through OpenAI API integration and leverage GitHub repository mining to construct a comprehensive and diverse corpus of documentation-code pairs. The subsequent sections elaborate on our specific methodological components: SVM model

[1]https://sites.google.com/view/irse2024/home

implementation, ANN architecture exploration, and dataset synthesis using OpenAI API and GitHub repository resources. These methodological elements collectively establish the foundation for our novel approach to automated software documentation quality evaluation. The architectural design of our methodology is illustrated in Figure 1, which provides a comprehensive visualization of our system's structural framework.

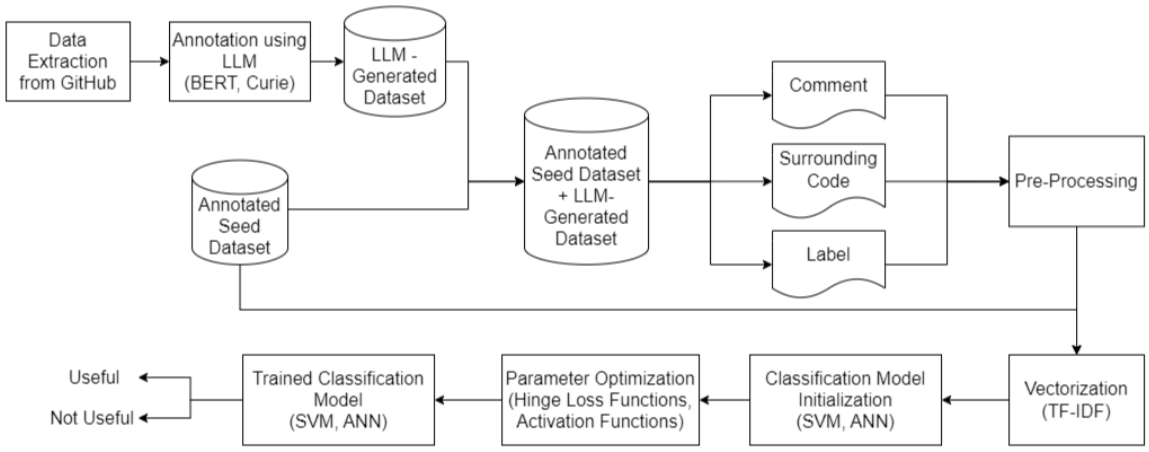
Data Extraction from GitHub
Annotation using LLM (BERT, Curie)
LLM - Generated Dataset
Annotated Seed Dataset
Annotated Seed Dataset + LLM- Generated Dataset
Comment
Surrounding Code
Label
Pre-Processing
Vectorization (TF-IDF)
Classification Model Initialization (SVM, ANN)
Parameter Optimization (Hinge Loss Functions, Activation Functions)
Trained Classification Model (SVM, ANN)
Useful
Not Useful

Figure 1: Architecture Diagram

4.1. Support Vector Machine Implementation

Linear Support Vector Machine (SVM) architectures represent sophisticated classification methodologies that identify optimal hyperplane configurations for effective data partitioning, mathematically expressed as $y = mx + b$, where y represents the predicted class designation, x denotes the input feature vector, m corresponds to the slope parameter, and b represents the y-intercept term. The algorithm maximizes the separation margin, defined as the distance between the hyperplane and the closest data instances. This margin (M) is computed using:

$$M = \frac{2}{||m||} \tag{1}$$

where $||m||$ represents the magnitude of the weight vector m.
SVM optimization seeks to minimize the squared magnitude of the weight vector ($||m||^2$) while maintaining correct classification for each data instance x_i:

$$y_i(m \cdot x_i + b) \geq 1 \tag{2}$$

Equation 2 establishes that the discriminant function output must exceed or equal unity for all training instances, highlighting the significance of robust class boundary definition in SVM implementations. This constraint constitutes the foundation of SVM's objective to identify optimal hyperplane positioning, maximizing inter-class margins while ensuring reliable data instance classification. Support vectors, representing data points nearest to the decision boundary, play crucial roles in margin determination, consequently affecting overall SVM classification performance.

4.2. Artificial Neural Network Architectures

Artificial Neural Network (ANN) frameworks constitute flexible machine learning paradigms inspired by biological neural system structures and computational processes. These models demonstrate exceptional capability in identifying intricate data patterns and relationships, rendering them particularly suitable for applications such as automated software documentation quality classification. The mathematical formulation of an individual neuron within an ANN architecture is expressed as:

$$Z = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b \tag{3}$$

where x_n represent input feature components, w_n denote corresponding weight parameters, and b signifies the bias coefficient.

The computed weighted summation (Z) undergoes transformation through an activation function, which incorporates non-linear characteristics into the model architecture. Various activation functions produce distinct learning dynamics and behavioral patterns.

The following represent commonly utilized activation functions and their mathematical expressions:

i) Sigmoid (Logistic) Function:

$$f(Z) = \frac{1}{1 + e^{-Z}} \tag{4}$$

ii) Rectified Linear Unit (ReLU):

$$f(Z) = \max(0, Z) \tag{5}$$

iii) Hyperbolic Tangent (tanh):

$$f(Z) = \frac{e^Z - e^{-Z}}{e^Z + e^{-Z}} \tag{6}$$

4.3. Large Language Model Integration for Dataset Synthesis

Our data generation methodology implements a comprehensive multi-phase approach to corpus construction. Initially, we utilized both OpenAI API services, leveraging the Curie Model architecture, and GitHub repository mining to enhance dataset diversity. The API integration enabled simulation of realistic programming contexts, generating authentic documentation-code pairs while substantially expanding our training corpus. This approach was complemented by extraction of additional pairs from diverse open-source software projects hosted on GitHub, ensuring contextual relevance and practical utility. This dual-strategy approach significantly expanded dataset coverage while maintaining rigorous quality standards. Subsequently, the synthesized documentation-code pairs underwent processing through OpenAI's Curie Model implementation combined with BERT architectures for automated label assignment, indicating documentation utility classifications. This process involved constructing structured prompts containing both code segments and associated documentation, then utilizing the LLM framework to generate appropriate quality labels. Finally, the comprehensive dataset was systematically compiled, with each instance containing code segments, corresponding documentation, and algorithmically generated quality classifications. This methodical approach establishes a solid foundation for our automated software documentation quality assessment framework.

5. Experimental Results and Performance Analysis

The assessment of our automated software documentation quality classification framework constitutes a fundamental component in establishing its practical effectiveness. We utilized a combination of Support Vector Machines (SVM) and Artificial Neural Networks (ANN) with various activation functions, including ReLU, identity, logistic, and tanh, to conduct a comprehensive analysis of the model's performance. This multidimensional approach offered valuable insights into the model's adaptability, revealing its robustness across diverse scenarios. Additionally, integrating these methodologies resulted in a significant improvement in precision, underscoring the model's ability to categorize code comments accurately based on practical value. These findings align with previous research that demonstrates the reliability of SVM and ANN models for comment quality assessment. The use of diverse activation functions further highlights the flexibility of our approach, reinforcing the model's potential applicability in real-world software development.

5.1. Model Performance Evaluation

The systematic assessment of our automated documentation quality classification models produced significant insights, demonstrating the substantial impact of incorporating LLM-synthesized data into our foundational corpus of 9000 instances. This initial dataset was thoughtfully partitioned into training, testing and validation sets, with the testing set comprising 1740 entries. With the Seed Data, SVM exhibited commendable precision (0.75), while ANN with ReLU activation demonstrated remarkable effectiveness, resulting in a notable recall score (0.715). Models with tanh and logistic activation functions showed similar precision scores of 0.6987 and 0.6935.

Post integration of 1400 LLM-generated entries, which seamlessly enriched the Seed Data, SVM's precision notably increased by 8.0%, elevating the preceding value to 0.81, highlighting the value of incorporating generative AI. Using ReLU, ANN achieved a noteworthy 3.02% rise in its recall, giving it a final recall of 0.7151, while tanh and logistic functions yielded marginal changes. Extensive experimentation with varied SVM models and ANN activation functions was performed, and the results depicts the effectiveness of our approach, emphasizing the importance of meticulous experimentation in fine-tuning models for code comment quality analysis.

Additionally, comprehensive numerical analysis is presented in Table 2, which offers detailed performance comparisons and classification reports for our optimal model configurations. This tabular presentation serves as an exhaustive reference for our experimental findings and facilitates comparative analysis of testing accuracies and F1-score metrics across baseline and augmented dataset configurations.

Table 2
Model Performance Comparison

Model	Performance with Seed Data		Performance with Integrated Data	
	Test Accuracy	F1-Score	Test Accuracy	F1-Score
Linear SVM	0.771	0.752	0.772	0.751
SVM (poly. kernel)	0.758	0.742	0.773	0.756
ANN (ReLU)	0.712	0.704	0.719	0.707
ANN (tanh)	0.715	0.693	0.712	0.706
ANN (logistic)	0.712	0.694	0.708	0.700
ANN (identity)	0.711	0.692	0.706	0.693

5.2. Synthetic Dataset Analysis and Integration Effects

The incorporation of synthetically generated training data through OpenAI's Large Language Model (LLM) architectures, combined with Curie model implementation and the utilization of heterogeneous datasets sourced from multiple GitHub repositories and open-source software projects, constitutes a substantial advancement in enhancing our automated documentation quality classification framework. Through the systematic addition of 1400 novel instances to our baseline corpus, we achieved significant enrichment in training data diversity. This augmentation of corpus heterogeneity resulted in notable accuracy improvements across our classification architecture, providing benefits to both Support Vector Machine (SVM) and Artificial Neural Network (ANN) implementations. The enhanced discriminative capacity achieved through this data fusion approach improves the system's generalization capabilities and predictive performance, validating the efficacy of integrating external synthetic data sources. Additionally, the combination of BERT embedding representations and Curie model architectures enabled our framework to effectively model the complexities of software documentation, substantially improving its capacity to differentiate between "Useful" and "Not Useful" documentation categories. This discriminative capability represents a critical advancement for practical deployment scenarios, where accurate documentation assessment significantly impacts software development workflow efficiency and maintenance process effectiveness.

6. Comparative Analysis and Research Implications

This section presents a comprehensive comparative evaluation of our modeling approaches and embedding strategies in the context of existing literature on automated software documentation classification. Our strategic focus on Support Vector Machine (SVM) and Artificial Neural Network (ANN) architectures, incorporating distinct activation function configurations, facilitates detailed examination of their effectiveness. This targeted investigation yields sophisticated insights into their capabilities for automated documentation quality assessment, providing contrast to the extensive classifier ensemble approaches employed by Majumdar et al. (2022a) [30].

Furthermore, our research approach differs substantially from the methodology presented by Majumdar et al. (2020) [4], which focuses primarily on knowledge domain extraction from software documentation to support developer inquiries during maintenance activities. Conversely, our investigation concentrates on the design and assessment of automated documentation quality classification systems. This encompasses the strategic integration of LLM-synthesized training data, yielding substantial improvements in classification accuracy.

Regarding embedding methodologies, Majumdar et al. (2022b) [34] concentrate on contextualized word representations adapted for software development textual content. Our approach implements both BERT-based representations and specialized embeddings designed specifically for software engineering domain concepts. This methodology delivers high-dimensional semantic feature representations suitable for diverse natural language processing applications. Notably, our labeling process utilizes the Curie model architecture. This methodological distinction emphasizes the adaptability and enhanced applicability of our embedding approach compared to the contextualized representations examined by Majumdar et al (2022b)[34].

Essentially, our research contribution concentrates on specialized model architectures and embedding methodologies, delivering distinctive insights into their effectiveness for automated software documentation quality evaluation. The focus on targeted models and domain-specific embeddings provides comprehensive insights into documentation quality assessment, differentiating our approach from the broader, context-oriented methodologies employed in previous investigations [34].

7. Conclusions and Future Directions

Based on these fundamental research contributions, our investigation demonstrates the practical viability and scalability of advanced AI technologies for real-world software engineering applications. Through the synthesis and strategic integration of artificial training data into established datasets, we have validated that sophisticated AI methodologies can substantially improve the performance of conventional models in automated software documentation quality evaluation. This methodological approach not only enhanced our systems' precision and recall metrics but also established the potential of AI-driven approaches to deliver comprehensive solutions for advancing software documentation practices, positioning these technologies as transformative tools for future development workflows.

The strategic incorporation of LLM-synthesized training data significantly enhanced model performance characteristics, with precision improvements of 8.0% for SVM architectures and recall enhancements of 3.02% for ANN implementations. These improvements elevated testing accuracies to 77.3% for SVM configurations and 71.9% for ANN systems, representing substantial progress from their pre-augmentation baseline performance levels. These measurable improvements validate the effectiveness of data augmentation through advanced AI methodologies, demonstrating how strategic dataset expansion can produce significant enhancements in model reliability and accuracy, particularly for sophisticated classification challenges within software engineering domains.

Considering future research directions, the implications of this investigation extend significantly beyond automated software documentation classification. The methodological frameworks introduced establish an adaptable foundation that can be extended to diverse applications within software development and quality assurance domains. Through the strategic utilization of advanced AI technologies, particularly Large Language Model (LLM) architectures, we demonstrate a powerful methodological approach that could fundamentally transform code analysis and documentation evaluation practices. As the software engineering industry continues evolving, this research provides compelling evidence of the substantial benefits achievable through advanced technology adoption, emphasizing the critical importance of innovative methodological solutions in improving efficiency and accuracy within practical engineering applications.

Declaration on Generative AI

In the preparation of this research work, the author utilized ChatGPT for grammatical refinement and linguistic enhancement. Following the application of these AI-assisted tools, the author conducted comprehensive review and revision of all content, maintaining complete responsibility for the academic integrity and accuracy of this publication.

References

[1] S. C. B. de Souza, N. Anquetil, K. M. de Oliveira, A study of the documentation essential to software maintenance, Conference on Design of communication, ACM, 2005, pp. 68–75.

[2] D. Haouari, H. Sahraoui, P. Langlais, How good is your comment? a study of comments in java programs, in: 2011 International symposium on empirical software engineering and measurement, IEEE, 2011, pp. 137–146.

[3] C. Ebert, P. Louridas, Generative ai for software practitioners, IEEE Software 40 (2023) 30–38.

[4] S. Majumdar, S. Papdeja, P. P. Das, S. K. Ghosh, Comment-mine—a semantic search approach to program comprehension from code comments, in: Advanced Computing and Systems for Security, Springer, 2020, pp. 29–42.

[5] T. Roehm, R. Tiarks, R. Koschke, W. Maalej, How do professional developers comprehend software?, in: 2012 34th International Conference on Software Engineering (ICSE), IEEE, 2012, pp. 255–265.

[6] S. Majumdar, S. Papdeja, P. P. Das, S. K. Ghosh, Smartkt: a search framework to assist program comprehension using smart knowledge transfer, in: 2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS), IEEE, 2019, pp. 97–108.

[7] N. Chatterjee, S. Majumdar, S. R. Sahoo, P. P. Das, Debugging multi-threaded applications using pin-augmented gdb (pgdb), in: International conference on software engineering research and practice (SERP). Springer, 2015, pp. 109–115.

[8] S. Majumdar, N. Chatterjee, S. R. Sahoo, P. P. Das, D-cube: tool for dynamic design discovery from multi-threaded applications using pin, in: 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS), IEEE, 2016, pp. 25–32.

[9] S. Majumdar, N. Chatterjee, P. P. Das, A. Chakrabarti, A mathematical framework for design discovery from multi-threaded applications using neural sequence solvers, Innovations in Systems and Software Engineering 17 (2021) 289–307.

[10] S. Majumdar, N. Chatterjee, P. Pratim Das, A. Chakrabarti, Dcube_ nn d cube nn: Tool for dynamic design discovery from multi-threaded applications using neural sequence models, Advanced Computing and Systems for Security: Volume 14 (2021) 75–92.

[11] J. Siegmund, N. Peitek, C. Parnin, S. Apel, J. Hofmeister, C. Kästner, A. Begel, A. Bethmann, A. Brechmann, Measuring neural efficiency of program comprehension, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, 2017, pp. 140–150.

[12] N. Chatterjee, S. Majumdar, P. P. Das, A. Chakrabarti, Parallelc-assist: Productivity accelerator suite based on dynamic instrumentation, IEEE Access 11 (2023) 73599–73612.

[13] S. Majumdar, A. Deshpande, P. P. Das, P. P. Chakrabarti, Comprehending c codes with llms: Effective comment generation through retrieval and reasoning, Pattern Recognition Letters (2025).

[14] S. Paul, S. Majumdar, R. Shah, S. Das, M. Ghosh, D. Ganguly, G. Calikli, D. Sanyal, P. P. Das, P. D. Clough, et al., Overview of the "information retrieval in software engineering"(irse) track at forum for information retrieval 2024, in: Proceedings of the 16th Annual Meeting of the Forum for Information Retrieval Evaluation, 2024, pp. 18–21.

[15] P. Chakraborty, S. Dutta, D. K. Sanyal, S. Majumdar, P. P. Das, Bringing order to chaos: Conceptualizing a personal research knowledge graph for scientists., IEEE Data Eng. Bull. 46 (2023) 43–56.

[16] S. Paul, S. Majumdar, A. Bandyopadhyay, B. Dave, S. Chattopadhyay, P. Das, P. D. Clough, P. Majumder, Efficiency of large language models to scale up ground truth: Overview of the irse track at forum for information retrieval 2023, in: Proceedings of the 15th Annual Meeting of the Forum for Information Retrieval Evaluation, 2023, pp. 16–18.

[17] N. Chatterjee, S. Majumdar, P. P. Das, A. Chakrabarti, Tool assisted agile approach for legacy application migration, International Journal of System Assurance Engineering and Management (2025) 1–16.

[18] L. Tan, D. Yuan, Y. Zhou, Hotcomments: how to make program comments more useful?, in: Conference on Programming language design and implementation (SIGPLAN), ACM, 2007, pp. 20–27.

[19] Y. Wang, H. Le, A. D. Gotmare, N. D. Bui, J. Li, S. C. Hoi, Codet5+: Open code large language models for code understanding and generation, arXiv preprint arXiv:2305.07922 (2023).

[20] D. Steidl, B. Hummel, E. Juergens, Quality analysis of source code comments, International Conference on Program Comprehension (ICPC), IEEE, 2013, pp. 83–92.

[21] S. Majumdar, A. Bandyopadhyay, P. P. Das, P. Clough, S. Chattopadhyay, P. Majumder, Can we predict useful comments in source codes?-analysis of findings from information retrieval in software engineering track@ fire 2022, in: Proceedings of the 14th Annual Meeting of the Forum for Information Retrieval Evaluation, 2022, pp. 15–17.

[22] S. Majumdar, A. Bandyopadhyay, S. Chattopadhyay, P. P. Das, P. D. Clough, P. Majumder, Overview of the irse track at fire 2022: Information retrieval in software engineering., in: FIRE (Working Notes), 2022, pp. 1–9.

[23] J. L. Freitas, D. da Cruz, P. R. Henriques, A comment analysis approach for program comprehension, Annual Software Engineering Workshop (SEW), IEEE, 2012, pp. 11–20.

[24] S. Majumdar, P. P. Das, Smart knowledge transfer using google-like search, arXiv preprint arXiv:2308.06653 (2023).

[25] M. M. Rahman, C. K. Roy, R. G. Kula, Predicting usefulness of code review comments using textual features and developer experience, International Conference on Mining Software Repositories (MSR), IEEE, 2017, pp. 215–226.

[26] A. Bosu, M. Greiler, C. Bird, Characteristics of useful code reviews: An empirical study at microsoft, Working Conference on Mining Software Repositories, IEEE, 2015, pp. 146–156.

[27] P. Oman, J. Hagemeister, Metrics for assessing a software system's maintainability, in: Proceedings Conference on Software Maintenance 1992, IEEE Computer Society, 1992, pp. 337–338.

[28] B. Fluri, M. Wursch, H. C. Gall, Do code and comments co-evolve? on the relation between source code and comment changes, in: 14th Working Conference on Reverse Engineering (WCRE 2007), IEEE, 2007, pp. 70–79.

[29] H. Yu, B. Li, P. Wang, D. Jia, Y. Wang, Source code comments quality assessment method based on aggregation of classification algorithms, Journal of Computer Applications 36 (2016) 3448.

[30] S. Majumdar, A. Bansal, P. P. Das, P. D. Clough, K. Datta, S. K. Ghosh, Automated evaluation of comments to aid software maintenance, Journal of Software: Evolution and Process 34 (2022) e2463.

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

[32] S. Paul, S. Majumdar, R. Shah, S. Das, M. Ghosh, D. Ganguly, G. Calikli, D. Sanyal, P. P. Das, P. D Clough, A. Bandyopadhyay, S. Chattopadhyay, Generative ai for code metadata quality assessment, in: Proceedings of the 16th Annual Meeting of the Forum for Information Retrieval Evaluation, 2024.

[33] S. Majumdar, S. Paul, D. Paul, A. Bandyopadhyay, S. Chattopadhyay, P. P. Das, P. D. Clough, P. Majumder, Generative ai for software metadata: Overview of the information retrieval in software engineering track at fire 2023, arXiv preprint arXiv:2311.03374 (2023).

[34] S. Majumdar, A. Varshney, P. P. Das, P. D. Clough, S. Chattopadhyay, An effective low-dimensional software code representation using bert and elmo, in: 2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS), IEEE, 2022, pp. 763–774.

[35] P. Rani, S. Panichella, M. Leuenberger, A. Di Sorbo, O. Nierstrasz, How to identify class comment types? a multi-language approach for class comment classification, Journal of systems and software 181 (2021) 111047.

[36] A. Deshpande, A. Maji, D. Mondol, P. P. Das, P. D. Clough, S. Majumdar, The code–llm handshake: Smarter maintenance through ai, in: Proceedings of the 17th annual meeting of the Forum for Information Retrieval Evaluation, 2025, pp. 9–12.

[37] A. Mitra, S. Majumdar, A. Mukhopadhyay, P. P. Das, P. D. Clough, P. P. Chakrabarti, Operationalizing large language models with design-aware contexts for code comment generation, arXiv preprint arXiv:2510.22338 (2025).

[38] L. Igual, S. Seguí, L. Igual, S. Seguí, Introduction to data science, Springer, 2017.