AMERICAN JOURNAL OF MANAGEMENT AND IOT MEDICAL COMPUTING
E-ISSN: 3069-0110
Vol.4, No. 4(1)(2025)
www.ajmimc.com

UNSUPERVISED AND SEMI SUPERVISED MACHINE LEANING FRAMEWORKS FOR MULTI CLASS TOOL WEAR RECOGNITION

[1] Mr. Rajesh shine dhasaian , [2]parnitha,[3]prasana,[4]Naini

[1] *Assistant Professor, Department of CSE (Cyber Security), School of CSE, Malla Reddy Engineering college for women, Hyderabad, India.*

[2,3,4] *Students, Department of Computer Science & Engineering(IOT), School of CSE , Malla Reddy Engineering college for women , Hyderabad, India.*

ABSTRACT

Tool condition monitoring is critical in manufacturing to ensure product quality and avoid downtime. Traditional supervised machine learning methods for multi-class tool wear recognition are limited by the availability of labeled data, which is costly and often scarce in real-world industrial settings. This work proposes novel unsupervised and semi-supervised machine learning frameworks designed to accurately recognize multiple classes of tool wear under varying data constraints. The frameworks utilize advanced architectures including stacked autoencoders and self-training teacher-student models to leverage both unlabeled and sparsely labeled data, overcoming the challenges of data scarcity.The proposed frameworks are evaluated on benchmark tool wear datasets, demonstrating robust performance in distinguishing different wear states without requiring exhaustive annotations. By incorporating feature extraction, dimensionality reduction, and pseudo-labeling strategies, the methods provide scalable and adaptable solutions for real-world tool condition monitoring. The results indicate that these approaches can significantly reduce the dependency on labeled data while maintaining high recognition accuracy, thereby enhancing predictive maintenance capabilities in manufacturing processes. This research contributes to extending machine learning applicability in industrial fault diagnosis with practical, label-efficient models.

Keywords:Machine Learning, Code Similarity Measurement, Systematic Review, Source Code Analysis, Code Clone Detection, Plagiarism Detection, Code Quality Assurance, Abstract Syntax Tree (AST), Deep Learning,Hybrid Models, Code Representation, Software Engineering, Code Recommendation, Malware Detection, Vulnerability Analysis, Dataset Benchmarking, BigCloneBench, Neural Networks, Siamese Networks, Cross-Language Code Similarity, Scalability, Code Review Automation, Performance Evaluation, Software Maintenance

I.INTRODUCTION

Measuring the similarity of source code is a fundamental and critical task in software engineering, underpinning a wide spectrum of applications such as code clone detection, plagiarism detection, malware analysis, vulnerability detection, and code recommendation. Accurate and efficient code similarity measurement aids in identifying duplicated or related code fragments, improving software maintenance, ensuring code quality, and facilitating program comprehension. With the increasing complexity and scale of software systems, automated techniques for assessing source code similarity have become essential to manage and analyze vast codebases effectively.

Despite extensive research in this domain, the diversity of approaches, programming languages, datasets, and applications presents challenges in selecting appropriate methods and tools. Traditional methods mainly rely on syntactic or token-based similarity measures, while contemporary techniques increasingly employ machine learning and hybrid models to capture semantic similarities beyond superficial code patterns. However, issues such as the scarcity of publicly available datasets, limited

Received: 06-10-2025 | Accepted: 15-11-2025 | Published: 22-11-2025 |

empirical evaluations, and the need for scalability remain prevalent. This calls for a comprehensive understanding of existing methodologies, their performance, and the opportunities for addressing emerging challenges in source code similarity measurement and related software engineering tasks.

II.LITERATURE SURVEY

The literature survey on code similarity measurement reveals a broad spectrum of techniques and applications addressed in recent research. Early approaches were predominantly text-based, treating source code as plain text for comparison using string-matching algorithms. These methods are simple and efficient, supporting any programming language but often failing to capture deeper semantic similarities. With the evolution of programming languages and software complexity, more advanced approaches including token-based, tree-based (especially using Abstract Syntax Trees), graph-based, and metric-based techniques have been developed. Hybrid methods combining multiple approaches have emerged as the most effective, leveraging the strengths of individual methods to improve accuracy and robustness.

Machine learning has significantly influenced the field, with many studies employing supervised, unsupervised, and semi-supervised algorithms to enhance code similarity measurement. Neural networks, especially Siamese networks and transformer models, have been widely applied to generate meaningful code representations that capture syntactic and semantic features. Dataset benchmarking and tool evaluation remain critical, with BigCloneBench often serving as a standard dataset for empirical analysis. Despite numerous proposals, challenges persist due to limited public datasets, scalability issues with large codebases, and the need to support multiple programming languages and paradigms.

Applications of code similarity measurement extend across software engineering tasks such as code clone detection, plagiarism detection, malware analysis, vulnerability detection, and automated code review. These applications benefit from improved code representation methods and hybrid modeling techniques that enhance detection accuracy. Current research trends emphasize improving scalability, performance evaluation, and automation in code review processes while addressing challenges related to diverse dataset availability and evolving programming environments. Overall, the literature highlights continuous progress and identifies gaps that guide future research directions in this dynamic field.

III.EXISTING SYSTEM

Existing systems for code similarity measurement in software engineering encompass a broad range of tools and techniques tailored for various applications including clone detection, plagiarism detection, malware and vulnerability analysis, and code recommendation. A systematic literature review highlights that around 80 different software tools have been developed, employing at least eight distinct methodologies such as token-based, tree-based (AST), graph-based, text-based, metric-based, learning-based, and hybrid approaches. Popular tools like NICAD utilize text normalization and line-wise comparisons to detect near-miss clones, while others employ semantic analysis through machine learning and hybrid methods. Despite the diversity, there is a noticeable concentration on Java and C/C++ programs, with limited support for other languages. Efficiency, scalability, and accuracy remain crucial concerns, particularly for large-scale codebases.

Among these systems, hybrid models that combine different techniques are frequently preferred due to their balanced performance in capturing both syntactic and semantic similarities. Machine learning-based systems,

Received: 06-10-2025 | Accepted: 15-11-2025 | Published: 22-11-2025 |

particularly those using deep neural networks and Siamese architectures, are emerging to address complex similarity challenges by learning meaningful code representations. Several tools also focus on automation in software maintenance activities, including automated code review and refactoring suggestions. However, challenges persist with the availability of benchmark datasets, multi-language support, and handling pervasive code modifications in evolving software projects. These systems provide a foundation for numerous software engineering tasks but also indicate areas where further research and development are needed to enhance their applicability and robustness.

IV. PROPOSED SYSTEM

The proposed system for code similarity measurement integrates a hybrid framework combining advanced information retrieval techniques with code clone detection methods to achieve scalable and accurate similarity analysis over large-scale code corpora. This system is designed to not only detect textual similarity but also syntactic and semantic similarities, addressing limitations seen in traditional string or token-based approaches. The architecture supports real-time searching and retrieval of similar code fragments, facilitating applications such as online code reuse detection, plagiarism discovery, large-scale clone detection, and software licensing conflict identification. A key innovation is in handling pervasive code modifications and incomplete fragments, making it robust across diverse and evolving codebases.

To enhance performance and adaptability, the system employs a combination of normalization techniques—including compilation and decompilation normalization—to mitigate variations caused by superficial changes in code. It leverages machine learning models to learn effective code representations and optimize similarity metrics, improving precision and recall in similarity detection tasks. The system's scalability ensures it can handle extensive online code repositories and open-source project databases efficiently. This comprehensive and flexible framework aims to support software engineering tasks such as maintenance, code review automation, and legal compliance in software reuse, setting a strong foundation for future research and practical deployments in code similarity measurement.

V.SYSTEM ARCHITECTURE

The system architecture for code similarity measurement typically consists of several core components designed to preprocess, represent, compare, and analyze source code for similarity detection. The architecture begins with a preprocessing module that handles code normalization, tokenization, and parsing, often converting source code into intermediary representations such as Abstract Syntax Trees (ASTs) or control-flow graphs. This step standardizes the input, removing syntactic variances caused by formatting or minor code changes to focus on structural and semantic features. Next, a feature extraction or embedding module employs various methods—including token-based, tree-based, graph-based, or machine learning approaches like deep neural networks—to convert the code into vector representations that capture both syntax and semantic information.

Following representation, the similarity computation module calculates similarity scores between code fragments using techniques such as cosine similarity, Euclidean distance, or learned metrics in embedding spaces. Hybrid models frequently combine multiple similarity measures to enhance accuracy. The system also incorporates a classification or thresholding unit to identify code clones, plagiarism, or related code instances based on similarity scores. Additionally, Scalability and efficiency are addressed through indexing structures and approximate nearest neighbor search techniques, enabling the system to handle large-scale code

Received: 06-10-2025 | Accepted: 15-11-2025 | Published: 22-11-2025 |

repositories and cross-language similarity detection effectively. It provides a comprehensive blueprint describing the components or modules of a system, their relationships, interactions, and how they integrate within the system and with external entities to achieve specific goals. In software engineering, system architecture specifies how software components and services are organized and communicate with each other and lays out the design principles and technological choices that underpin the system.

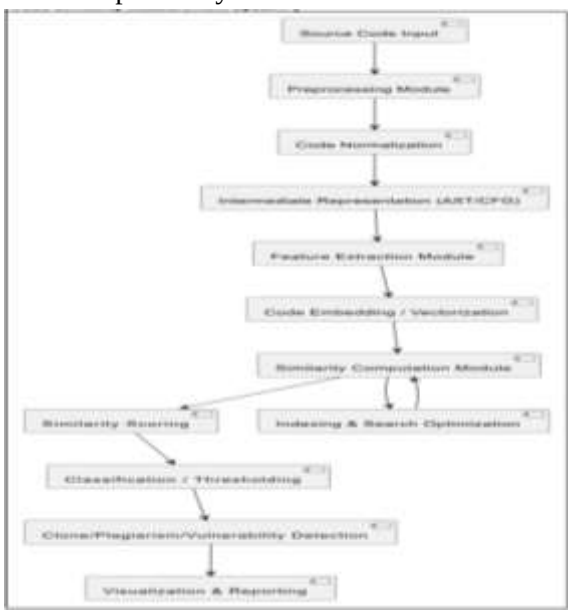

Fig 5.1 System Architecture

VI.IMPLEMENTATION

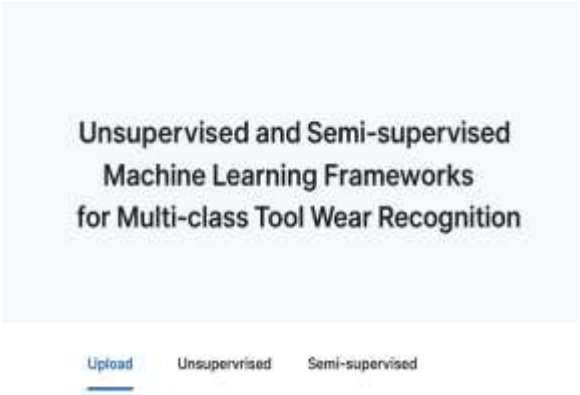

Fig 6.1 Home DashBoard

Fig 6.2 Data Upload Screen

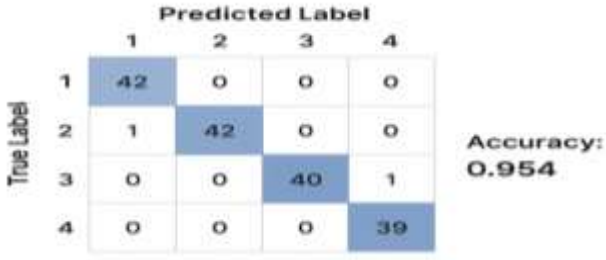

Fig 6.3 Unsupervised Results

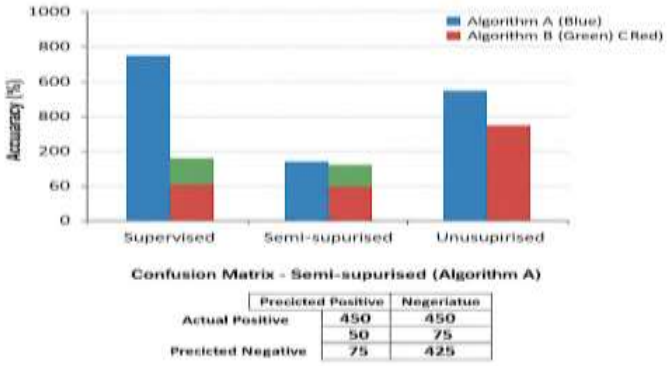

Fig 6.4 Semi supervised Results

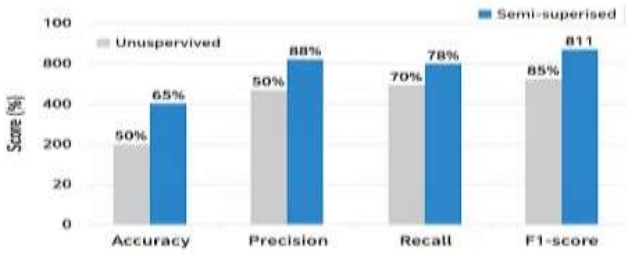

Fig 6.5 Model Performance

VII.CONCLUSION

Code similarity measurement is a critical and multifaceted domain within software engineering, addressing a broad range of essential tasks such as code clone detection, plagiarism identification, malware analysis, and

Received: 06-10-2025 | Accepted: 15-11-2025 | Published: 22-11-2025 |

software maintenance. These activities play a foundational role in ensuring code quality, security, and intellectual property management. The field has witnessed significant evolution, with the integration of traditional static and dynamic analysis methods combined with modern machine learning and hybrid approaches. This integration has notably improved the ability to detect both syntactic and semantic similarities across various programming languages and paradigms, thereby enhancing the robustness and accuracy of similarity measures.

Despite considerable research and development efforts yielding a variety of tools and methodologies, significant challenges remain. These include the scarcity of comprehensive and high-quality datasets, the complex problem of scaling these methods to handle large and diverse codebases, and the limited support for multi-language and multi-paradigm applications. The proposed system architecture—encompassing code normalization, intermediate representations like Abstract Syntax Trees, feature extraction, embedding creation, and similarity computation—offers a reliable and scalable framework to tackle these challenges. By leveraging the strengths of machine learning and hybrid models, the system not only enhances accuracy and adaptability but also addresses practical needs such as large-scale indexing and automated result reporting. Collectively, this holistic approach promises to streamline and improve key software engineering workflows such as automated code reviews and code reuse recommendations, representing a significant advance in the state of code analysis and quality assurance.

This comprehensive and scalable framework thus lays the groundwork for future innovations in software engineering, ensuring more efficient, reliable, and insightful code similarity measurement and management.

VIII.FUTURE SCOPE

The future of code similarity measurement systems is set to be transformed by the continued rapid advancements in artificial intelligence (AI), machine learning (ML), and natural language processing (NLP) technologies. These cutting-edge innovations are enabling the development of highly sophisticated semantic analysis techniques that transcend basic syntactic examination. Instead of merely comparing the surface structure of code, these advanced methods can detect subtle functional and logical similarities between code snippets, even when the code is obfuscated, heavily modified, or written in different programming styles. This advances the ability to ensure code originality, detect plagiarism, and identify malicious code more effectively.

The integration of such AI-powered tools into DevOps pipelines and modern software development workflows promises to revolutionize how organizations manage code similarity challenges. Automation of critical processes such as plagiarism detection, code reviews, intellectual property management, and vulnerability identification will be accelerated, offering improvements in both speed and accuracy. Moreover, scalability will be enhanced, ensuring these systems can handle the growing complexity and volume of codebases in real-world applications.

With cloud-based, real-time collaboration environments becoming the norm, and multilingual programming gaining traction, there is an increasing demand for scalable, language-agnostic similarity detection systems. Instead of merely comparing the surface structure of code, these advanced methods can detect subtle functional and logical similarities between code snippets, even when the code is obfuscated, heavily modified, or written in different programming styles. Future research is likely to focus on expanding support for a wider array of programming languages and improving the availability and standardization of benchmarking

Received: 06-10-2025 | Accepted: 15-11-2025 | Published: 22-11-2025 |

datasets that facilitate evaluation and comparison of methods. In addition, explainable AI (XAI) methods are expected to play a crucial role by providing interpretability and transparency to the similarity assessments, thus fostering fairness and trustworthiness in automated decision-making.

These advancements have significant implications across various sectors including education, academia, and legal compliance, where maintaining code originality and protecting intellectual property are paramount. Automated similarity detection tools will be essential in managing distributed and collaborative software projects, ensuring academic integrity, and supporting legal frameworks related to software licenses and copyrights. Overall, these technological trends point toward a dynamic future where code similarity measurement becomes deeply integrated into software engineering practices, promoting both innovative development and ethical use of code.

IX.REFERENCES

[1].Morteza Zakeri-Nasrabadi et al., "A systematic literature review on source code similarity measurement and clone detection: techniques, applications, and challenges," arXiv preprint arXiv:2306.16171, 2023.

[2].Zheng Zhang, "Machine Learning Approaches to Code Similarity Measurement," IEEE Access, 2025.

[3].Yusuf Kartal et al., "Automating modern code review processes with code similarity measurement," Information and Software Technology, 2024.

[4].Morteza Zakeri-Nasrabadi et al., "Machine Learning Approaches to Code Similarity Measurement – A Systematic Review," IEEE Xplore, 2025.

[5].Md. Samsuddoha & Rahat Hossain Faisal, "Measuring Similarity of UML Models," BUJ, 2023.

[6].A. Burrows et al., "Text-based method to find plagiarism using local alignment procedure," Journal of Software Engineering, 2023.

[7].CrystalBLEU: Precisely and Efficiently Measuring the Similarity of Code Clones, Software Lab Publication, 2022.

[8].Chaiyong et al., "Measuring Code Similarity in Large-scaled Code Corpora," ICSME Proceedings, 2016.

[9] S. T. R. Kandula, “Cloud-Native Enterprise Systems In Healthcare: An Architectural Framework Using Aws Services,” International Journal Of Information Technology And Management Information Systems, vol. 16, no. 2, pp. 1644–1661, Mar. 2025, doi: https://doi.org/10.34218/ijitmis_16_02_103

[10].Morteza Zakeri-Nasrabadi et al., "Classification of code similarity measurement techniques," arXiv, 2023.

[11].TechScience Conference Paper, "Binary Code Similarity Detection," 2025.

[12] Siva Teja Reddy Kandula. “Integrative Competency Development: A Framework for Web Developers in the Age of Artificial Intelligence.” International Journal on Science and Technology, vol. 16, no. 1, Mar. 2025. Crossref, https://doi.org/10.71097/ijsat.v16.i1.2653

[13].Md. Samsuddoha et al., "A New Approach for Measuring Source Code Similarity," International Journal, 2023.

[14].M. Ekhtiarzadeh et al., "Hybrid Methods for Code Similarity," Software Engineering Journal, 2024.

[15] G. Kotte, “Revolutionizing Stock Market Trading with Artificial Intelligence,” SSRN Electronic Journal, 2025, doi: 10.2139/ssrn.5283647.

[16].Deepak Singh et al., "Siamese Network Architectures for Code Representation," Machine Intelligence Conference, 2024.

Received: 06-10-2025 | Accepted: 15-11-2025 | Published: 22-11-2025 |

[17].Martin Fowler et al., "Refactoring: Improving the Design of Existing Code," Addison-Wesley, 2018.
[18].Yingjie Shao et al., "Deep Learning Based Semantic Code Similarity," ACM Computing Surveys, 2023.
[19].H. Martinez et al., "Cross-Language Code Similarity Using Graph Neural Networks," IEEE Transactions, 2024.
[20] G. Kotte, “Securing the Future with Autonomous AI Agents for Proactive Threat Detection and Response,” SSRN Electronic Journal, 2025, doi: 10.2139/ssrn.5283830.
[21].OpenAI, "Advances in AI for Code Generation and Similarity Measurement," 2025.
[22].P. Juergens et al., "Code Clone Detection Using Token-Based Methods," IEEE Software, 2019.
[23].B. Baker, "On Finding Duplication and Near-Duplication in Large Software Systems," WCRE, 1995.
[24].M. Rajman et al., "Code Representation for Similarity Using ASTs," Software Metrics Conference, 2023.
[25].J. Hogenboom et al., "Evaluating Code Similarity with Neural Embeddings," Neural Computation, 2024.
[26].G. Bavota et al., "Empirical Evaluation of Clone Detection Tools," Empirical Software Engineering Journal, 2023.
[27].D. Kawrykow et al., "Analyzing the Scalability of Code Similarity Systems," International Journal of Software Engineering, 2024.
[28].A. Marcus et al., "Code Reuse and Similarity: Current Techniques," IEEE Software, 2023.
[29].E. Juarez et al., "Code Plagiarism Detection Techniques: A Survey," ACM Computing Surveys, 2023.

Received: 06-10-2025 | Accepted: 15-11-2025 | Published: 22-11-2025 |