A Simplified Spiking Neural Network for Developer Experience Classification Using Software Engineering Metrics

Bharat Babaso Mane
Department of Computer Science and Engineering, Alliance University, Bengaluru, Karnataka, India
bharat.mane@gmail.com (corresponding author)

Rathnakar Achary
Department of Computer Science and Engineering, Alliance University, Bengaluru, Karnataka, India
rathnakar.achary@alliance.edu.in

Received: 3 April 2026 | Revised: 20 April 2026, 1 May 2026, and 25 May 2026 | Accepted: 9 June 2026

 DOI: https://doi.org/10.48084/etasr.19085

ABSTRACT

In modern enterprises, software development is an important part, as businesses are increasingly relying on it to support and improve operational performance. Nevertheless, low-quality software decreases the level of satisfaction of clients. Software development is now considered a combined and technology-dependent activity, and the developer's experience can play an essential role in shaping the quality of software they generate. In software development, the allocation of the appropriate software developers (for example, those who have appropriate coding skills) to a project is an important feature. The difficulty lies in the fact that it is very complicated for project managers, clients, and software development companies to find a suitable developer to allocate a specific project. Therefore, there is a need for a scalable mechanism to identify the level of coding expertise of the software developer. Deep Learning (DL) methods have been extensively applied to assess the impact of developers' experience on code quality. This study presents an Empirical Evaluation of Developer Experience-Based Software Quality Estimation Using Spiking Neural Networks and Metaheuristic Optimization (EESQA-DELMOA) model, which aims to assess software quality by analyzing the developers' experience levels on code reliability and maintainability. EESQA-DELMOA employs a Bio-inspired Artificial Hummingbird (BAHB) technique to select the most relevant features for improving model performance. Subsequently, a Simplified Spiking Neural Network (SSNN) is deployed for classification. Finally, parameter tuning is performed using an Adaptive Migration Butterfly Optimization Algorithm (AMBOA) to improve the classification performance of the SSNN classifier. EESQA-DELMOA was experimentally evaluated using a benchmark dataset, and the results demonstrate its enhanced performance compared to recent approaches.

Keywords-software quality assessment; developers experience level; adaptive migration butterfly optimization algorithm; feature selection; deep learning

I. INTRODUCTION

Today, the field of software development is emerging rapidly, as software solutions are everywhere and play a vital role in all aspects of life, including healthcare, education, business, transport, industry, and finance [1]. However, software quality assessment is a complex process. Some factors that influence software quality and productivity include the coding knowledge and technical expertise of developers [2], which can result in fewer rework, coding errors, and bugs [3]. This theory is straightforward and has led many studies to explore the association between developers' experience and code quality. However, experimental studies in this domain frequently produce contradictory outcomes, underlining the difficulties of measuring and defining both knowledge and code quality in software development.

In the software development domain, the experience of the developer plays a major role in determining the overall quality of the developed software. Experienced developers are commonly experts or well-known for writing optimized, maintainable, and error-free code because of their deeper understanding of design principles, debugging techniques, and coding practices. This often results in reduced defect rates, enhanced code readability, and better system reliability. Certain studies have focused on identifying the methods and cognitive skills of developers, which is an intricate task [4, 5]. Recently, Deep Learning (DL) approaches have been investigated to determine the skill levels of developers by methodically examining their source code. DL methods and implementations offer several advantages that justify their application in many critical areas.

The study in [6] utilized pull requests, comments, and details to suggest the best developer based on their active experience. In [7], the effect of AI-developed code on software quality was examined, focusing on readability, code complexity, quality, and documentation. In [8], the perceptions, challenges, and practices of developers in the Internet of Things (IoT) landscape were investigated. In [9], the Software Quality Assurance as a Service (SQAaaS) model was introduced. In [10], the application of AI-based software testing was investigated, with a particular focus on the impact of LLMs on classical testing methods. Software testing is an important phase in software development that ensures the high-quality delivery of secure and reliable software systems. In [11], the focus was on the challenges of assigning an appropriate developer to a new bug, using ML-driven system recommendations based on developer profiles. In [12], a Software Quality Assurance (SQA) paradigm was presented, focusing on the telecommunications industry. This study recognized perceptions and aspects and relied on a complete work analysis and specialists' input through the Hierarchical Decision Modeling (HDM) method. In [13], an architecture for software quality testing focused mainly on automating software defect classification for eight specific defects. In [14], ML, NLP, and other modern methods were utilized to improve the invention of test cases and coverage and adapt to emerging software environments. The study in [15] demonstrated an approach to how traditional software engineering concepts, such as versioning, testing, Continuous Integration and Delivery (CI/CD), and lifecycle management, can be used in AI systems, particularly for ML Operations (MLOps).

Existing software quality assessment models mostly depend on traditional ML approaches with limited ability to acquire complex, nonlinear relations in developer-centric data, and frequently lack an incorporated pipeline that integrates efficient preprocessing, intelligent feature selection, and adaptive model optimization. To overcome these research gaps, this paper presents an Empirical Evaluation of Developer Experience-Based Software Quality Estimation Using Spiking Neural Networks and Metaheuristic Optimization (EESQA-DELMOA) model, with the following key contributions:

- Employs the Bio-inspired Artificial Hummingbird (BAHB) technique as a feature selection process.
- Uses a Simplified Spiking Neural Network (SSNN) for classification.
- Implements parameter optimization through the Adaptive Migration Butterfly Optimization Algorithm (AMBOA), improving classification performance.
- The experimental evaluation of the proposed EESQA-DELMOA model on a benchmark dataset exhibits improved performance.

II. PROPOSED METHOD

This EESQA-DELMOA framework aims to assess software quality by analyzing the levels of developer experience in code reliability and maintainability. Figure 1 presents the workflow of the EESQA-DELMOA model, which involves four major steps:

- Step 1: Preprocess the input dataset by employing min-max normalization.
- Step 2: Perform BAHB-based dimensionality reduction to identify the most related features and eliminate redundant ones, reducing noise and computation cost.
- Step 3: Feed the selected feature subset into the SSNN, which learns complex and subtle patterns in the data to perform accurate classification.
- Step 4: Use AMBOA to optimize the SSNN hyperparameters. AMBOA repeatedly searches for optimal hyperparameter values by validating SSNN performance on the chosen features, thus ensuring effective interaction between feature selection and model optimization.

Compared to simpler techniques that rely on raw features and fixed parameters, this integrated approach achieves better generalization, higher accuracy, and more reliable performance, especially in complex and high-dimensional datasets.

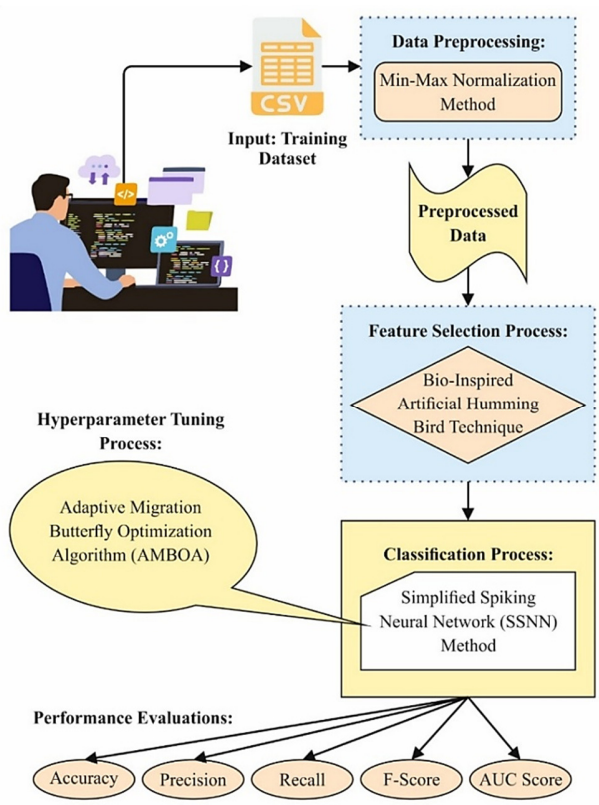

Fig. 1. Workflow of the EESQA-DELMOA model.

A. Data Preprocessing

In this stage, min-max normalization is utilized for cleaning and scaling the raw data into a common range, usually [0,1]. This is important in software quality assessment because developer-related features, such as experience level, number of past projects, and contribution frequency, frequently come in diverse scales. By standardizing them, each feature contributes

fairly during analysis, and no single variable dominates due to its magnitude. This also makes the data more fit for statistical and ML approaches, resulting in more consistent and reliable quality evaluation.

B. Dimensionality Reduction Using BAHB

The proposed EESQA-DELMOA method utilizes the BAHB optimization method to choose the most relevant attributes [16]. BAHB is inspired by the foraging behavior of hummingbirds and is developed to balance exploration and exploitation while searching for optimal feature subsets. Compared to traditional approaches such as genetic algorithms, BAHB performs more proficiently in managing complex and high-dimensional feature spaces. It leverages mechanisms like directed foraging and territorial behavior for improving search quality.

1) Initialization

A flock of k hummingbirds is located at a food resource, with their first locations being random.

$$p_n = l + r(u - l) n = 1, \ldots . k \quad (1)$$

where l and u symbolize low and upper boundaries, p_n specifies the position of k food resources, and r is a vector in the range from zero to one. The population of hummingbirds is deliberated as a feature.

$$TV_{n,m} = \begin{cases} 0 & n \neq m \\ null & n = m \end{cases} n = 1, \ldots k; m = 1, \ldots . . k \quad (2)$$

where $TV_{n,m} = null$ represents a hummingbird that has been nourished at its allocated source of food. For example, where $n \neq m$, $TV_{n,m} = 0$ is the n -th hummingbird that visited recently the m-th food resource in the existing iteration.

2) Guided Food Search

In hunting for meals, AHA effectively integrates three kinds of flight—diagonal, axial, and omnidirectional—by presenting a vector direction that functions to regulate the accessibility of more than one direction in the d-dimensional space.

$$A^n = \begin{cases} 1 & if n = ran(1, d) \\ 0 & else \end{cases} \quad n = 1, \ldots . d \quad (3)$$

Diagonal flight:

$$A^n = \begin{cases} 1 \; if \; n = s(m), m \in [1, x] s = ranpermutation(h), h \in [2, \lceil r(d-2) \rceil + 1] \\ 0 \; else \end{cases}$$

$$n = 1, \ldots . d \quad (4)$$

$$A^n = 1 \quad n = 1, \ldots . d \quad (5)$$

The function $ran(1, d)$ creates a uniformly distributed, pseudo-random integer among $[1, d]$, $ranpermutation(h)$ produces an arbitrary arrangement of integers in the interval $[1 - h]$, and r indicates a decimal fraction between zero and one.

3) Food Searching in a Particular Territory

The mathematical equation for the local searching behavior of humming-birds in searching for a particular territory toward a possible food resource is obtained using:

$$e_n(te + 1) = y_{n,aim}(te) + ter. A. (y_{n,aim}) \quad (6)$$

$$ter \sim ND(0,1) \quad (7)$$

where ter specifies a territorial factor.

4) The Process of Migration in Searching for Food

In migration, the hummingbird searching for food moves toward the resource with the lower rate of nectar-refilling, arbitrarily picking one.

$$p_{low} = l + r(u - l) \quad (8)$$

where p_{low} is a food resource with the minimum favorable rate of nectar-refilling with the population. BAHB selects optimum features effectively. The proposed fitness function balances the minimization of feature size with the maximization of classification accuracy. The evaluation process is described by:

$$Fitness = \alpha \gamma_R(D) + \beta \frac{|R|}{|C|} \quad (9)$$

where $\gamma_R(D)$ denotes classification error rates, $|R|$ signifies the cardinality of the chosen subset, and $|C|$ refers to the complete number of attributes in the data set, and α and β represent dual parameters related to the significance.

C. SSNN-Based Classification

An SSNN is applied for classification. An SSNN is a simplified form of SNN that utilizes basic neuron methods to reduce computational complexity while maintaining the main characteristics of spike-based information handling. It is more computationally effective than ANNs and may process temporal data. Conventional models fail to handle spike-based inputs, where each incoming spike continuously updates the neuron's potential based on its synaptic weight.

$$D_h = \begin{cases} D_{h-1} + \sum_{j=1}^{m} V_j \, C_{jh} - B, & if \; D_{\min} D_{h-1} D_{threshold} \\ D_{refract}, if \; D_{h-1} \geq D_{threshold} \\ G_D \; if \; D_{h-1} \leq D_{\min} \end{cases} \quad (10)$$

A spiker responding to a primary postsynaptic possible ramp is substituted by the immediate membrane-potential move, which allows neurons to fire immediately when the spike takes place in the succeeding time interval.

D. Parameter Tuning Using AMBOA

AMBOA is utilized to optimize the parameters of the KNN classifier, leading to improved predictive performance. AMBOA is motivated by butterfly mating and foraging behaviors on possible direction of food resources for mating partners, examining the intensity of airborne scent. Butterfly scent intensity is based on three factors: stimulus intensity, power index, and perceived morphology.

$$f = cI^a \quad (11)$$

where I represents stimulus intensity, f signifies the observed size of scent, c specifies the sensory modality, and a represents the power exponent condition. This model has dual key stages, which are local and global searching. The stages of local and global searching comprise the position vectors of butterflies.

$$x_i^{t+1} = X_i^t + (r^2 \times g^* - x_i^t) \times f_i \quad (12)$$

$$x_i^{t+1} = X_i^t + \left(r^2 \times x_j^t - x_k^t\right) \times f_i \quad (13)$$

where x_i^t denotes the location vector of the i-th butterfly at the t-th iteration, f_i refers to the fragrance strength of the i-th butterfly, g^* indicates the optimum solution of every solution in the existing iteration, r represents an estimated value in $[0,1]$, x_i^f, x_j^f, and x_k^f are the respective vector positions of the i-th, j-th, and k-th iterations. This algorithm is prone to falling into local optima and has a slow convergence speed. The migration factor can rapidly upgrade the location of the butterfly. BOA [17] has been improved and optimized utilizing the beginning of PSO.

III. EXPERIMENTAL VALIDATION

The model was developed on a Python 3.6.5 environment, powered by an Intel Core i5-8600k processor, 16 GB of RAM, and a 4 GB graphics card. Storage was handled by a 250 GB solid-state drive and a 1 TB hard drive. Training was performed using a learning rate of 0.01, the ReLU activation function, 50 epochs, a 0.5 dropout rate, and a batch size of 5. The proposed EESQA-DELMOA model was experimentally evaluated using the open-source dataset in [18]. This dataset has 703 samples with 23 developer-centric features related to software quality prediction, measuring developer activity and project context. This dataset was selected because of its relevance to developer-centric analysis and its suitability for evaluating intelligent feature selection and classification models, as represented below in Table I. From the total 26 features, only 18 were selected.

TABLE I. DETAILS OF DATASET

Class labels	No. of samples
ESE	69
SA	29
SE	73
NSE	17
BOT	10
UNK	505
Total Instances	703

Figure 2 presents a correlation matrix of all attributes in the dataset, showing the robustness and direction of relation between variables. These metrics fall on a scale from −1 to 1, with coefficients close to 1 indicating a strong direct relationship, those close to -1 indicating a strong inverse relationship, and those hovering around 0 suggesting little to no association. It can be observed that the features AB, NAB, CNII, and NCE have strong positive correlations with each other, suggesting similar behavior.

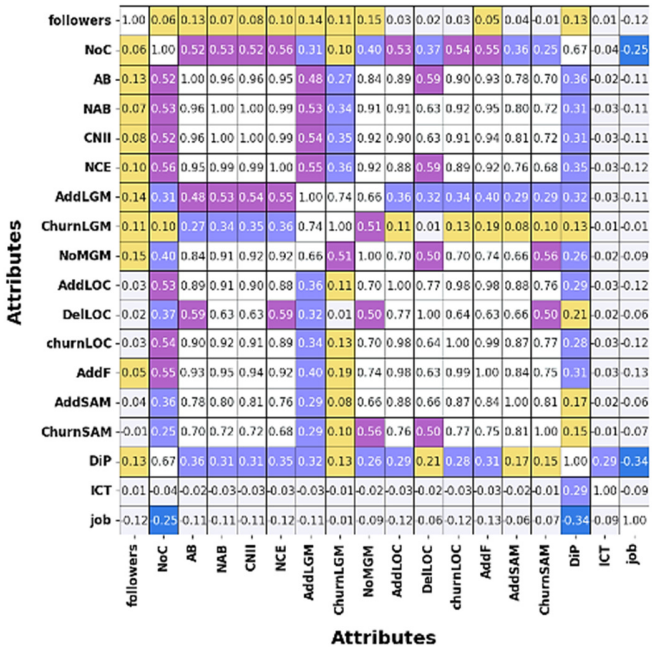

Fig. 2. Correlation matrix.

Table II presents a comparative study of the proposed EESQA-DELMOA method with current methods [19]. Based on $accur_y$, the EESQA-DELMOA model achieved an $accur_y$ of 98.74%, while the RF, DT, NB, ANN, DBN, CNN, and AlexNet approaches achieved 94.70%, 94.08%, 89.15%, 93.20%, 91.86%, 94.78%, and 92.34%, correspondingly. Based on $preci_n$, EESQA-DELMOA achieved 96.16% while the RF, DT, NB, ANN, DBN, CNN, and AlexNet methods had 92.82%, 89.52%, 95.60%, 90.93%, 92.12%, 90.30%, and 94.04%, respectively. Lastly, based on $F1_{score}$, EESQA-DELMOA achieved 89.41%, whereas the RF, DT, NB, ANN, DBN, CNN, and AlexNet methods had 81.28%, 79.79%, 88.28%, 85.58%, 85.88%, 84.58%, and 86.35%, respectively.

TABLE II. COMPARATIVE ANALYSIS OF EESQA-DELMOA WITH EXISTING CLASSIFIERS

Classifier	$Accur_y$	$Preci_n$	$Recal_l$	$F1_{score}$
Random Forest	94.70	92.82	81.41	81.28
Decision Tree	94.08	89.52	79.53	79.79
Naïve Bayes	89.15	95.60	82.01	88.28
Artificial Neural Network	93.20	90.93	80.83	85.58
DBN Model	91.86	92.12	80.44	85.88
CNN Method	94.78	90.30	79.55	84.58
AlexNet Model	92.34	94.04	79.82	86.35
EESQA-DELMOA	98.74	96.16	83.54	89.41

Table III presents the ablation evaluation of diverse components of EESQA-DELMOA. The SSNN classifier achieved $accur_y$ of 96.65%, $preci_n$ of 94.30%, $recal_l$ of 81.80%, and F1-score of 87.61%. The SSNN-only classifier with BAHB-based FS system achieved slightly higher $accur_y$ at 97.45%, $preci_n$ at 94.91%, $recal_l$ at 82.30%, and F1-score at 88.16%. EESQA-DELMOA (BAHB+SSNN+AMBOA) achieved the best results with an $accur_y$ of 98.74%, $preci_n$ of 96.16%, $recal_l$ of 83.54%, and F1-score of 89.41%, emphasizing the efficiency of the components.

TABLE III. ABLATION STUDY OF EESQA-DELMOA

Methods	$Accur_y$	$Preci_n$	$Recal_l$	$F1_{score}$
SSNN only classifier (without FS and optimization)	96.65	94.30	81.80	87.61
SSNN with BAHB feature selection (without optimization)	97.45	94.91	82.30	88.16
EESQA-DELMOA (BAHB + SSNN + AMBOA)	98.74	96.16	83.54	89.41

Table IV represents the K-fold cross-validation results of the proposed model over 10 folds. The model provides better accuracy and precision across all folds, indicating that it makes correct and reliable predictions. However, the lower recall shows that some actual cases are missed.

TABLE IV. K-FOLD CROSS-VALIDATION RESULTS

Fold	$Accur_y$	$Preci_n$	$Recal_l$	$F1_{score}$
Fold-1	94.36	95.31	77.59	85.54
Fold-2	91.03	90.91	78.41	84.20
Fold-3	90.21	93.83	76.63	84.36
Fold-4	90.02	95.32	80.21	87.11
Fold-5	95.12	95.24	77.76	85.62
Fold-6	90.42	95.40	81.29	87.78
Fold-7	96.83	94.90	81.26	87.55
Fold-8	90.51	92.14	76.72	83.72
Fold-9	91.80	93.82	75.03	83.38
Fold-10	96.33	93.45	77.42	84.68

IV. CONCLUSION

This study presented the EESQA-DELMOA method to assess software quality by inspecting developers' experience levels on code reliability and maintainability. The proposed EESQA-DELMOA approach utilizes the BAHB method to select the most relevant attributes and reduce dimensionality. In addition, an SSNN is applied for classification, and AMBOA is used for the optimization process. The proposed method was tested using a benchmark dataset, which suffers from class imbalance, with the UNK class dominating over minority classes such as BOT and NSE, which is highlighted by the use of strong evaluation metrics instead of relying solely on accuracy. The EESQA-DELMOA method demonstrates a strong ability in managing intricate data distributions and enhancing software quality prediction performance. The experimental results confirm that the EESQA-DELMOA approach considerably enhances performance over existing techniques.

However, this study has some limitations. The relatively small dataset might influence generalizability and robustness. In addition, the proposed model is mainly developer-centric and does not comprise a comprehensive analysis of software artifacts. Although the selected quality indicators provide initial validation, integrating artifact-level metrics could further strengthen the architecture. Future work will involve using larger datasets, improving class balance, and integrating developer, process, and artifact-level data. Combining these with advanced code analysis techniques and real-world data can make the model more scalable and suitable for practical software quality assessment.

DECLARATION OF COMPETING INTERESTS

The authors declare that they have no competing interests that could have appeared to influence the results of this study.

ACKNOWLEDGMENT

Not applicable.

DATA AVAILABILITY

The data that support the findings of this study are openly available in [18].

REFERENCES

[1] C. Ebert and P. De Neve, "Surviving global software development," *IEEE Software*, vol. 18, no. 2, pp. 62–69, Mar. 2001, https://doi.org/10.1109/52.914748.

[2] D. Graziotin, X. Wang, and P. Abrahamsson, "Happy software developers solve problems better: psychological measurements in empirical software engineering," *PeerJ*, vol. 2, Mar. 2014, Art. no. e289, https://doi.org/10.7717/peerj.289.

[3] E. E. Miandoab and F. S. Gharehchopogh, "A Novel Hybrid Algorithm for Software Cost Estimation Based on Cuckoo Optimization and K-Nearest Neighbors Algorithms," *Engineering, Technology & Applied Science Research*, vol. 6, no. 3, pp. 1018–1022, June 2016, https://doi.org/10.48084/etasr.701.

[4] S. Surakka and L. Malmi, "Cognitive skills of experienced software developer: Delphi study," in Kolin Kolistelut—Koli Calling 2004 - Proceedings of the Fourth Finnish/Baltic Sea Conference on Computer Science Education, 2004, pp. 37–46.

[5] P. Narang and P. Mittal, "Performance Assessment of Traditional Software Development Methodologies and DevOps Automation Culture," *Engineering, Technology & Applied Science Research*, vol. 12, no. 6, pp. 9726–9731, Dec. 2022, https://doi.org/10.48084/etasr.5315.

[6] S. Zamir, A. Rehman, H. Mohsin, E. Zamir, A. Abbas, and F. A. M. Al-Yarimi, "Integrating Pull Request Comment Analysis and Developer Profiles for Expertise-Based Recommendations in Global Software Development," *IEEE Access*, vol. 13, pp. 16637–16648, 2025, https://doi.org/10.1109/ACCESS.2025.3532386.

[7] H. Fawareh, H. M. Al-Shdaifat, A. R. M, F. A. Fawareh, and M. Khouj, "Investigates the Impact of AI-generated Code Tools on Software Readability Code Quality Factor," in *2024 25th International Arab Conference on Information Technology (ACIT)*, Dec. 2024, pp. 1–5, https://doi.org/10.1109/ACIT62805.2024.10876897.

[8] D. Rodriguez-Cardenas, S. A. Khan, P. Mandal, A. Nadkarni, K. Moran, and D. Poshyvanyk, "Testing Practices, Challenges, and Developer Perspectives in Open-Source IoT Platforms," in *2025 IEEE Conference on Software Testing, Verification and Validation (ICST)*, Mar. 2025, pp. 290–301, https://doi.org/10.1109/ICST62969.2025.10988986.

[9] S. Bernardo *et al.*, "Software Quality Assurance as a Service: Encompassing the quality assessment of software and services," *Future Generation Computer Systems*, vol. 156, pp. 254–268, July 2024, https://doi.org/10.1016/j.future.2024.03.024.

[10] V. Bayrı and E. Demirel, "AI-Powered Software Testing: The Impact of Large Language Models on Testing Methodologies," in *2023 4th International Informatics and Software Engineering Conference (IISEC)*, Dec. 2023, pp. 1–4, https://doi.org/10.1109/IISEC59749.2023.10391027.

[11] M. Samir, N. Sherief, and W. Abdelmoez, "Improving Bug Assignment and Developer Allocation in Software Engineering through Interpretable Machine Learning Models," *Computers*, vol. 12, no. 7, June 2023, https://doi.org/10.3390/computers12070128.

[12] A. A. MohamadSaleh and S. Alzahrani, "Development of a Maturity Model for Software Quality Assurance Practices," *Systems*, vol. 11, no. 9, Sept. 2023, https://doi.org/10.3390/systems11090464.

[13] R. G. Hussain, K. C. Yow, and M. Gori, "Leveraging an Enhanced CodeBERT-Based Model for Multiclass Software Defect Prediction via

Defect Classification," *IEEE Access*, vol. 13, pp. 24383–24397, 2025, https://doi.org/10.1109/ACCESS.2024.3525069.

[14] M. Baqar and R. Khanda, "The Future of Software Testing: AI–Powered Test Case Generation and Validation," in *Intelligent Computing*, 2025, pp. 276–300, https://doi.org/10.1007/978-3-031-92605-1_18.

[15] V. R. Gopinathan, "Software Engineering Practices for AI-Driven Systems: From Development to Deployment (MLOps Perspective)," *International Journal of Science, Research and Technology*, vol. 8, no. 1, pp. 13493–13500, Feb. 2025, https://doi.org/10.15662/IJSRAT.2025.0801002.

[16] W. Zhao, L. Wang, and S. Mirjalili, "Artificial hummingbird algorithm: A new bio-inspired optimizer with its engineering applications," *Computer Methods in Applied Mechanics and Engineering*, vol. 388, Jan. 2022, Art. no. 114194, https://doi.org/10.1016/j.cma.2021.114194.

[17] S. Arora and S. Singh, "Butterfly optimization algorithm: a novel approach for global optimization," *Soft Computing*, vol. 23, no. 3, pp. 715–734, Feb. 2019, https://doi.org/10.1007/s00500-018-3102-4.

[18] Q. Perez, C. Urtado, and S. Vauttier, "Dataset of Open-Source Software Developers Labeled by their Experience Level and Associated with their Software Metrics." Zenodo, Dec. 05, 2022, https://doi.org/10.5281/zenodo.7011334.

[19] B. Desai and R. K. Sungkur, "Software Quality Prediction Using Machine Learning," *International Journal of Software Innovation (IJSI)*, vol. 10, no. 1, pp. 1–35, Jan. 2022, https://doi.org/10.4018/IJSI.297997.