A Hybrid Metaheuristic Aware Enhanced Deep Learning Approach for Software Effort Estimation

Mahesh Badana
GITAM Deemed to be University, Vishakhapatnam, India | Department of Computer Science and Engineering, ANITS, Vishakhapatnam, India
b.mahesh498@gmail.com (corresponding author)

Mandava Kranthi Kiran
Department of Computer Science and Engineering, GITAM deemed to be University, Vishakhapatnam, India
kmandava@gitam.edu

Received: 2 September 2024 | Revised: 26 September 2024 | Accepted: 1 October 2024

Licensed under a CC-BY 4.0 license | Copyright (c) by the authors | DOI: https://doi.org/10.48084/etasr.8890

ABSTRACT

Software Effort Estimating (SEE) is a fundamental task in all software development lifecycles and procedures. Therefore, when deciding how to anticipate effort in a variety of project types, the comparative assessment of effort prediction methods has emerged as a standard strategy. Unfortunately, these studies include a range of sample techniques and error metrics, making a comparison with other work challenging. To overcome these drawbacks, this study proposes a deep learning model to effectively estimate software effort. The estimation is mainly focused on minimizing the cost and time consumption. The input data is taken from the dataset and preprocessing is performed to remove the noise content. Then the required features are extracted using the preprocessed data with the help of the simple and higher-order statistical features. A novel Modified Chaotic Enriched Jaya with Moth Flame Optimization (MCEJMO) algorithm is introduced for feature selection to enhance SEE accuracy. The estimation is performed using Multilayer Long Short-Term Memory (M-LSTM). The proposed method achieved a Mean Square Error (MSE) of 0.2825 for dataset 1 and 0.2285 for dataset 2.

Keywords-software effort estimation; statistical features; Jaya optimization algorithm; moth flame optimization; modified long short-term memory

I. INTRODUCTION

Software Effort Estimating (SEE) is one of the most difficult aspects of project management. Project managers have struggled for years to accurately estimate the time, money, and effort needed to complete initiatives to create schedules and budgets [1-2]. Numerous interconnected aspects that affect development effort and productivity are present during the software development process. As many of these interactions are poorly understood, accurate forecasting has proven to be challenging [3-4]. SEE for the planning of software projects and estimating methodologies' actual themes are crucial. Erroneous project planning that results from inadequate task estimation is a significant risk in the management of software engineering projects [5-6]. Optimization has been one of the most significant scientific areas in recent years. Optimization involves procedures to determine the ideal solution to a specific issue. Natural laws and processes have an impact on how computing systems are created to address difficult issues [7-8].

The key benefit of adopting machine learning algorithms is their ease of implementation and relatively high processing performance [9]. Nowadays, non-algorithmic strategies are becoming more and more important for SDEE because of the numerous constraints of algorithmic models. The imprecision of the inputs is accommodated by these strategies, and they are nevertheless able to deliver respectable outcomes [10]. Machine learning techniques use historical project data to create a regression model that will be used to forecast the amount of work needed for upcoming software projects. However, it has been discovered that none technique is completely stable and dependable under all circumstances. Furthermore, the properties of the dataset used to build the model have a general impact on how well any technique performs [11-12]. Estimating effort and development time improves performance by managing human resources, project schedules, cost estimation, and other factors in addition to enhancing software's success potential. These advantages minimize the likelihood of software failure and control project delays [13-14]. Additionally, they put forth a prediction model that helps a team by suggesting a story-point estimate for a specific user narrative. To forecast the size of new issues, such a method learns from the team's prior narrative point estimations [15-16]. Machine learning, and especially Deep Learning (DL), varies from traditional software engineering (SE) because its behavior is highly reliant on information from the outside world. The main contributions of this study are as follows:

- Effectively selects the optimal features for the Modified Chaotic Enriched Jaya with Moth Flame Optimization (MCEJMO).
- Employs a Modified multilayer Long-Short-Term Memory (M-LSTM) model to estimate the software estimation, using the GRU and RBM at the top and bottom layers, respectively.
- Hyperparameter tuning in the M-LSTM is performed using the MCEJMO algorithm.

II. PROPOSED METHODOLOGY

SEE is the process of estimating how much time, money, and resources are needed to build a software project. The goal of effort estimation is to provide stakeholders with a reasonable estimate of the resources required to complete a project, which can help to plan and budget resources, set schedules and deadlines, and manage risk. The block diagram of the software effort estimation is given in Figure 1.

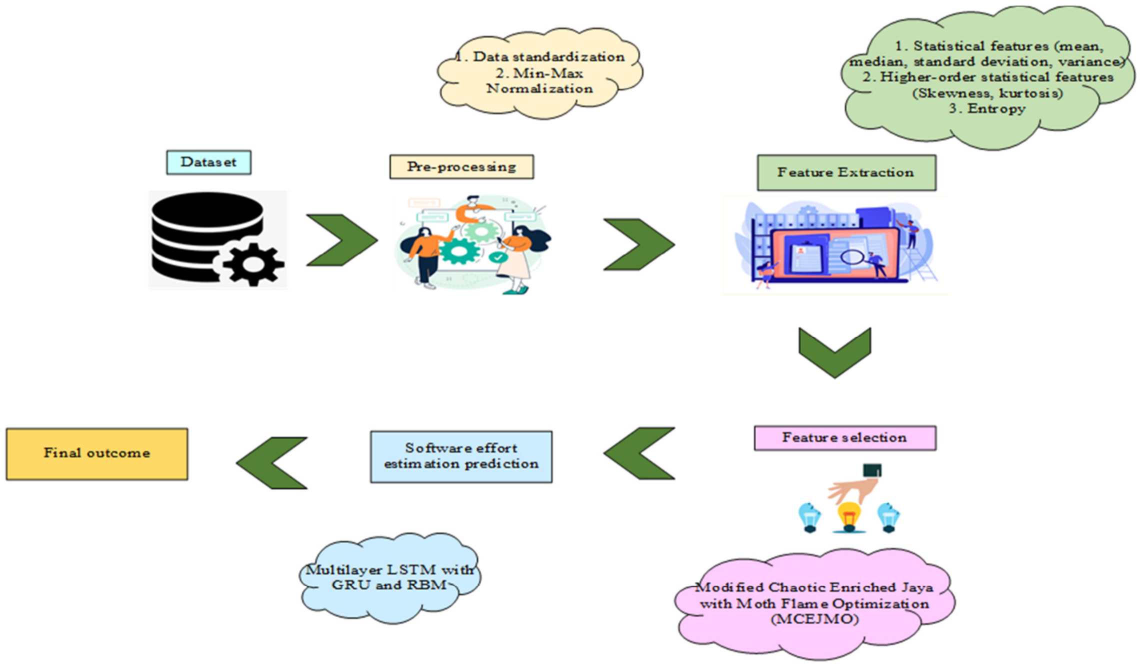
1. Data standardization
2. Min-Max Normalization
1. Statistical features (mean, median, standard deviation, variance)
2. Higher-order statistical features (Skewness, kurtosis)
3. Entropy
Dataset
Pre-processing
Feature Extraction
Feature selection
Software effort estimation prediction
Final outcome
Multilayer LSTM with GRU and RBM
Modified Chaotic Enriched Jaya with Moth Flame Optimization (MCEJMO)

Fig. 1. SEE block diagram.

The input data are chosen, standardized, and normalized during the preprocessing stage. The preprocessed data is given as input to the feature extractor. Here, higher-order statistical features, statistical features, and entropy are extracted. From the extracted features, the optimal characteristics are chosen using MCEJMO. Finally, SEE is predicted using the M-LSTM with RBM.

A. *Preprocessing*

The data taken from the dataset is given to the preprocessor to perform data standardization and normalization [17].

1) *Data Standardization*

Data standardization is a process of transforming raw data into a consistent, unambiguous, and standardized format to facilitate data processing, analysis, and comparison. Data standardization aims to ensure that data is easily understood and can be used by different systems, tools, and applications. Standardized data improve data quality, simplify data integration, and enable more accurate data analysis and decision-making [18].

2) *Data Normalization using Min-Max Normalization (MMN)*

Data normalization [20] is performed using the MMN model. In this method, the unnormalized data (k_i) is linearly scaled to predetermined lower and higher bound. Usually, the data is rescaled between a range of 0 and 1 or -1 and 1. This process is represented by:

$$MMN = \frac{k_{i,n}-\min(k_i)}{\max(k_i)-\min(k_i)}(nMax - nMin) + nMin \qquad (1)$$

where min and max stand for the i^{th} feature's minimum and maximum values, respectively. The lower and higher constraints to rescale the data are indicated by the numbers nMin and nMax, respectively. In this study, the classification performance is examined using both [0, 1] and [-1, 1] scales.

B. Feature Selection using MCEJMO

The extracted features are given to the feature selection option for selecting the best-required features. In feature selection, the MCEJMO algorithm is used, combining the JAYA optimization algorithm, MFO, and chaotic map function [21].

C. JAYA Optimization Algorithm

The population-based JAYA algorithm was developed to compare the best and worst solutions for each solution. This algorithm forces users to choose the best option while avoiding the worst option. In comparison to cutting-edge methods, the JAYA algorithm has demonstrated superior outcomes. Jaya does not have any algorithm-specific parameters and is readily modified to solve difficulties [22]. Let the objective function $g(x)$ having D dimension variables ($j = 1,2,\dots.,D$), the jth variable's value for the ith potential solution is $y_{i,j}$, then the ith candidate's position is represented as $y_i = (y_{i,1}, y_{i,2}, \dots\dots, y_{i,D})$. Similarly, the position of the best candidate is denoted as $y_{best} = (y_{best,1}, y_{best,2}, \dots\dots, y_{best,D})$, which has the best value for $g(x)$, and similarly for the worst. Then, the updated y_{ij} is given as:

$$y_{i,j}{}^{*} = y_{i,j} + rand_1.\left(y_{best,j} - |y_{i,j}|\right) - rand_2.\left(y_{worst,j} - |y_{i,j}|\right) \quad (2)$$

where the best solution is represented as $y_{best,j}$ and the worst is $y_{worst,j}$ for the jth variable. The $y_{i,j}$ is updated as $y_{ij}{}^{*}$, and the absolute value of $y_{i,j}$is $|y_{i,j}|$. The two random variables $rand_1$ and $rand_2$ are uniformly distributed between 0 and 1. The term $rand_1.\left(y_{best,j} - |y_{i,j}|\right)$ in represents the probability of the solution being directed to the best solution and $rand_2.\left(y_{worst,j} - |y_{i,j}|\right)$ is the probability of the solution being directed to the worst solution, respectively.

The updated solution $y_{ij}{}^{*} = (y_{i,1}{}^{*}, y_{i,2}{}^{*}, \dots\dots, y_{i,D}{}^{*})$ is acceptable if it provides the best function value. Moving away from the worst option and towards the best solution are two search-process outcomes produced by the JAYA algorithm. By seeking the optimal solution, the JAYA algorithm seeks to win.

D. Modified Chaotic Enriched Jaya with Moth Flame Optimization (MCEJMO)

To choose the best features effectively, the population of the flame is improved by the chaotic map function and the JAYA optimization solution. The chaotic map function is used to introduce randomness into the optimization process, which can help to overcome the limitations of traditional optimization algorithms. By adding chaotic elements to the optimization process, the hybrid population of the flame optimization algorithm can better explore the search space and find more optimal solutions. The current solutions of the JAYA algorithm are hybrid with the population of the flame with a chaotic map. The mathematical model is given as:

$$y_{i,j^*} = Dis_i * e^{ct} * \cos(2\pi t) + y_{i,j}\ (Chaotic\ map) \quad (3)$$

The distance from the moth to the flame is updated using the best and worst solutions of the JAYA algorithm. The distance for the optimal solution is obtained using (4):

$$Dis_i = rand_1.\left(y_{best,j} - |y_{i,j}|\right) - rand_2.\left(y_{worst,j} - |y_{i,j}|\right) \quad (4)$$

where Dis_i is the distance from the ith moth to the jth flame.

E. Software Effort Estimation (SEE) Using Multilayer LSTM (M-LSTM)

The M-LSTM involves a combination of several types of neural network layers, specifically a GRU layer, the LSTM layer, and the RBM layer. The GRU layer is presented in the top layer, which is often used in sequential data processing tasks such as SEE due to its ability to capture long-term dependencies [24]. Below the GRU layer, there is an LSTM layer, which is also a type of RNN layer that is designed to capture long-term dependencies. The LSTM layer uses memory cells to store information from previous time steps and decides when to let information flow and when to block it, allowing it to better handle long-term dependencies compared to traditional RNNs.

The additional LSTM layer enables the addition of more hyperparameters. Hyperparameter tuning was performed based on the MCEJMO algorithm. The hyperparameters used in this study are shown in Table I.

TABLE I. HYPERPARAMETERS FOR M-LSTM TRAINING

Parameter	Value
Activation function	Leaky ReLU, sigmoid
Loss function	MSE
Dropout rate	0.14
Optimizer	MCEJMO
Number of epochs	47
Learning rate	0.01
Batch size	80

III. RESULTS AND DISCUSSION

This section discusses the results obtained using the proposed model and compares them with existing techniques such as Recurrent Neural Network (RNN), traditional LSTM, and GRU. Along with prediction performance, the effectiveness of feature selection is also assessed using currently available techniques such as Moth Flame Optimization (MFO), Jaya Optimization (JO), Particle Swarm Optimization (PSO) [26], and Genetic Algorithm (GA) [27]. The proposed method was implemented using Python and used two datasets released by ISBSG version 11 [28], given as Dataset 1 and Dataset 2. The collection contains more than 5,000 industrial projects that were developed using a variety of programming languages and implemented in accordance with a number of different software development life cycles. Each of the projects falls under the category of either new or upgraded development. In addition, the size of the software for each project was calculated in function points by making use of industry standards such as IFPUG and COSMIC, amongst others. The ISBSG also assigns grades to the project data quality, from "A" to "D," with "A" standing for the highest-quality projects, followed by "B," and so on. Dataset 1 consists of A and D, and dataset 2 consists of B and C.

A. Performance Metrics

The error metrics Mean Square Root Error (MSRE), Mean Absolute Percentage Error (MAPE), Normalized Mean Square Error (NMSE), Mean Square Error (MSE), and Root Mean Square Error (RMSE) were used for evaluation.

MAPE [27] divides the demand by the total number of different absolute mistakes and is calculated by:

$$MAPE = \frac{1}{T_j}\sum_{j=1}^{T_j}\left|\frac{A_v - F_v}{A_v}\right| \quad (5)$$

where T_j is the occurrences of the summation iteration in total, A_v is the actual value, and F_v is the forecast value.

MSE measures the average squared difference between the predicted and the actual values [27]. A model's effectiveness is evaluated using the MSE loss to guide its training. The smaller the MSE, the better the model fits the data.

$$MSE = \frac{1}{N}(A_v - P_v)^2 \quad (6)$$

where A_v is the actual value and P_v is the predicted value of the target variable.

MSRE measures the average error in a set of predictions or forecasts [27]. It is a measure of how well the predictions or forecasts match the actual values.

$$MSRE = \sqrt{\frac{1}{N}(A_v - P_v)^2} \quad (7)$$

where N is the number of predictions or forecasts.

NMSE [27] measures the difference between two signals and is typically used for evaluating the performance of a prediction algorithm. It is defined as the ratio of the MSE of the predicted signal to the variance of the target signal.

$$NMSE = \left(\frac{1}{n}\right) * sum\left(\frac{(A_v - P_v)^2}{var(A_v)}\right) \quad (8)$$

where $var(A_v)$ is the variance of the actual values.

RMSE [27] is given by:

$$RMSE = \sqrt{\frac{1}{N}(A_v - P_v)^2} \quad (9)$$

B. Overall Performance Comparison

Comparisons are made between the performance of the suggested approach and that of currently used methods such as GA, JA, MFO, and PSO. Table II provides a comparison of the proposed and the existing techniques for Dataset 1.

TABLE II. COMPARISON FOR DATASET 1

Metrics	GA	JA	MFO	PSO	Proposed
MSE	0.2957	0.3195	0.3104	0.3078	0.2825
MSRE	0.2741	0.2853	0.3030	0.2659	0.2627
NMSE	0.3989	0.4234	0.4294	0.4016	0.3816
RMSE	0.3687	0.3427	0.3493	0.3216	0.3729
MAPE	0.2898	0.3131	0.3042	0.3017	0.2768

The results show that the proposed algorithm had the lowest values for MSE, MSRE, and MAPE, indicating that it had the best performance compared to the other algorithms. Table III shows a comparison of error metrics for Dataset 2.

TABLE III. COMPARISON FOR DATASET 2

Metrices	GA	JA	MFO	PSO	Proposed
MSE	0.2511	0.2490	0.2584	0.2377	0.2285
MSRE	0.2450	0.2151	0.2308	0.2553	0.2125
NMSE	0.3751	0.3508	0.3698	0.3727	0.3334
RMSE	0.2221	0.1950	0.2092	0.2314	0.1926
MAPE	0.2500	0.2848	0.2577	0.2966	0.2469

MAPE measures the average absolute percentage error between the expected and real values. The fit between the anticipated and actual values is better when the MAPE value is smaller. Figure 2 shows the comparison of MAPE values among the existing and proposed methods.

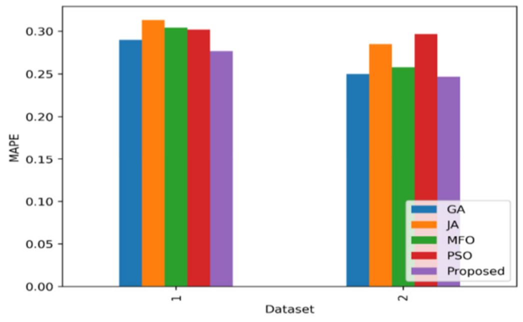

Fig. 2. Comparison of MAPE values for the proposed and existing techniques.

This figure compares the MAPE values for Dataset 1 and Dataset 2. The MAPE values were lower for Dataset 2 compared to Dataset 1.

MSE denotes the mean of the squared deviations between the expected and observed values. The fit between the anticipated and actual values is better when the MSE is smaller. Figure 3 compares the MSE values for the currently used and the proposed techniques.

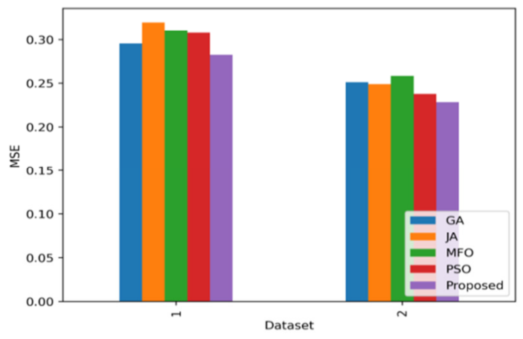

Fig. 3. Comparison of MSE values for the proposed and existing techniques.

The graph illustrates the MSE results between Dataset 1 and Dataset 2. Compared to Dataset 1, the MSE values for Dataset 2 are lower. MSRE is similar to MSE, but it is normalized by the mean of the actual values. This helps to adjust for the scale of the data. Figure 4 compares the MSRE values for the existing and the proposed techniques on both datasets Compared to Dataset 1, the MSRE values for Dataset 2 are lower. NMSE is similar to MSE, but it is normalized by the variance of the actual values. This helps to adjust for the variability of the data. Figure 5 displays a comparison of the NMSE values between the currently used and the proposed techniques. The graphic contrasts datasets 1 and 2's NMSE values. Comparing datasets 1 and 2, the NMSE values are lower for the latter.

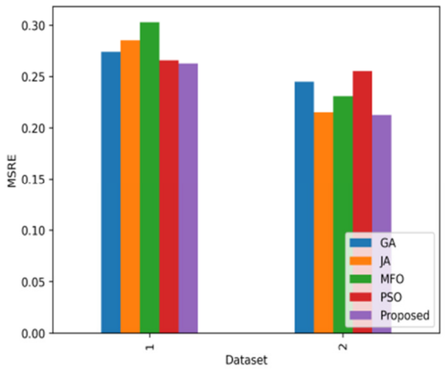

Fig. 4. Comparison of MSRE values for the proposed and existing techniques.

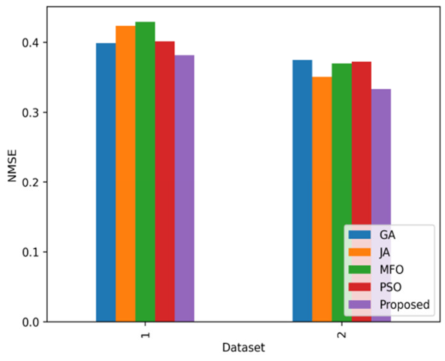

Fig. 5. Comparison of NMSE values for the proposed and existing techniques.

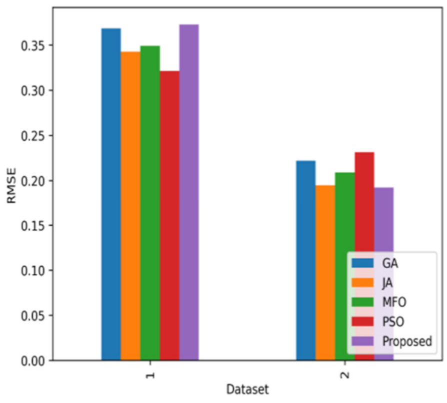

Fig. 6. Comparison of RMSE values for the proposed and existing techniques.

This metric is the square root of the MSE and gives a more intuitive representation of the error, as it is expressed in the same units as the data. Figure 6 compares the RMSE values of the existing and the proposed techniques. Compared to Dataset 1, the RMSE values for Dataset 2 are lower. Figure 7 shows a comparison between the actual results and the predicted results by the proposed MCEJMO method.

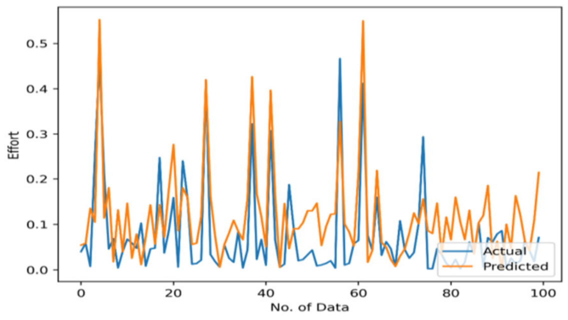

Fig. 7. Comparison of actual and predicted values for the proposed MCEJMO technique.

IV. CONCLUSION

This study introduced a new hybrid optimization model to effectively predict SEE. Preprocessing involves selecting the input data from the dataset to standardize and normalize them. The feature extractor receives the preprocessed data as input, extracting normal and higher-order statistical features and entropy. The proposed MCEJMO method is used to select the best of the retrieved features. Finally, the improved LSTM with RBM and GRU is used to predict the SEE.

REFERENCES

[1] R. Mohanani, I. Salman, B. Turhan, P. Rodríguez, and P. Ralph, "Cognitive Biases in Software Engineering: A Systematic Mapping Study," *IEEE Transactions on Software Engineering*, vol. 46, no. 12, pp. 1318–1339, Sep. 2020, https://doi.org/10.1109/TSE.2018.2877759.

[2] A. Calleja, J. Tapiador, and J. Caballero, "The MalSource Dataset: Quantifying Complexity and Code Reuse in Malware Development," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 12, pp. 3175–3190, Sep. 2019, https://doi.org/10.1109/TIFS.2018.2885512.

[3] J. Iglhaut, C. Cabo, S. Puliti, L. Piermattei, J. O'Connor, and J. Rosette, "Structure from Motion Photogrammetry in Forestry: a Review," *Current Forestry Reports*, vol. 5, no. 3, pp. 155–168, Sep. 2019, https://doi.org/10.1007/s40725-019-00094-3.

[4] C. Tam, E. J. da C. Moura, T. Oliveira, and J. Varajão, "The factors influencing the success of on-going agile software development projects," *International Journal of Project Management*, vol. 38, no. 3, pp. 165–176, Apr. 2020, https://doi.org/10.1016/j.ijproman.2020.02.001.

[5] W. Rafique, L. Qi, I. Yaqoob, M. Imran, R. U. Rasool, and W. Dou, "Complementing IoT Services Through Software Defined Networking and Edge Computing: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1761–1804, 2020, https://doi.org/10.1109/COMST.2020.2997475.

[6] N. Dey, A. E. Hassanien, C. Bhatt, A. S. Ashour, and S. C. Satapathy, Eds., *Internet of Things and Big Data Analytics Toward Next-Generation Intelligence*, vol. 30. Cham, Switzerland: Springer International Publishing, 2018.

[7] R. Jayanthi and L. Florence, "Software defect prediction techniques using metrics based on neural network classifier," *Cluster Computing*,

vol. 22, no. 1, pp. 77–88, Jan. 2019, https://doi.org/10.1007/s10586-018-1730-1.

[8] S. S. Rathore and S. Kumar, "A study on software fault prediction techniques," *Artificial Intelligence Review*, vol. 51, no. 2, pp. 255–327, Feb. 2019, https://doi.org/10.1007/s10462-017-9563-5.

[9] N. Qamar, F. Batool, and K. Zafar, "Efficient effort estimation of web based projects using neuro-web," *International Journal of Advanced and Applied Sciences*, vol. 5, no. 11, pp. 33–39, 2018.

[10] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 75–88, Dec. 2019, https://doi.org/10.1016/j.jpdc.2019.07.007.

[11] P. Pospieszny, B. Czarnacka-Chrobot, and A. Kobylinski, "An effective approach for software project effort and duration estimation with machine learning algorithms," *Journal of Systems and Software*, vol. 137, pp. 184–196, Mar. 2018, https://doi.org/10.1016/j.jss.2017.11.066.

[12] P. Suresh Kumar, H. S. Behera, A. K. K, J. Nayak, and B. Naik, "Advancement from neural networks to deep learning in software effort estimation: Perspective of two decades," *Computer Science Review*, vol. 38, Nov. 2020, Art. no. 100288, https://doi.org/10.1016/j.cosrev.2020.100288.

[13] R. Silhavy, P. Silhavy, and Z. Prokopova, "Evaluating subset selection methods for use case points estimation," *Information and Software Technology*, vol. 97, pp. 1–9, May 2018, https://doi.org/10.1016/j.infsof.2017.12.009.

[14] S. H. S. Moosavi and V. K. Bardsiri, "Poor and rich optimization algorithm: A new human-based and multi populations algorithm," *Engineering Applications of Artificial Intelligence*, vol. 86, pp. 165–181, Nov. 2019, https://doi.org/10.1016/j.engappai.2019.08.025.

[15] C. Liu, D. Yang, X. Xia, M. Yan, and X. Zhang, "A two-phase transfer learning model for cross-project defect prediction," *Information and Software Technology*, vol. 107, pp. 125–136, Mar. 2019, https://doi.org/10.1016/j.infsof.2018.11.005.

[16] K. Curcio, T. Navarro, A. Malucelli, and S. Reinehr, "Requirements engineering: A systematic mapping study in agile software development," *Journal of Systems and Software*, vol. 139, pp. 32–50, May 2018, https://doi.org/10.1016/j.jss.2018.01.036.

[17] N. K. Jain, S. Celo, and V. Kumar, "Internationalization speed, resources and performance: Evidence from Indian software industry," *Journal of Business Research*, vol. 95, pp. 26–37, Feb. 2019, https://doi.org/10.1016/j.jbusres.2018.09.019.

[18] W. Gao, J. Alsarraf, H. Moayedi, A. Shahsavar, and H. Nguyen, "Comprehensive preference learning and feature validity for designing energy-efficient residential buildings using machine learning paradigms," *Applied Soft Computing*, vol. 84, Nov. 2019, Art. no. 105748, https://doi.org/10.1016/j.asoc.2019.105748.

[19] A. Kaushik and N. Singal, "A hybrid model of wavelet neural network and metaheuristic algorithm for software development effort estimation," *International Journal of Information Technology*, vol. 14, no. 3, pp. 1689–1698, May 2022, https://doi.org/10.1007/s41870-019-00339-1.

[20] A. Zakrani, A. Idri, and M. Hain, "Software Effort Estimation Using an Optimal Trees Ensemble: An Empirical Comparative Study," in *Proceedings of the 8th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT'18)*, Genoa, Italy, 2020, vol. 1, pp. 72–82, https://doi.org/10.1007/978-3-030-21005-2_7.

[21] M. S. Khan, Ch. A. Ul Hassan, M. A. Shah, and A. Shamim, "Software Cost and Effort Estimation using a New Optimization Algorithm Inspired by Strawberry Plant," in *2018 24th International Conference on Automation and Computing (ICAC)*, Newcastle upon Tyne, UK, Sep. 2018, pp. 1–6, https://doi.org/10.23919/IConAC.2018.8749003.

[22] M. Choetkiertikul, H. K. Dam, T. Tran, T. Pham, A. Ghose, and T. Menzies, "A Deep Learning Model for Estimating Story Points," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 637–656, Jul. 2019, https://doi.org/10.1109/TSE.2018.2792473.

[23] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software Engineering Challenges of Deep Learning," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Prague, Aug. 2018, pp. 50–59, https://doi.org/10.1109/SEAA.2018.00018.

[24] M. Azzeh, A. B. Nassif, and C. L. Martín, "Empirical analysis on productivity prediction and locality for use case points method," *Software Quality Journal*, vol. 29, no. 2, pp. 309–336, Jun. 2021, https://doi.org/10.1007/s11219-021-09547-0.

[25] Z. abdelali, H. Mustapha, and N. Abdelwahed, "Investigating the use of random forest in software effort estimation," *Procedia Computer Science*, vol. 148, pp. 343–352, Jan. 2019, https://doi.org/10.1016/j.procs.2019.01.042.

[26] B. K. Kumar, S. Bilgaiyan, and B. S. P. Mishra, "Software Effort Estimation Based on Ensemble Extreme Gradient Boosting Algorithm and Modified Jaya Optimization Algorithm," *International Journal of Computational Intelligence and Applications*, vol. 23, no. 01, Mar. 2024, Art. no. 2350032, https://doi.org/10.1142/S1469026823500323.

[27] K. K. Beesetti, S. Bilgaiyan, and B. S. P. Mishra, "Software Effort Estimation through Ensembling of Base Models in Machine Learning using a Voting Estimator," *International Journal of Advanced Computer Science and Applications*, vol. 14, no. 2, 2023.

[28] "ISBSG Home," *ISBSG*. https://www.isbsg.org/home/.