Journal of Intelligent Systems and Internet of Things
Vol. 18, No. 01, PP. 274-287, 2026
DOI: https://doi.org/10.54216/JISIoT.180121

Feature Weight-Based Optimization in Software Development Model Using Meta Heuristic Machine-Learning Algorithms

N. Durga Devi[1,*], Tirimula Rao Benala[2]

[1]Research Scholar, Department of Computer Science and Engineering-Information Technology, Jawaharlal Nehru Technological University Gurajada Vizianagaram, Dwarapudi, Vizianagaram, Andhra Pradesh-535003, India

[2]Department of Information Technology, JNTU-GV College of Engineering, Vizianagaram, Jawaharlal Nehru Technological University Gurajada Vizianagaram, Dwarapudi, Vizianagaram, Andhra Pradesh-535003, India

Emails: durgadevisd3@gmail.com; b.tirimula@gmail.com

Abstract

System users are increasingly interested in software correctness and efficiency checks prior to usage. Programmers in the twenty-first century are therefore making a conscious effort to create software that is more accurate, more efficient, and less prone to bugs. A software development model utilizing metaheuristic machine learning algorithms involves using metaheuristic optimization techniques to enhance various aspects of the software development lifecycle, such as optimizing machine learning models, hyperparameters, and even software architecture. This research propose novel technique in feature weight model based optimization in software development utilizing Meta heuristic ML method. Here the feature weight and feature selection is carried out for software model using support additive regression Laplacian score perceptron neural network. Then the software model parameter optimization is carried out using ant binary swarm component encoder optimization method. Simulation analysis is carried out in terms of training accuracy, MAR (Mean absolute residual), Mean balanced relative error (MBRE), F-measure.

Received: March 05, 2025 Revised: June 03, 2025 Accepted: July 08, 2025

Keywords: Feature weight model; Software development; Meta heuristic; Machine learning model; Regression Laplacian score

1. Introduction

From gathering requirements to testing and maintenance, software project development involves a variety of tasks that must be completed within a given timeframe and budget. The foundation of software engineering is logical as well as analytical work. Because of the quick growth of technology and high rate of change in client needs, software development is more difficult than other types of engineering projects. Thus, it becomes difficult to accomplish particular goals while meeting a variety of limitations for software project management. The quick development of technology has led to a scenario where hardware is less expensive than programming [1]. No matter how fast the hardware is, software optimization is also necessary for noticeable performance increase. This places the onus on developers to provide flexible software within a limited budget and period in order to close the gap between software and hardware development. One such area that is developing quickly is the technology sector. Furthermore, trend towards software-defined architectures to streamline management as well as foster innovation through programmability is growing in tandem with the development of network technologies and applications [2]. Large and dispersed projects present special planning and estimate challenges. The most crucial component of software project

management is project planning, which includes a number of managerial as well as technical procedures that can be broadly categorized as creating project plan, carrying it out, foreseeing potential issues, creating preliminary solutions to those issues [3]. Estimating software costs and effort falls under project planning phase as well as involves figuring out how much a project will cost, how many person-hours it will take to finish, how long it will take. Project failure and cost increases might arise from inaccurate estimation [4]. When making decisions, accurate estimations are essential. Underestimating project effort could result in a situation where work cannot be completed due to a lack of funds and time commitments, while overestimating could result in a project proposal being rejected. For software quality assurance, software testing is a crucial step in software development life cycle. By identifying possibly flawed software modules before to the software product's release, defect prediction can help the quality assurance teams manage the limited testing resources in a reasonable manner. As a result, efficient defect prediction can enhance software quality and reduce testing expenses [5]. Most current research builds fault prediction tools using a variety of machine learning techniques. Specifically, a variety of classification methods, including DT, LR, NN, NB, RF, SVM, and ensemble methods, have been employed as defect prediction models. Various feature selection techniques have been used to choose the best feature subset for defect prediction since redundant and irrelevant features in defect data may impair performance of classification methods. Gottfried Leibniz's theories and concepts were the first to develop artificial intelligence (AI). An evolutionary model of the human brain put forward by McCulloch and Pitts sparked interest in artificial neural networks (ANNs). ANNs are capable of learning, identifying, and resolving a variety of challenging issues. The most widely used and primary machine learning (ML) algorithmic approaches nowadays are artificial neural networks (ANNs) and deep learning (DL) methodologies. DL algorithms outperform traditional ML techniques when adequate data and processing capacity are available [6].

Contribution of this research: Existing feature weighting models in software development often struggle with effectively handling the complexities of real-world datasets and the nuanced relationships between features. Specifically, they may lack the ability to adaptively learn sample weights, consider inter-feature dependencies, and provide meaningful interpretations of feature importance beyond the model's context. To use a Meta heuristic ML method to suggest a new technique for feature weight model-based optimization in software development. Here, support additive regression is utilized to determine feature weight as well as feature selection for the software model. Perceptron neural network with Laplacian score. The Ant Binary Swarm Component Encoder Optimization Model is then used to optimize the software model parameters.

2. Literature Survey

Most of the earlier research created models for estimating effort using machine-learning techniques including decision trees, SVM, genetic algorithms, ANN. Earlier research that used general-purpose regression methods to create models for estimating software work. Work [7] used the Best (K) technique and kNN with a certain number of "k"s to accurately estimate software development work on nine datasets, including one ISBSG dataset. They concluded that choosing neighbors from areas with low variation yields superior outcomes when using the estimation by analogy approach. One drawback of this method is that it is unable to forecast a value on a dimension, like software development effort, that is beyond the range of values in the training data [8]. The automatically transformed linear baseline model (ATLM), a multiple linear regression approach put forth by [9], may have further comparable problems. With no need for parameter customization, ATLM is regarded as having good performance across a range of project types and is simple to deploy. Before examining the evaluation criteria using hybrid linear regression and decision trees across three datasets, the author [10] employed a binary genetic method for feature selection. Other researchers used log-linear regression (LLR). [11] investigated the application of ensemble learning for SDP. Authors introduced a system that trains naive Bayes, SVM, and random forest separately before combining them into an ensemble methodology using soft voting technique. Suggested approach maintained strong stability while achieving one of the best results. However, for the identical problem, [12] investigated the possibilities of a bidirectional long short-term model. Random and synthetic minority oversampling methods were used to test the method. Despite the fact that both tests demonstrated excellent performance, the authors concluded that the SDP problem's class imbalances made random oversampling superior. Finally yet importantly, [13] suggested a deep Q-learning network (DQN)-based approach for eliminating superfluous features. The study used 22 SDP datasets, and the authors thoroughly tested a number of methods both with and without DQN.

NLP techniques have a wide range of applications in ML, [14] have given a thorough rundown of ML approaches appropriate for these kinds of situations. The reported application cases span a variety of industries, including entertainment and healthcare. Nature of the computational methods employed in NLP is also thoroughly examined in the work. Image-text, audio-visual, audio-image-text, labelling, document-level, sentence-level, phrase-level, word-

level methods are among the various methods used in sentiment analysis. For this use case, the writers also surveyed a number of datasets. A model for SDP based on the bidirectional encoder representations from transformers (BERT) NLP approach was suggested in work [15]. While authors used term frequency–inverse document frequency for semantic analysis, they used keywords and string selection for text mining [16]. There is work that optimizes software cost as well as effort evaluation utilizing hybrid approach. A hybrid approach blends two or more approaches. Author [17] introduced the Whale-Crow Optimization (WCO) method for software cost estimate. To determine best regression coefficients for a regression model, WCO combines Crow Search Algorithm (CSA) and WOA. In order to improve software estimates, BATGSA hybrid method—which is based on BAT as well as Gravitational Search Algorithm (GSA)—is introduced in [18]. During the exploration phase, the BAT algorithm determines the bat's hunting and routing behavior. This is further enhanced by employing the GSA's gravitational pull effect to increase the BAT's seeking speed. In comparison to the COCOMO model, better estimation is obtained. In order to estimate effort and produce the lowest error rate, work [19] employed computational intelligence techniques, specifically PSO, using clustered data within three to four groups. In order to address this, the author [20] suggested a novel method that uses Grey Relational Analysis of Grey System Theory. Early in a software development process, feature subset selection as well as software effort prediction are used to two different dataset types. Furthermore, [21] used the GRA approach on the Kemerer dataset and contrasted the outcomes with those of other fundamental models.

2.1 Problem Formulation

It is quite challenging to determine which meta-heuristic approach offers superior accuracy in software effort evaluation, which is primary flaw in the research now in publication. The following are the primary causes of the meta-heuristic algorithms' unpredictable performance. Every existing work evaluates method using a various dataset. Public datasets that are accessible vary widely in size and contain information from a range of organizational projects. On one dataset, the meta-heuristic algorithm may perform exceptionally well, but on other datasets, it may perform poorly or worse.

3. Proposed MetaFE-SD (Metaheuristic Feature Estimation for Software Development)

The application of the analytical research method made it possible to pinpoint issues related to the usage of hybrid machine learning approaches, which are employed in software development processes to increase productivity. An analysis of the artificial intelligence research was carried out using the statistical method, which aids in comprehending the quantity and reasons of mistakes made in enhancing AI, which forms the foundation for the long-term growth of automation systems and the enhancement of software development procedures. The analytical approach also looked at ways to make data processing mechanisms work better, how to use these programs, and how to make hybrid machine learning approaches more productive and sustainable.

This study uses a wrapper method for feature selection. Using a learning method that employs a search approach to explore space of possible feature subsets, wrapper techniques rank them based on how well they perform in a particular algorithm. Because feature selection procedure is customized for particular classification method being used, wrapper approaches typically perform better than filter methods. On the other hand, because wrapper approaches necessitate calculation of every feature set utilizing classifier method, they are unreasonably time- as well as resource-intensive for high-dimensional data. Structure of wrapper-based feature selection method is shown in Figure 1. Database was originally divided into separate training and testing sets, as shown in Figure 1. An 80:20 split between the train and test sets was taken into consideration in this work. After that, train set was put into wrapper-based feature selection framework, where the metaheuristic algorithm (Equation (1) optimized a composite function. In order to create a classification model, framework first chooses a subset of features and applies designated ML method. After assessing the model's performance, the optimizer feeds a new set of characteristics to create a new categorization method based on the results. Until a predetermined termination threshold is satisfied, iterative classification method creation, method performance evaluation, feeding of updated features into classification method continue.

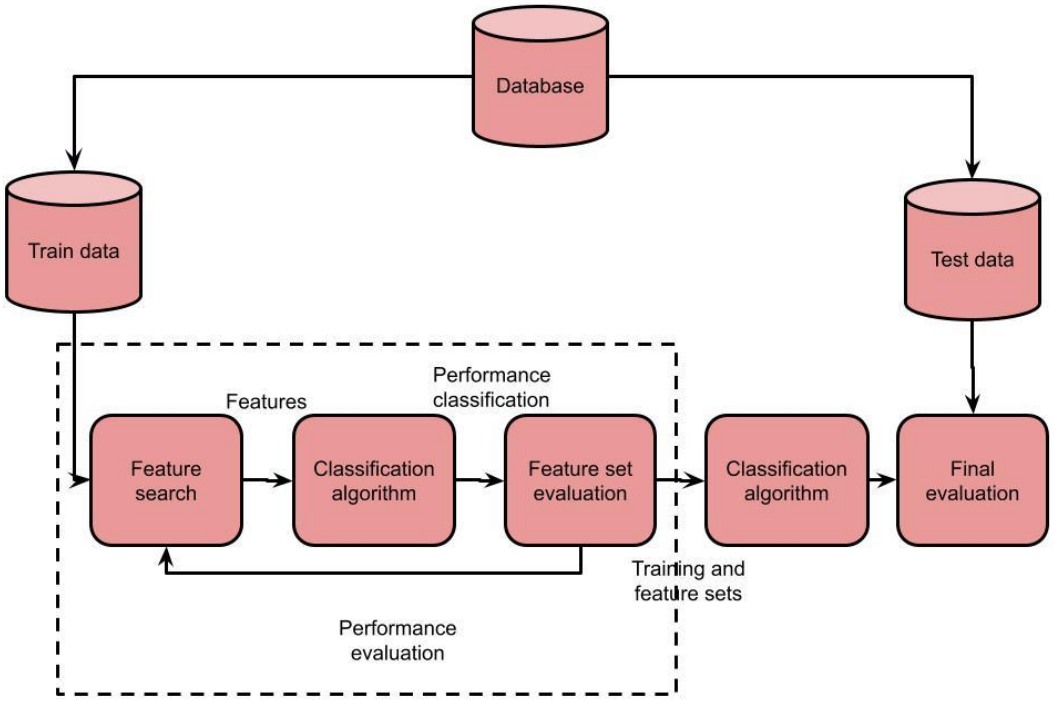


Figure 1. MetaFE-SD (Metaheuristic Feature Estimation for Software Development)

Fitness Function

Associated fitness function of an optimizer is used to assess its efficacy. The amount of features used for classification and the classification error rate both affect fitness function in feature selection. If chosen feature subset lowers classification error rate as well as number of features selected, it is considered a good solution. Fitness function listed below is taken into account by eqn (1)

$$+ \text{Fitness } = \lambda\gamma_S(D) + (1-\lambda)\frac{|S|}{|F|} \quad (1)$$

$\lambda \in [0, 1] \lambda \in 0, 1$ is a factor corresponding to length of reduced subset and significance of classification performance, $\gamma S(D) \gamma$SD is classification error calculated by classifier.

3.1 Feature weight and feature selection using software model using support additive regression Laplacian score perceptron neural network (SARLSPNN)

Figure 2 illustrates that the input data set consists of both training and testing data sets. Feature subset selection selects related feature subsets and eliminates unrelated feature subsets. They become the training data set of a chosen feature subset as well as testing data set of a chosen feature subset after the training and testing data sets have eliminated irrelevant feature subsets. Training sets with chosen feature subsets are used to train classifiers and classifications from randomly generated training sets are combined to improve classification. By using random sampling to divide a training set into many new training sets, the bagging classifier creates methods based on new training sets. Each model votes to determine the ultimate categorizations outcome. Additionally, it lessens variation and prevents overfitting. The bagging process is described as follows. Bagging creates m new training sets Di, every size n', from a regular training set D of size n.

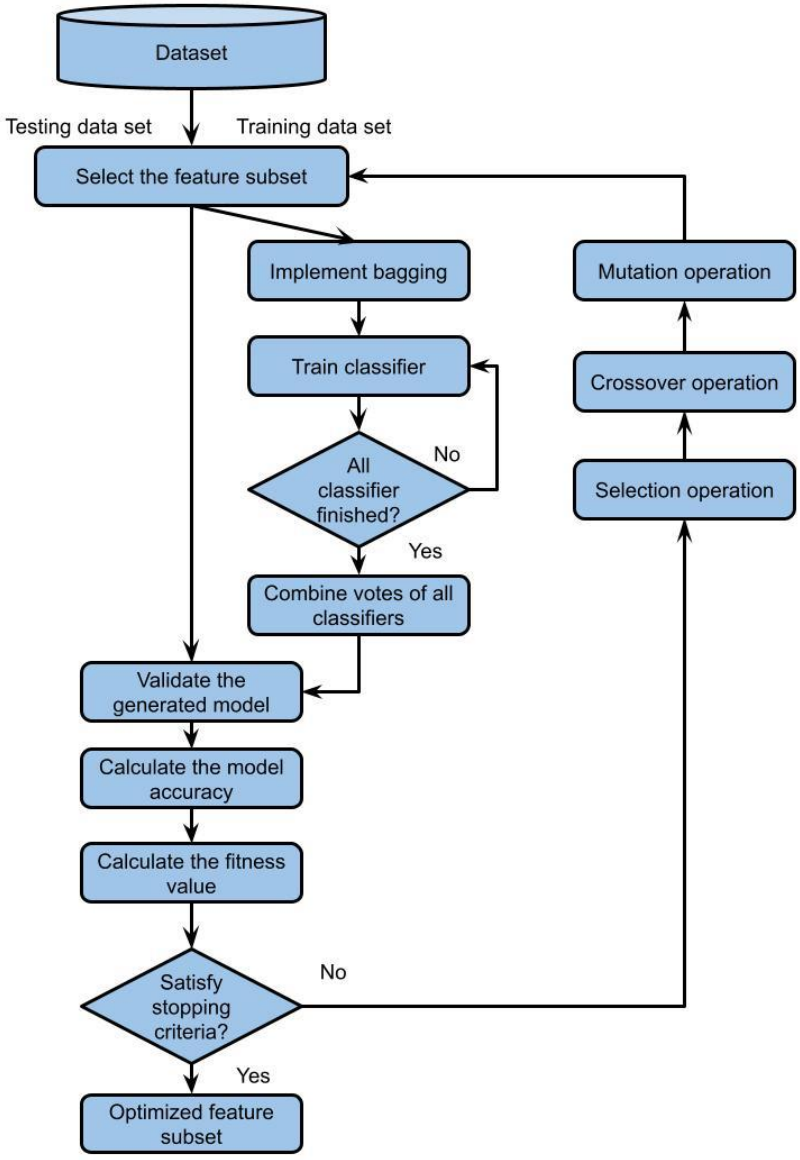


Figure 2. Activity Diagram of the Integration of SARLSPNN

solves following Primal Problem (PP) over ω, b, ξ, ξ∗ to yield repressor function, f(x):= ω ′φ(x) + b, for a data set D:= {(xi, yi): i ∈ ID} with xi ∈ Rd and y ∈ R by eqn (2)

$$\min_{\omega,\xi_i\geq 0\xi_i^*\geq 0} \frac{1}{2}\omega'\omega + C\sum_{i\in I_D}(\xi_i + \xi_i^*)$$

$$\text{s.t.} \quad \begin{aligned} y_i - \omega'\phi(x_i) - b &\leq \epsilon + \xi_i, \quad \forall i \in \mathcal{J}_D \\ \omega'\phi(x_i) + b - y_i &\leq \epsilon + \xi_i^*, \quad \forall i \in \mathcal{J}_D \end{aligned} \qquad (2)$$

Function φ: Rd → H translates x into a high dimensional Hilbert space. While regularisation parameter, `ϵ > 0, determines permitted departure of the f(xi) from yi, the regularisation parameter, C > 0, trades off size of ω as well as number of slacks. In reality, PP's Dual Problem (DP) is frequently used to solve it by eqn (3)

$$\max_{\alpha,\alpha^*} -\frac{1}{2}\sum_{i\in I_D}\sum_{j\in I_D}(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)K(x_i,x_j)-\epsilon\sum_{i\in I_D}(\alpha_i+\alpha_i^*)+\sum_{i\in I_D}y_i(\alpha_i-\alpha_i^*)$$

$\sum_{i\in I_D}(\alpha_i-\alpha_i^*)=0, 0\le\alpha_i\le C, 0\le\alpha_i^*\le C, i\in I_D$ (3)

where αi and α ∗ i are respective Lagrange multipliers of (2) and (3), $\omega=\overline{\sum_{i\in I_D}(\alpha_i-\alpha_i^*)\phi(x_i)}$ and $K(x_i,x_j)=$ φ(xi) ′φ(xj). Utilizing these expressions, regressor function is given by eqn (4)

$f(x)=\omega'\phi(x_i)+b=\sum_{i\in I_D}(\alpha_i-\alpha_i^*)K'(x_i,x)+b$ (4)

The density functions of y for a given x are thus determined by eqn(5).

$$p^L(y\mid x;\sigma)=\frac{1}{2\sigma}\exp\left(-\frac{|y-f(x)|}{\sigma}\right)$$

$p^G(y\mid x;\sigma)=\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y-f(x))^2}{2\sigma^2}\right)$ (5)

Logarithm function of L(σ) can be maximized to yield the expressions of σ, further presuming that p(x) is independent of σ. Although there are more versions, these approaches search for feature that maximizes KL distance between p(x j) (or p(y|x j)) and p(y). In contrast to theirs, employ a wrapper technique for SVR issue to measure difference between two density functions. Two distributions, p(y) and q(y), are given by eqn (6)

$D_{KL}(p(y);q(y))=\int p(y)\log\frac{p(y)}{q(y)}dy$ (6)

An aggregation of DKL(p(y|x); p(y|x−j)) over all x in x space is suggested feature importance metric is given by eqn (7)

$S_D(j)=\int D_{KL}\left(p(y\mid x);p\left(y\mid x_{-j}\right)\right)p(x)dx.$ (7)

Reasoning behind calculating SD is straightforward: the more the DKL divergence between p(y|x) and p(y|x−j) throughout x space. As a result, training SVR d times, each with D−j for a distinct j, is necessary for assessments of SD(j), j = 1, · · ·, d. This procedure is obviously computationally costly. The fundamental principle behind RP process is to maintain values of all other features in D while jumbling values of jth feature.

It has introduced the Laplacian Score (LS), an unsupervised feature selection metric. Each feature is assessed by the LS according to how well it correlates with the graph Laplacian's leading eigenvectors. Quadratic form f T Lunf, where Lun is unnormalized graph Laplacian, is the fundamental component of the LS technique and determines the score of feature f by eqn (8)

$f^TL_{\text{un}}f=\sum_{i=1}^{n}\lambda_i\langle\mathbf{u}_i,f\rangle^2$ (8)

Score is lower when f has a greater projection on subspace of leading eigenvectors of Lun, where Lun = Pn i=1 λiuiu T i is the eigen-decomposition of Lun. Because they adhere to the graph structure, these traits might be regarded as "informative." The Laplacian's eigenvalues can be seen of as frequencies, and eigenvectors that correlate to higher Lun eigenvalues (or lower Ldiff eigenvalues) oscillate more quickly. By joining each point to its closest neighbors, create the graph Laplacian. The next step would be to look into the circumstances under which each point's neighbors fall into the appropriate cluster. Examine points x and y that are part of the same cluster. Hence, kx − yk 2 ∼ χ 2 d, where ui iid∼ N(0, 1), and (x − y) = (0, u1,..., ud). Likewise, if x and y are part of distinct clusters, then kx − yk 2 ∼ r 2 + χ 2 d, where r is distance in first coordinate between clusters. χ 2 measure-concentration bounds are used to determine conditions for n and d under which every point's neighbors are most likely to be members of same cluster. Eq. (9) initial weight matrix is defined as

$$S_{ij}=\begin{cases}e^{\left\{-\frac{d^2(x_i,x_j)}{\sigma_i\sigma_j}\right)} & \text{, if } x_i\in KNN(x_j)\vee x_j\in KNN(x_i)\\ 0, & \text{otherwise.}\end{cases}$$ (9)

Here, σ_i=d(x_i,x_i^K) is provided, where xK i stands for xi's KNN, and σ_j has a comparable situation. S-based L, D is updated to produce matrices using Eq (10)):

$$\widetilde{Z_A} = Z_A - \mathbf{1} \times \left(\frac{{Z_A}^T \times D \times \mathbf{1}}{\mathbf{1}^T \times D \times 1}\right)^T, \widetilde{Z_A}^T = \left(\widetilde{Z_A}\right)^T \quad (10)$$

where the sample matrix for feature subset A ⊆ F is ZA ∈ R(u) × |A|. Consequently, feature subset A's modified LS is defined as eqn (11)

$$J_{FILS}(A) = \begin{cases} \frac{trace}{\infty,} \left(\widetilde{Z_A} \times L \times \widetilde{Z_A}\right) & \text{if } A \neq \emptyset, \\ & \text{otherwise ,} \end{cases} \quad (11)$$

where the diagonal matrix elements' total is indicated by trace(·). The selection criterion is based on eqn (12)

$$f_{k^\circ} = \arg\min_{f_k \in \bar{A}} J_{FILS}(f_k) = \arg\min_{f_k \in \bar{A}} \left|1 - \frac{J_{FILS}(A \cup \{f_k\})}{J_{FILS}(F)}\right|$$

where

$$J_{FILS}(f_k) = \left|1 - \frac{J_{FILS}(A \cup \{f_k\})}{J_{FILS}(F)}\right| \quad (12)$$

As a result, in order to accurately reflect the elements' inherent relationships, it generally excludes similar elements. Furthermore, variation LS is concerned with feature subsets rather than individual features. As a result, it includes real interactions and local structures more thoroughly. In summary, the new LS is advantageous in unsupervised FS since it completely takes into account the mutual influences between elements and features to enhance the prior LS. MLP-NN, in which the numbers m, l, and s represent input, hidden, output nodes, respectively. Output of MLP-NN is calculated by eqn (14)

$$U_j = \sum_{i=1}^{m} \left(W_{ij} \cdot X_i\right) - \theta_j \cdot j = 1.2, \cdots l \quad (14)$$

Wij represents weight of edge from i-th node (input layer) to j-th node (hidden layer). Output of every hidden node is obtained as a eqn (15) to a sigmoid function.

$$U_j = \text{sigmoid}\left(U_j\right) = \frac{1}{\left(1 + \exp\left(-s_j\right)\right)} \cdot j = 1.2, \cdots h$$

$$Z_k = \sum_{j=1}^{l} \left(W_{jk} \cdot U_j\right) - \theta'_k \cdot k = 1.2. \cdots . s$$

$$Z_k = \text{sigmoid}(Z_k) = \frac{1}{\left(1+\exp(-Z_k)\right)}, k = 1.2. \cdots . s \quad (15)$$

3.2 Parameter optimization using ant binary swarm component encoder (ABSCEO) model

ABSCEO is very helpful for optimizing situations like intrusion detection and FS. A randomly generated set of potential solutions, represented as binary strings, is presented at start of ABSCEO. The algorithm uses the data gathered from previously determined solutions to modify the likelihood of incorporating each feature at each iteration. The PD in BACO acts as a guide for ants that come after, allowing them to determine whether to include a feature in their solution.

$$p_{i,j}^k(1) = \frac{\tau_{i,j}^\alpha(1) \cdot \eta_{i,j}^\beta(1)}{\tau_{i,j}^\alpha(0) \cdot \eta_{i,j}^\beta(0) + \tau_{i,j}^\alpha(1) \cdot \eta_{i,j}^\beta(1)}$$

$$p_{i,j}^k(0) = 1 - p_{i,j}^k(1) \quad (16)$$

The relative relevance of the PD and HF is controlled by the parameters α and β, respectively. As ants make their choices, the PD for each path first drops from its initial comparable value. Equations (17) define the update rule for PD from bit I to bit J.

$$\tau_{i,j}(1)(t+1) = (1-\rho) \cdot \tau_{i,j}(1)(t) + \Delta\tau$$

$$\tau_{i,j}(0)(t+1) = (1-\rho) \cdot \tau_{i,j}(0)(t) \quad (17)$$

Crossing the maintenance and sterile lines, which have the biggest fitness value differences, is part of the hybridisation process. If a newly created hybrid individual shows better fitness than the original, this procedure seeks to update the sterile line by replacing it. Following formula is utilized to find new individual inside the sterile line is given by eqn (18)

$$\begin{cases} X^d_{\text{new}(t)}(t) = r_1 \cdot X^d_s(t) + (1 - r_1) \cdot X^d_m(t), \\ m \in \{1,2,\dots,p\}; i,s \in \{2p+1, 2p+2, n\}, \end{cases} \quad (18)$$

where dth gene of ith hybrid individual of sterile line at iteration tth is indicated by X_("new " (i))d (t). A random number, r1, is produced from [0,1]. The goal of the selfing process is to guide individual towards globally optimal solution by updating the restorer line. The following equation can be used to mathematically model this behaviour is given by eqn (19)

$$\begin{cases} X^d_{\text{new }(i)}(t) = r_2\left(X^d_{\text{best}}(t) - X^d_{r(j)}(t)\right) + X^d_{r_{(0)}}(t), \\ i,j \in \{p+1, p+2, \dots, 2p\}, j \neq i, \end{cases} \quad (19)$$

Once hybridization and selfing have produced a new individual, it is assessed and contrasted with the initial candidate solution. Equation (20) defines the substitution process, which only substitutes the new individual for the old one if the new individual's fitness value is greater than the old one's.

$$X_i(t+1) = \begin{cases} X_{\text{new }(i)}(t), & \text{if } f\left(X_{\text{new }(i)}(t)\right) > f(X_i(t)), \\ X_i(t), & \text{otherwise.} \end{cases} \quad (20)$$

Utilizing a parameter known as SC (self-crossing), selfing process in HRO counts amount of iterations in which a restorer has not received updates. A reset is carried out on the restorer if their SC hits the upper limit, which is represented as tmax iterations without updates. The following equation precisely captures this reset behavior is given by eqn (21)

$$X^d_{r(i)}(t+1) = r_3\left(V^d_{\max} - V^d_{\min}\right) + X^d_{r(i)}(t) + V^d_{\min}, \quad (21)$$

The binary map used in this study is the sigmoid function, which is defined by eqn (22)

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$$

$$X^d_i = \begin{cases} 1, & \text{if sigmoid } \left(X^d_i\right) > \text{ rand}, \\ 0, & \text{otherwise.} \end{cases}$$

$$\sum_{j=1}^{J} b^2_{jq} = 1, \text{ for } q = 1, \dots, Q$$

$$\sum_{j=1}^{J} \left(b_{jq} b_{jr}\right)^2 = 0, \text{ for } q = 1, \dots, Q-1 \text{ and}$$

$$r = q+1, \dots, Q$$

$$\sum_{q=1}^{Q} b^2_{/q} > 0, \text{ for } j = 1, \dots, J \quad (22)$$

After N observations for x variables (θ1, θ2,... θx), matrix Z is determined. Each row depicts a numerical estimate of the dataset's sample data; column number n refers to the number of samples discovered using Equation (23).

$$Z = \begin{bmatrix} Z_{11} & Z_{12} & \cdots & Z_{1x} \\ Z_{21} & Z_{22} & \cdots & Z_{2x} \\ \vdots & \vdots & \ddots & \vdots \\ Z_{n1} & Z_{n2} & \cdots & Z_{nx} \end{bmatrix} \quad (23)$$

Step2: For observation matrix, centralize data process and evaluate both sample mean is given by eqn (24)

$$\overline{z_b} = \frac{1}{n}\sum_{a=1}^{n} z_{ab} \quad (24)$$

And standard deviation is given by eqn (25)

$$S_b = \sqrt{\frac{1}{n}(z_{ab} - \overline{z_b})^2}$$

$$\widehat{z_{ab}} = \frac{z_{ab} - \overline{z_b}}{s_b}(a = 1,2, \dots, n, b = 1,2, \dots, x) \quad (25)$$

Step3: To compute the sample correlation matrix, use method W=1/n Z^τ Z. Each element in W is computed according to Equation (26).

$$w_{ab} = \frac{\sum_{k=1}^{n}(z_{ka} - z_a)(z_{kb} - z_b)}{\sqrt{\sum_{k=1}^{n}(z_{ka} - z_a)^2 \sum_{k=1}^{n}(z_{kb} - z_b)^2}} \quad (26)$$

Step4: Calculate eigenvector and eigenvalue of W. Obtain x features values $\lambda 1 \geq \lambda 2 \geq \dots \geq \lambda x \geq 0$ for W is given by eqn (27)

$$r_a = \frac{\lambda_a}{\lambda_1 + \lambda_2 - + \lambda_x}(a = 1,2, \dots, x) \quad (27)$$

Evaluate associated eigenvectors e1, e2,..., ex. Choose top p feature vectors to form PCL is given by eqn (28)

$$L_{x \times p} = (e_1, e_2, \dots, e_p) \quad (28)$$

Step5: Create a linear transformation for real data using PCL matrix Lx×p and Equation (29), resulting in new principal component variables w1, w2, …, wp.

$$\begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix} = L_{x \times p}^T \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_p \end{bmatrix} \quad (29)$$

Matrix dimension reduced from x to p after linear transformation, considerably reducing the sample data storage.

4. Experimental analysis

This study addresses the topic of efficiently identifying feature selection strategies in modern data processing systems. As volume of data created by data systems grows, ML approaches are frequently used to extract useful data. However, due to their vast size, these datasets may operate slowly, necessitating the removal of redundant features during pre-processing. Feature selection is an important stage in current data processing systems, but it is a difficult operation due to complexity as well as amount of data.

Simulation setup-

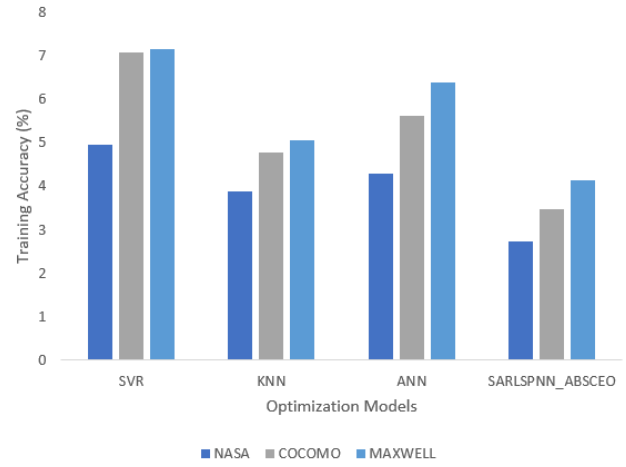
Version 3.1.1 of the Jikes Research Virtual Machine was altered for the studies in this study. An Intel x86-based system that supported two AMD Opteron 2216 dual core processors operating at 2.6GHz with an L1 and L2 cache and RAM of 128K, 1M, and 8GB, respectively, was used to run the virtual machine. Linux with kernel 2.6.32 was the machine's operating system. We employed Jikes RVM's FastAdaptiveGenMS configuration, which means that the generational mark-sweep garbage collector was used and the optimising compiler generated the core virtual machine at the most aggressive optimisation level.

Dataset description: DATASETS SELECTION the following publicly available datasets were used to compare the performance of seven nature-inspired meta-heuristic methods. These data sets were obtained from software engineering repository. • NASA: Dataset, which contains 93 projects, is described using 15 attributes. For many years, data about software developments has been documented from various NASA centres. It features fifteen effort multipliers and five scaling factors. • COCOMO 81: 15 attributes define the dataset of 63 projects. The COCOMO software cost method, which determines the amount of effort needed to develop a software project in calendar months, is the basis for the software project data in this repository. Additionally, effort multipliers with standard numerical values are included. • Maxwell: A promise repository provided the dataset. There are 62 projects in all, and each project has 27 qualities.

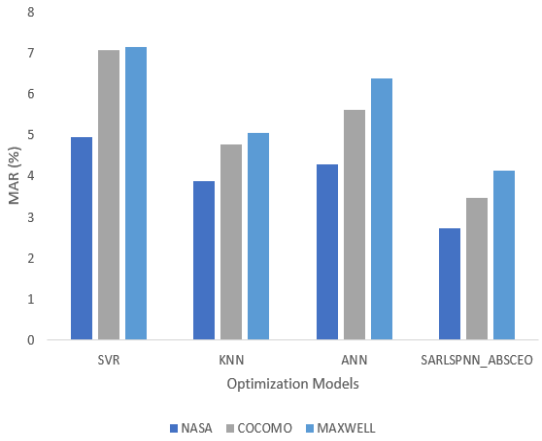
Table 1: Parameter evaluation using different datasets between proposed and existing model

Optimization models	NASA				COCOMO				MAXWELL			
	Training accuracy	MAR	MBRE	F-measure	Training accuracy	MAR	MBRE	F-measure	Training accuracy	MAR	MBRE	F-measure
SVR	7.89	4.95	6.39	6.59	7.51	7.07	9.12	6.75	8.12	7.15	9.23	6.28
KNN	8.12	3.87	5.00	7.12	8.65	4.78	6.16	7.84	8.56	5.04	6.51	7.89
ANN	8.75	4.28	5.53	8.15	9.45	5.61	7.24	8.56	9.52	6.38	8.23	8.25
SARLSPNN_ABSCEO	9.95	2.72	3.51	9.12	9.88	3.47	4.47	9.12	9.89	4.12	5.31	8.95

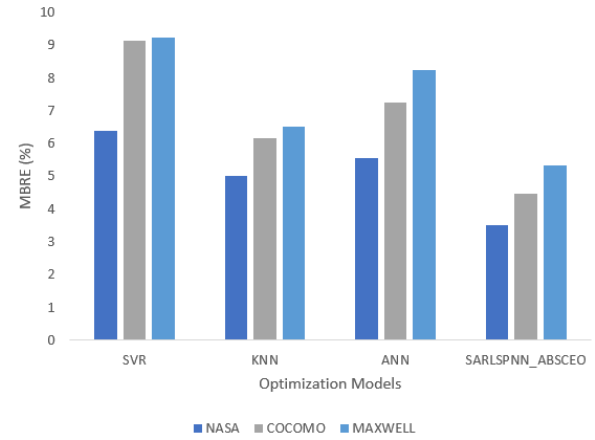
Various parameters are selected for the estimation of time and effort for these features. The input layer of the deep neural network receives the descriptions of 15 common variables listed in Table 1. Since this is supervised machine learning, the output layer must provide a value as the result value that reflects the total amount of work needed to produce the software product. Preprocessing this data is an essential step. All of the numbers in each of this dataset's columns should be scaled to between 0 and 1 in order to train the deep neural network. This is because neural network training will be ineffective if the numbers in one column are large while the numbers in the other column are small. Using the MinMaxScaler object from the well-known scikit-learn package is one of the best ways to achieve this. It was created especially for this purpose. This method requires us to create a new MinMaxScaler first, after, which just pass in a feature range argument specifying that all numbers should be scaled from 0 to 1.



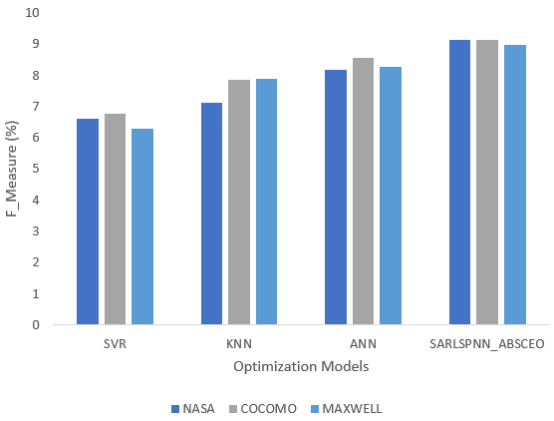
Training accuracy



MAR (Mean absolute residual)



Mean balanced relative error (MBRE)



F-measure

Figure 3. Comparative analysis for various datasets in terms of training accuracy, MAR (Mean absolute residual), Mean balanced relative error (MBRE), F-measure

With six inputs listed in Table 2 and an output (estimated effort), proposed SARLSPNN_ABSCEO technique has been designed with following features: friedman_mse function to measure quality of a split, squared error to optimize loss function, value of 200 for number of boosting stages to perform, value of 3 as less number of samples needed to split an internal node. Create training as well as test sets for each project using a 50:50 split and stratified sampling. To be more precise, choose 50% of the cases at random to be training set and remaining 50% to be test set using stratified sampling. In accordance with real application scenario, the stratified sampling technique ensures that the training and test sets have the identical defect rates. Such a sampling environment also aids in lowering sample biases. In earlier studies on defect prediction, stratified sampling and the 50:50 split were frequently employed. Repeat this method 30 times on every project by rearranging model order in order to lessen effect of random division treatment on experimental data as well as provide a general conclusion. Thus, run 30 times for every combination of parameters and note the average indicator values. Lastly, the best average F-measure value determines the ideal combination of parameters ω and q. The average values of the four indicators for 30-round experiments are presented in this study.

Table 2: Test results for SARLSPNN_ABSCEO method.

Metrics	**Datasets**	**Metric values**			**Parameters**		**Estimated effort**		
		Minim	**Maxim**	**Mean**	**d**	**t**	**Minim**	**Maxim**	**Mean**
Training accuracy	**NASA**	7.015	19.525	10.259	7	150	9.568	99.525	69.568
	COCOMO	83.459	139.798	156.321	8	200	189.26	139.798	189.26
	MAXWELL	256.325	2442.569	289.654	9	250	159.639	2442.569	159.639
MAR	**NASA**	4.965	9.568	5.458	6	150	19.525	99.569	79.568
	COCOMO	75.259	189.26	99.569	9	200	139.798	67.258	189.26
	MAXWELL	45.956	159.639	67.258	8	250	2442.569	25.369	159.639
MBRE	**NASA**	7.895	33.596	25.369	9	150	7.895	97.015	65.458

	COCOMO	89.652	139.569	521.468	6	200	89.652	83.459	99.569
	MAXWELL	499.589	5589.63	3558.26	8	250	499.589	256.325	67.258
F-measure	**NASA**	0.625	0.695	0.3256	9	150	7.015	97.895	80.259
	COCOMO	0.596	0.459	0.459	6	200	83.459	89.652	156.321
	MAXWELL	0.825	0.845	0.559	7	250	256.325	499.589	289.654

A graphical depiction of the software effort predicted by the optimized SARLSPNN_ABSCEO method in the case of the metrics' minimum values in relation to the actual effort unique to each of the three datasets is presented in Figure 4. In contemporary AI research, model interpretation is becoming more and more significant. It is frequently possible to identify any unintentional biases in data and gain a deeper knowledge of the issue at hand by knowing the components and how much of an impact they have on decisions. For simpler models, these interpretations are rather simple, and there are a number of new methods for doing so. However, it may be more challenging to analyses more complex models or models that use a multi-tier structure.

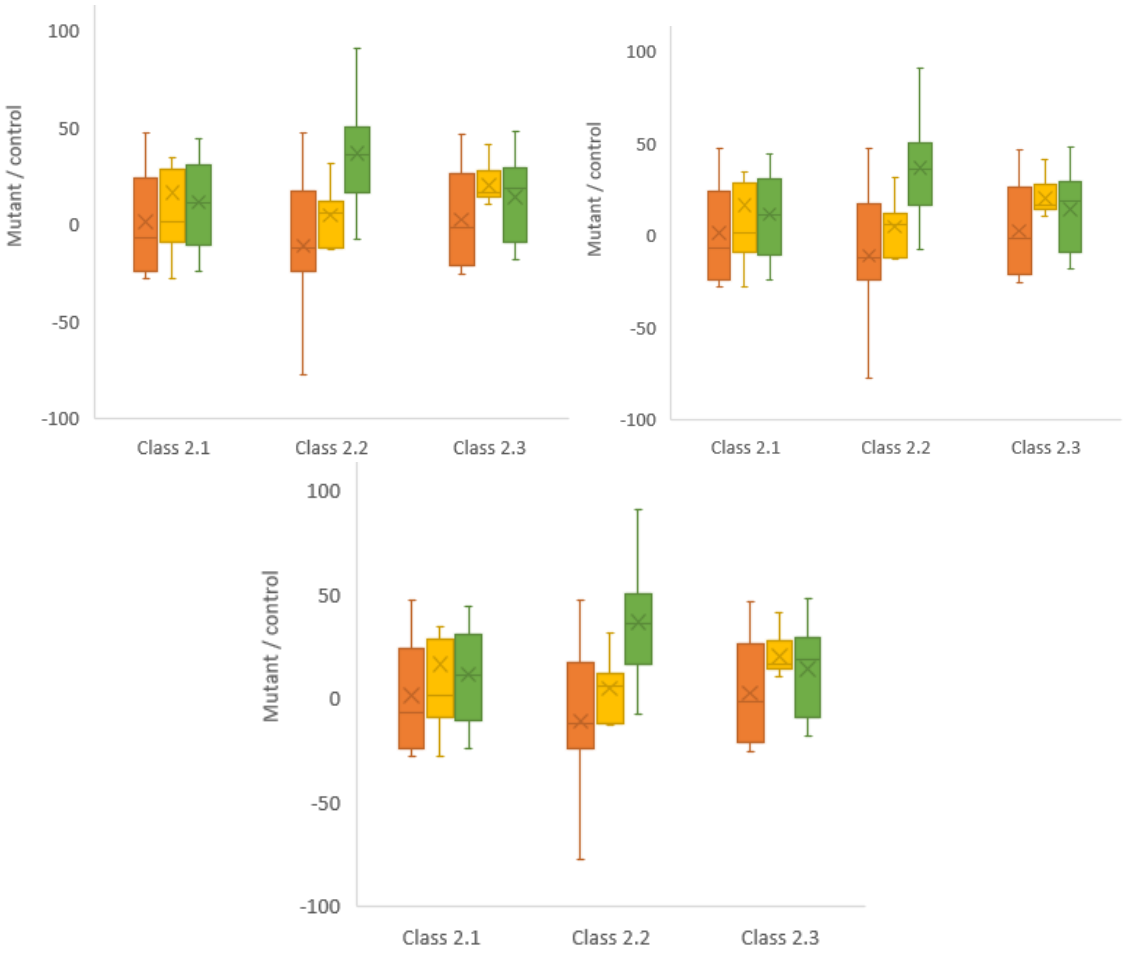


Figure 4. Effort estimated by optimized SARLSPNN_ABSCEO technique compared to real effort. (a) NASA, dataset; (b) COCOMO dataset; (c) Maxwell dataset.

Verifying the statistical significance of results is crucial when comparing methods that depend on randomness, like metaheuristics, because a single simulation could not be reflective of overall performance. In light of this, 30 separate independent runs of the simulations were conducted in this work, and data was gathered for additional research. Parametric as well as non-parametric tests are two categories of statistical tests that are used to confirm statistical significance of results. Therefore, determining whether parametric tests are justified or whether non-parametric analysis is appropriate is the first step. If the requirements of independence, normalcy, and homoscedasticity are satisfied, then the use of parametric testing is warranted. Furthermore, for the independence criteria to be satisfied, simulations have to be run with independent random seeds. Levene's test, which verifies homoscedasticity, produced p-values of 0.68 for every instance, suggesting that this requirement is also satisfied. The Shapiro-Wilk test is used to evaluate the final condition, normality, and the comparative analysis includes the p-values calculated for each approach. The null hypothesis (H0), which suggests that results do not derive from normal distributions, may be rejected because all of the values fall below a threshold of 0.05. The difficulties in detecting software defects can be better understood by comprehending the significance of features in model judgements. Additionally, identifying elements that are crucial to these classifications can help lower computational costs of methods during deployment as well as improve data gathering going forward. Enforcing confidence in model decisions and enhancing their generalizability and impartiality also depend on identifying hidden model biases.

5. Conclusion

Using a Meta heuristic machine learning model, this study suggests a novel approach to feature weight model-based optimization in software development. Here, support additive regression is used to determine the feature weight and feature selection for the software model. Perceptron neural network with Laplacian score. The Ant Binary Swarm Component Encoder Optimization Method is then utilized to optimize software method specifications. Findings demonstrate that each learner and algorithm has a different attitude towards the data, and that the performance of algorithms varies depending on the data. Used all of Feature Select's algorithms and learners on the input dataset using the multi-objective score function in a second extensive experiment. The optimal choice of initial weights and pertinent characteristics for model development has a significant impact on the SARLSPNN_ABSCEO architecture's performance. SARLSPNN_ABSCEO can be employed with meta-heuristic algorithms to minimize significant training delays since they enable the discovery of an optimal solution. The simulations have also shown that the suggested approach performs better than several current software effort estimating techniques that have been published in the literature. We demonstrate up to 20% improvements in total execution time. To the best of our knowledge, this research is the first to show that optimisation orders for compiler methods can be effectively selected using machine-learning models. To get results that are more thorough in future, refine estimation methods by experimenting with novel techniques as well as utilizing cloud computing for evaluating.

References

[1] K. Zhu, S. Ying, N. Zhang, and D. Zhu, "Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network," *Journal of Systems and Software*, vol. 180, p. 111026, 2021.

[2] M. Shahhosseini, G. Hu, and H. Pham, "Optimizing ensemble weights and hyperparameters of machine learning models for regression problems," *Machine Learning with Applications*, vol. 7, p. 100251, 2022.

[3] K. Li, A. Zhu, P. Zhao, J. Song, and J. Liu, "Utilizing deep learning to optimize software development processes," *arXiv preprint* arXiv:2404.13630, 2024.

[4] G. Stenzel, K. Schmid, M. Kölle, P. Altmann, M. Lingsch-Rosenfeld, M. Zorn, and L. Belzner, "SEGym: optimizing large language model assisted software engineering agents with reinforcement learning," in *International Conference on Bridging the Gap between AI and Reality*, Cham, Switzerland, 2024, pp. 107-124.

[5] H. L. T. K. Nhung, V. Van Hai, P. Silhavy, Z. Prokopova, and R. Silhavy, "Incorporating statistical and machine learning techniques into the optimization of correction factors for software development effort estimation," *Journal of Software: Evolution and Process*, vol. 36, no. 5, p. e2611, 2024.

[6] S. Talukdar, S. Bera, M. W. Naikoo, G. V. Ramana, S. Mallik, P. A. Kumar, and A. Rahman, "Optimisation and interpretation of machine and deep learning models for improved water quality management in Lake Loktak," *Journal of Environmental Management*, vol. 351, p. 119866, 2024.

[7] C. X. Do, N. T. Luu, and P. T. L. Nguyen, "Optimizing software vulnerability detection using RoBERTa and machine learning," *Automated Software Engineering*, vol. 31, no. 2, p. 40, 2024.

[8] D. B. Raven, Y. Chikkula, K. M. Patel, A. H. Al Ghazal, H. S. Salloum, A. S. Bakhurji, and R. S. Patwardhan, "Machine learning & conventional approaches to process control & optimization: Industrial applications & perspectives," *Computers & Chemical Engineering*, vol. 189, p. 108789, 2024.

[9] R. Manoharan, "Improving security and performance in chaotic optical communication via real-time pilot signal processing techniques," *IETE Journal of Research*, pp. 1-9, 2025.

[10] M. Rajesh, S. Ramachandran, K. Vengatesan, S. S. Dhanabalan, and S. K. Nataraj, "Federated learning for personalized recommendation in securing power traces in smart grid systems," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 1, pp. 88-95, Feb. 2024, doi: 10.1109/TCE.2024.3368087.

[11] M. Yazdani, S. Shahriari, and M. Haghani, "Real-time decision support model for logistics of emergency patient transfers from hospitals via an integrated optimisation and machine learning approach," *Progress in Disaster Science*, vol. 25, p. 100397, 2025.

[12] M. Ding, Y. Guo, Z. Huang, B. Lin, and H. Luo, "GROM: A generalized routing optimization method with graph neural network and deep reinforcement learning," *Journal of Network and Computer Applications*, vol. 229, p. 103927, 2024.

[13] M. Hassanali, M. Soltanaghaei, T. Javdani Gandomani, and F. Zamani Boroujeni, "Software development effort estimation using boosting algorithms and automatic tuning of hyperparameters with Optuna," *Journal of Software: Evolution and Process*, vol. 36, no. 9, p. e2665, 2024.

[14] C. Do Xuan, T. T. Luong, and M. C. Thanh, "Optimising source code vulnerability detection using deep learning and deep graph network," *Connection Science*, vol. 37, no. 1, p. 2447373, 2025.

[15] R. Siva, K. S., B. Hariharan, and N. Premkumar, "Automatic software bug prediction using adaptive golden eagle optimizer with deep learning," *Multimedia Tools and Applications*, vol. 83, no. 1, pp. 1261-1281, 2024.

[16] J. Huang, W. M. Ashraf, T. Ansar, M. M. Abbas, M. Tlija, Y. Tang, and W. Zhang, "Optimisation led energy-efficient arsenite and arsenate adsorption on various materials with machine learning," *Water Research*, vol. 271, p. 122815, 2025.

[17] J. Sarwar, S. A. Khan, M. Azmat, and F. Khan, "A comparative analysis of feature selection models for spatial analysis of floods using hybrid metaheuristic and machine learning models," *Environmental Science and Pollution Research*, vol. 31, no. 23, pp. 33495-33514, 2024.

[18] A. Avar and E. Ghanbari, "Optimal integration and planning of PV and wind renewable energy sources into distribution networks using the hybrid model of analytical techniques and metaheuristic algorithms: A deep learning-based approach," *Computers and Electrical Engineering*, vol. 117, p. 109280, 2024.

[19] B. Arasteh, K. Arasteh, A. Ghaffari, and R. Ghanbarzadeh, "A new binary chaos-based metaheuristic algorithm for software defect prediction," *Cluster Computing*, vol. 27, no. 7, pp. 10093-10123, 2024.

[20] R. M. Adnan, A. Mirboluki, M. Mehraein, A. Malik, S. Heddam, and O. Kisi, "Improved prediction of monthly streamflow in a mountainous region by Metaheuristic-Enhanced deep learning and machine learning models using hydroclimatic data," *Theoretical and Applied Climatology*, vol. 155, no. 1, pp. 205-228, 2024.

[21] Y. Sun, H. L. Dai, H. Moayedi, B. N. Le, and R. M. Adnan, "Predicting steady-state biogas production from waste using advanced machine learning-metaheuristic approaches," *Fuel*, vol. 355, p. 129493, 2024.

[22] J. Chen, W. Xiao, H. Zhang, J. Zuo, and X. Li, "Dynamic routing optimization in software-defined networking based on a metaheuristic algorithm," *Journal of Cloud Computing*, vol. 13, no. 1, p. 41, 2024.

[23] K. M. Hamdia, X. Zhuang, and T. Rabczuk, "An efficient optimization approach for designing machine learning models based on genetic algorithm," *Neural Computing and Applications*, vol. 33, no. 6, pp. 1923-1933, 2021.