Parameter Identification of a Multilayer Perceptron Neural Network using an Optimized Salp Swarm Algorithm

Mohamad Al-Laham[1], Salwani Abdullah[2], Mohammad Atwah Al-Ma'aitah[3],
Mohammed Azmi Al-Betar[4], Sofian Kassaymeh[5], Ahmad Azzazi[6]
MIS Department, Amman University College, Al-Balqa Applied University, Amman, Jordan[1]
DM and Optimization Research Group, Center for Artificial Intelligence Technology,
Universiti Kebangsaan Malaysia, Bangi Selangor, Malaysia[2]
MIS Department, Amman University College, Al-Balqa Applied University, Amman, Jordan[3]
Artificial Intelligence Research Center, College of Engineering and IT,
Ajman University, Ajman, United Arab Emirates[4]
Software Engineering Department, Faculty of IT, Aqaba University of Technology, Aqaba, Jordan[5]
Computer Science Department, Faculty of IT, Applied Science Private University, Amman, Jordan[6]

***Abstract*—Effort estimation in software development (SEE) is a crucial concern within the software engineering domain, as it directly impacts cost estimation, scheduling, staffing, planning, and resource allocation accuracy. In this scientific article, the authors aim to tackle this issue by integrating machine learning (ML) techniques with metaheuristic algorithms in order to raise prediction accuracy. For this purpose, they employ a multilayer perceptron neural network (MLP) to perform the estimation for SEE. Unfortunately, the MLP network has numerous drawbacks as well, including weight dependency, rapid convergence, and accuracy limits. To address these issues, the SSA Algorithm is employed to optimize the MLP weights and biases. Simultaneously, the SSA algorithm has shortcomings in some aspects of the search mechanisms as well, such as rapid convergence and being susceptible to the local optimal trap. As a result, the genetic algorithm (GA) is utilized to address these shortcomings through fine-tuning its parameters. The main objective is to develop a robust and reliable prediction model that can handle a wide range of SEE problems. The developed techniques are tested on twelve benchmark SEE datasets to evaluate their performance. Furthermore, a comparative analysis with state-of-the-art methods is conducted to further validate the effectiveness of the developed techniques. The findings demonstrate that the developed techniques surpass all other methods in all benchmark problems, affirming their superiority.**

***Keywords*—Software development effort estimation; machine learning; multilayer perceptron neural network; salp swarm algorithm; genetic algorithm**

I. Introduction

Software engineering encompasses a systematic, methodical, and quantitative approach to the creation, operation, and maintenance of software systems. In order to ensure the effective and timely production of software products, software engineering management adopts specific actions such as planning, monitoring, measuring, and reporting [1], [2]. Conversely, software development effort estimation (SEE) represents a formidable challenge that holds significant importance in the software development process. SEE can be defined as the process of constructing a model that aids software engineers in determining the cost of a software project prior to or at the commencement of the software development process [3], [4].

The scholarly literature proposes various methodologies to tackle the SEE problem. Some of these methodologies fall under the non-soft computing category, while others utilize machine learning (ML) techniques. ML techniques have demonstrated their effectiveness and capability to address similar problems encountered in diverse engineering fields. Among these ML techniques, artificial neural networks (ANN) have gained popularity and been extensively adopted in numerous research studies. One common type of ANN is the Multilayer Perceptron Neural Network (MLP), which is widely employed in addressing classification and prediction problems. Additionally, MLP exhibits excellent capability for handling non-linear and complex engineering problems [2], [3].

Motivated by the shortage of accuracy in the available models for estimating software development efforts in the literature, this research study aims to develop a robust and reliable ML model that has the ability to address the problem with high accuracy. However, to overcome limitations in prediction accuracy in the MLP network, the study integrates a metaheuristic algorithm known as the Salp Swarm Algorithm (SSA) into the MLP network. This integration aims to optimize the weights and biases of the MLP network, thereby enhancing its prediction accuracy. Furthermore, considering the SSA algorithm's search restrictions, the research suggests a strategy for using the Genetic Algorithm (GA) to fine-tune the SSA parameters.

Because the optimization procedure is stochastic, the effectiveness of metaheuristic algorithms essentially rests on finding a reasonable balance throughout the exploration and exploitation phases and refining the solutions over generations. The algorithm investigates the search space broadly during the exploration phase in an effort to avoid becoming stuck in local optima. Where, in the exploitation phase, the promising solutions found during the exploration phase are refined to attain the global optimum [5].

The Salp swarm algorithm (SSA) was put out in 2017 by Mirjalili [6] as an effective way for solving optimization issues in the context of metaheuristic methodologies. The swarming behavior of salps in deep waters, which commonly generates a chain of salps, served as the basis for this method. In order to have more control and make coordinated adjustments for quick foraging, this chain advances by pushing water inside its barrel-shaped shells.

The key drawbacks of the SSA algorithm, like those of other metaheuristic algorithms, are delayed convergence and premature convergence toward local optima. The No Free Lunch theorem in optimization states that it must be altered in order to address certain particular problems. This theorem argues that when addressing all optimization issues, all methods function on average equally well. As a result, a particular approach may work effectively for one group of problems but fail miserably for another. In order to enhance algorithms that are acceptable for the majority of issue types, researchers [7], [8], [9] determined that the proper balance between exploration and exploitation needed to be improved.

There are potential benefits to fine-tuning the SSA algorithm parameter settings using the GA algorithm, thus optimizing the MLP network weights. By integrating the SSA's exploration skills with the GA's global search, the SSA optimization capabilities are improved. Therefore, by overcoming local optimum conditions and responding to various scenarios, the developed model becomes adaptable and flexible. The MLP network's estimation error can be reduced, which in turn improves the accuracy of resource planning and project scheduling. The SSA-GA approach makes it easier to facilitate generalization and produce accurate estimates for varied software projects. Overall, this method effectively addresses the issue of estimating the effort required for software development, allowing for good resource management and project planning. The key contributions of this work are as follows:

1) Use the MLP network to address the issue of estimating software development effort.
2) Use the SSA algorithm to optimize the MLP parameters (Weights and biases).
3) Apply the GA algorithm to boost the SSA algorithm's optimization capabilities by fine-tuning its parameters.
4) Use several common benchmark SEE datasets to generalize the findings.
5) Develop three methods (e.g., MLP, SSA-MLP, and SSA-GA) and conduct statistical comparisons among the methods to determine the most effective one.

The rest of the paper contains the following: Section II provides a brief overview of the SEE problem, the MLP network, and the SSA algorithm. Section III introduces the developed methods. Section IV introduces the research results. Section V introduces a discussion for study finding. Section VI introduces the study conclusion.

II. Background

The research at hand encompasses a variety of tools and topics, including "Software Development Effort Estimation Problem", "Multilayer Perceptron Neural Network", and "Salp Swarm Algorithm." These components form the foundation of the study and contribute to its overall context and objectives. Below is a brief overview of each of them.

A. Software Development Effort Estimation Problem

Software development effort estimation (SEE) is a critical procedure that involves utilizing uncertain, noisy, inconsistent, and incomplete data inputs to forecast the optimal and realistic amount of effort required for software development and maintenance. Typically, the level of work accomplished is expressed in units such as man-months, man-hours, or the number of individuals involved in the software development process. Accurate estimations play a pivotal role in effectively planning for software project development. However, underestimation and overestimation are two complex issues that software project managers often face, and these challenges can potentially result in project failure [10], [11].

Robert N. Charette [12] extensively discussed the primary causes of failure in software projects, identifying a range of issues that contribute to project failures. These issues encompass unending system requirements, inadequate communication between developers and customers, the utilization of outdated technologies, ineffective project management practices, commercial pressures, difficulties in handling project complexity, inaccurate project status reports, unrealistic project objectives, uncontrolled risks, and stakeholder conflicts influenced by political factors. However, the ultimate success of a software project heavily relies on the accuracy of work estimation. While precise estimation is essential for both project managers and clients, there is a pressing need to enhance software development effort estimation (SEE). SEE plays a crucial role in supporting efficient software development and maintenance for software developers, while also empowering clients to negotiate contracts, plan project completion timelines, and establish release dates for prototypes, among other aspects. Despite the existence of numerous approaches to software effort estimation, the development of accurate and consistent estimation techniques remains challenging for researchers [13], [1].

B. Multilayer Perceptron Neural Network

The multilayer perceptron (MLP) neural network, a specific component of the feedforward neural network (FFNN), stands as a unique form of Artificial Neural Network (ANN) capable of effectively approximating and comprehending the characteristics exhibited by computational models [14].

The MLP neural network necessitates a unidirectional arrangement of neurons, where data is transmitted through stacked layers organized into three types: input, hidden, and output layers. The connections between these layers can be established using different weights. Neurons within the MLP perform calculations using summation and activation functions. Summation function responsible for calculating the weighted sum of inputs and their corresponding connection weights. This function aggregates the inputs and weights to generate a weighted sum. The activation function, on the other hand, introduces non-linearity to the output of the summation function. It determines the output of a neuron or an entire layer based on the weighted sum calculated by the summation function. The activation function introduces non-linear transformations,

allowing the network to learn and model complex relationships between inputs and outputs.

The hidden layer neurons in the neural network employ the sigmoid activation function, while the output layer neurons utilize the linear activation function. The linear and sigmoid functions can be mathematically represented by Eq. 1 and 2, respectively.

$$f(x) = x \tag{1}$$

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

Consequently, by adjusting the biases and weights, the network iteratively minimizes the error in the output and improves the accuracy of predictions. Graphical representations of the linear and sigmoid functions can be observed in Fig. 1 and 2, respectively.

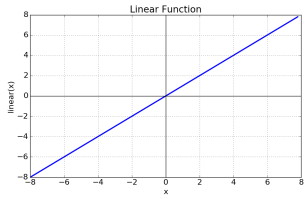

Fig. 1. Linear Activation Function

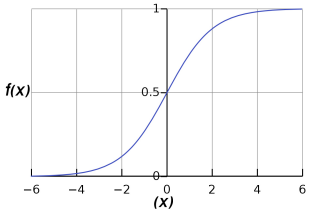

Fig. 2. Sigmoid Activation Function

C. Salp Swarm Algorithm

The Salp Swarm Algorithm (SSA) is a recently introduced metaheuristic optimization algorithm developed by Mirjalili et al. in 2017 [6]. This algorithm is inspired by the collective behavior of sea salps found in nature. Sea salps, which are transparent, barrel-shaped invertebrates resembling jellyfish, propel themselves forward by pumping water into the back of their shells [15], [16]. They live and swim together in groups, with one salp acting as the leader while the others are considered "followers" [6]. Fig. 3 illustrates the body shape of a salp in (a) and depicts a group of salp swarms in (b).

The collective swimming behavior of salps in a group or swarm can be mathematically formulated and modeled as an optimization algorithm. The positions of the salps are determined within a search space of dimensions $n \times N$, where n represents the number of variables in a specific problem and N corresponds to the number of solutions in the population. This search space encompasses a food source (F) that represents the desired target or optimal solution for the salps. The leader of the swarm updates its position using Eq. 3.

$$x_j^1 = \begin{cases} F_j + r_1 * ((ub_j - lb_j) * r_2 + lb_j) & r_3 \geq 0.5 \\ F_j - r_1 * ((ub_j - lb_j) * r_2 + lb_j) & r_3 < 0.5 \end{cases} \tag{3}$$

In the provided equations, x_j^1 represents the position of the swarm leader in the j^{th} dimension, F_j represents the position of the food source in the same dimension. The variables r_1, r_2, and r_3 correspond to three random numbers, while lb_j and ub_j represent the lower and upper boundaries of the search space in the j^{th} dimension, respectively.

The movement of the swarm leader is determined by the position of the food source F in the search space, as indicated in Eq. 3. The value of r_1 is used to achieve a balance between exploration and exploitation during the search process, and its formulation is given in Eq. 4.

$$r_1 = 2 * e^{-\left(\frac{4*l}{L}\right)^2} \tag{4}$$

where L represents the maximum number of iterations and l represents the current one.

The values of r_2 and r_3 are randomly generated within the range of 0 to 1. These values play a significant role in determining the magnitude of the movement step taken by the salps and influencing the direction of the search, whether it is positive or negative. Consequently, the position of the salps can be updated using the following expression:

$$x_j^{i+1} = \frac{1}{2}\left(x_j^i + x_j^{i-1}\right), \quad i \geq 2 \tag{5}$$

where x_j^i is the position of the i^{th} follower.

The SSA's optimization process starts with the population's random creation of solutions. The followers then start to update their locations, led by the leader's location, in an effort to find better locations with greater fitness values. Up until the termination condition is satisfied, which signifies the conclusion of the optimization process, these phases are repeated repeatedly.

III. Developed Technique

The method that has been developed combines an MLP network with SSA and is called "SSA-MLP." This integration's main goal is to use SSA to identify the MLP network's optimal weights and biases, thereby improving the accuracy of MLP predictions. The Genetic Algorithm (GA) is used to upgrade the SSA algorithm to achieve this modification. The goal of this upgrade is to fine-tune SSA's settings to increase its capacity for optimization. The GA algorithm is used in each iteration of the SSA algorithm to search for the most appropriate values for the SSA parameters, ensuring their optimal setting, which leads to the creation of a new technology called "SSA-GA". By applying GA iteratively, it fine-tunes the parameters of SSA, leading to enhanced optimization performance. The working steps of the proposed SSA-GA algorithm may be summed up as follows:

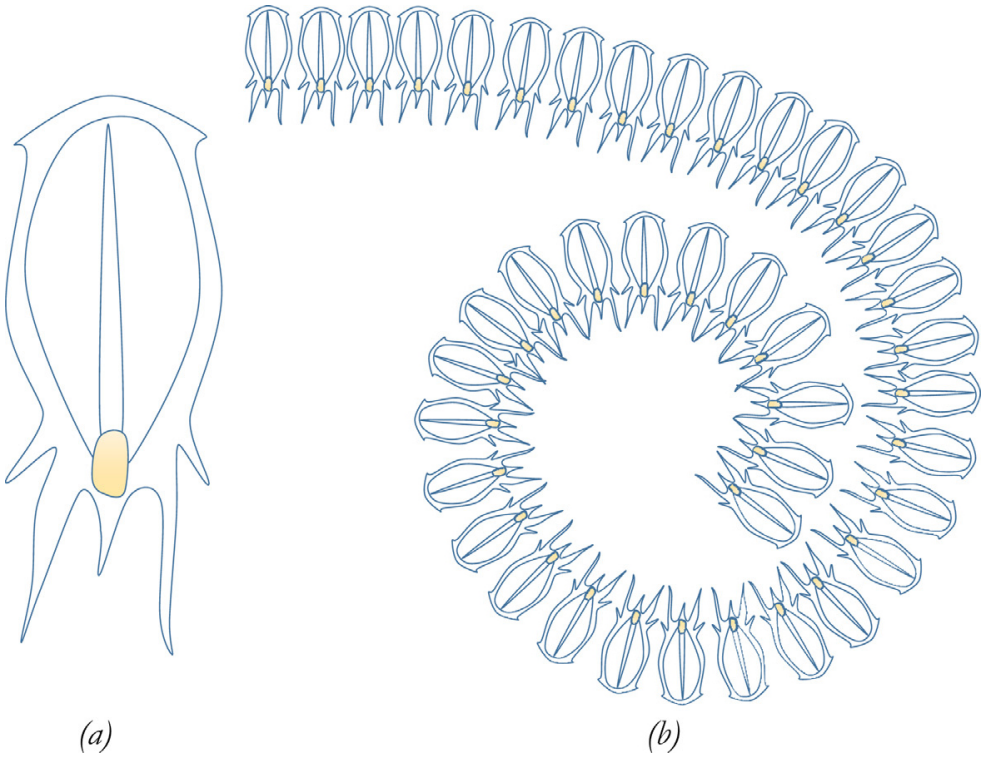

Fig. 3. Salp and Salp Swarm

- *MLP Initialization*: The Multilayer Perceptron (MLP) network's initial state is determined during initialization, and this procedure also provides the groundwork for further training and optimization.
  The weights and biases of the network are chosen at random during MLP startup. The biases serve as adjustable factors that regulate the activation thresholds of individual neurons, while the weights demonstrate the strength of connections between the neurons in the MLP's various layers. The MLP starts with a diverse collection of values thanks to the random initialization of these parameters, allowing for the exploration of many solutions during the training process.
  In MLP networks, random initialization is desirable because it lessens biases toward particular patterns or structures that may be present in the training data. The MLP is encouraged to learn a variety of characteristics and patterns by adding randomness, which helps it generalize well to data that has not yet been seen.
  In addition, initializing the MLP network with random weights and biases stimulates variation among its neurons and allows them to contribute to learning on their own. This variant prevents the network from becoming stale in local optima and enables it to explore diverse regions of the solution space.
- *SSA-Population Initialization*: The approach starts with the SSA-Population Initialization step by running the Multilayer Perceptron (MLP) training phase using the given training data. The number of times this training phase is repeated equals the SSA-population size, or the number of solutions in the population, exactly.

The MLP's updated weights and biases are retrieved and arranged as a vector after each training phase iteration. This vector form includes the precise set of adjusted weights and biases from the training procedure. The SSA-population is then updated with these vectors, which reflect the altered weights and biases. The SSA method treats each vector as an independent solution.
The next step is to evaluate the fitness of each solution inside the population once all the upgraded weights and biases vectors have been introduced to the SSA-population. Based on their individual training-fitness values, which represent how well each solution does in terms of minimizing errors throughout the training phase, an evaluation is made. The training-fitness values provide each solution with a numerical evaluation of its fitness or quality.
The optimal solution within the SSA-population is identified in this assessment by finding the one with the best fitness value. The weights and biases used in this study's best solution are those that produce the most accurate estimates while minimizing the mean squared error (MSE), which acts as the study's objective function. The mean squared difference between the anticipated values and the actual values is quantified by the MSE, an extensively used statistic in machine learning.
The SSA-Population Initialization stage makes it easier to identify the best solution within the population by using the MSE as the objective function. This allows for later optimization and enhancement of the MLP's performance in the estimation assignment.

- *GA-Population Initialization*: The GA-population is generated at random during the population initialization stage. The relevant parameters, indicated as p_1 and p_2, have discrete values between 1 and 15. The GA-population is formulated as a two-dimensional matrix, where each row represents one GA solution. The number of SSA parameters that need to be tweaked is indicated by the size of each row, which is $d = 2$. Two parameters, p_1 and p_2, make up each GA-solution and contribute to the set of accessible parameters shown in Eq. 6.

$$r_1 = p_1.e^{-\left(\frac{p_2 l}{L}\right)^2} \tag{6}$$

- *GA-Population Evaluation*: Each GA-solution's fitness value is evaluated in order to ascertain it. This evaluation entails running an SSA algorithm optimization process and introducing each GA solution into the SSA-population. The application of each SSA-solution to the MLP, along with the validation data, aims to determine the optimal SSA-solution. The resulting validation-fitness is then calculated.
  The SSA algorithm is combined with each GA solution to assess it through run its optimization process. The goal of the optimization is to find the SSA-solution that, when combined with the MLP, produces the best results on the validation data.
  The validation-fitness is calculated using the MLP and the chosen SSA-solution. Based on the results achieved during the MLP validation phase utilizing the validation data, this validation-fitness measures the caliber or efficacy of the GA solution.
  By evaluating each GA-solution's performance when it is incorporated into the SSA algorithm and then assessing the validation-fitness attained by the corresponding SSA-solution when combined with the MLP and the validation data, the fitness of each GA-solution is thus determined through this GA-population evaluation step.

- *Parameter Tuning*: The GA-solutions go through mutation and recombination after being encoded into chromosomes. Superior individuals within the population develop as a result of these genetic activities, which alter the encoded parameter values. The favorable parameter values that these superior GA-solutions possess boost their chances of surviving, reproducing, and passing on these enhanced parameter values to their progeny.

- *Selection*: The roulette wheel selection strategy is used in this phase to choose the GA-solution from the population. The fitness function for each solution is calculated using the standard SSA technique in this selection scheme. To do this, the solution is used as an input parameter for the SSA algorithm, and the fitness value that results is taken into account as the objective function value for that specific solution. The roulette wheel selection system ensures that the fittest individuals have a greater chance of being picked for continued breeding and development by favoring individuals with higher fitness ratings.

- *Encoding*: All GA-solutions or individuals are restructured in this stage using a binary format. The solutions are transformed by utilizing binary notation to express them. Each solution is recast in a standardized representation using the binary format, making it easier for genetic processes like crossover and mutation to take place in later iterations of the algorithm. The binary encoding makes it easy to manipulate and modify the GA-solutions, allowing the evolutionary process to explore various genetic material combinations.

- *Crossover*: The crossover operation is performed on the chosen solution in this stage, where two encoded parents are picked at random, the same as in the Selection-Step. Both the single crossover and double crossover approaches were used for this study. The chance of using the crossover operation is determined by the crossover rate, which is defined as $gamma_r$, where $gamma_r$ is a number produced at random between [0, 1].
  The amount of crossover applied throughout the population is influenced by the $gamma_r$ value. A major fraction of the population will experience crossover when $gamma_r$ is closer to 1, leading to a significant inheritance of genetic material across people. This suggests that during crossover, several genes will be transferred across individuals.
  The value of $gamma_r$ is set at 70% for the provided technique, suggesting a comparatively high crossover rate. This decision guarantees that the population experiences a sizable quantity of genetic recombination, enabling the evolutionary process to explore various genetic combinations.
  To prevent similarity or uniformity among the solutions, alterations are made to the chromosomes in the mutation stage. This is accomplished by altering one or more chromosomal genes; the mutation rate (mu_r) determines the degree of mutation. In order to ensure regulated exploration of the search space and prevent excessive and disruptive modifications to the solutions, the mu_r is often given a minimal value.
  The mu_r is set to 0.1 for the proposed approach, which is a quite low mutation rate. In order to promote a certain degree of diversity and exploration among the population, this choice permits modest alterations to the genetic code.

- *Decoding*: The chromosomes' binary representation is converted into a decimal format during the decoding process. The binary-encoded chromosomes are translated into their corresponding decimal values throughout this procedure. The genetic data contained inside the chromosomes is converted during this decoding procedure into a format that is better suited for additional analysis and interpretation during the following phases of the algorithm.

- *Evaluation*: In this stage, each new GA-solution goes through an evaluation using the SSA algorithm. The new values for the SSA parameters p_1 and p_2 are produced from the gene values found in each GA-solution. These gene values serve as the adjusted values for the SSA algorithm's associated parameters.

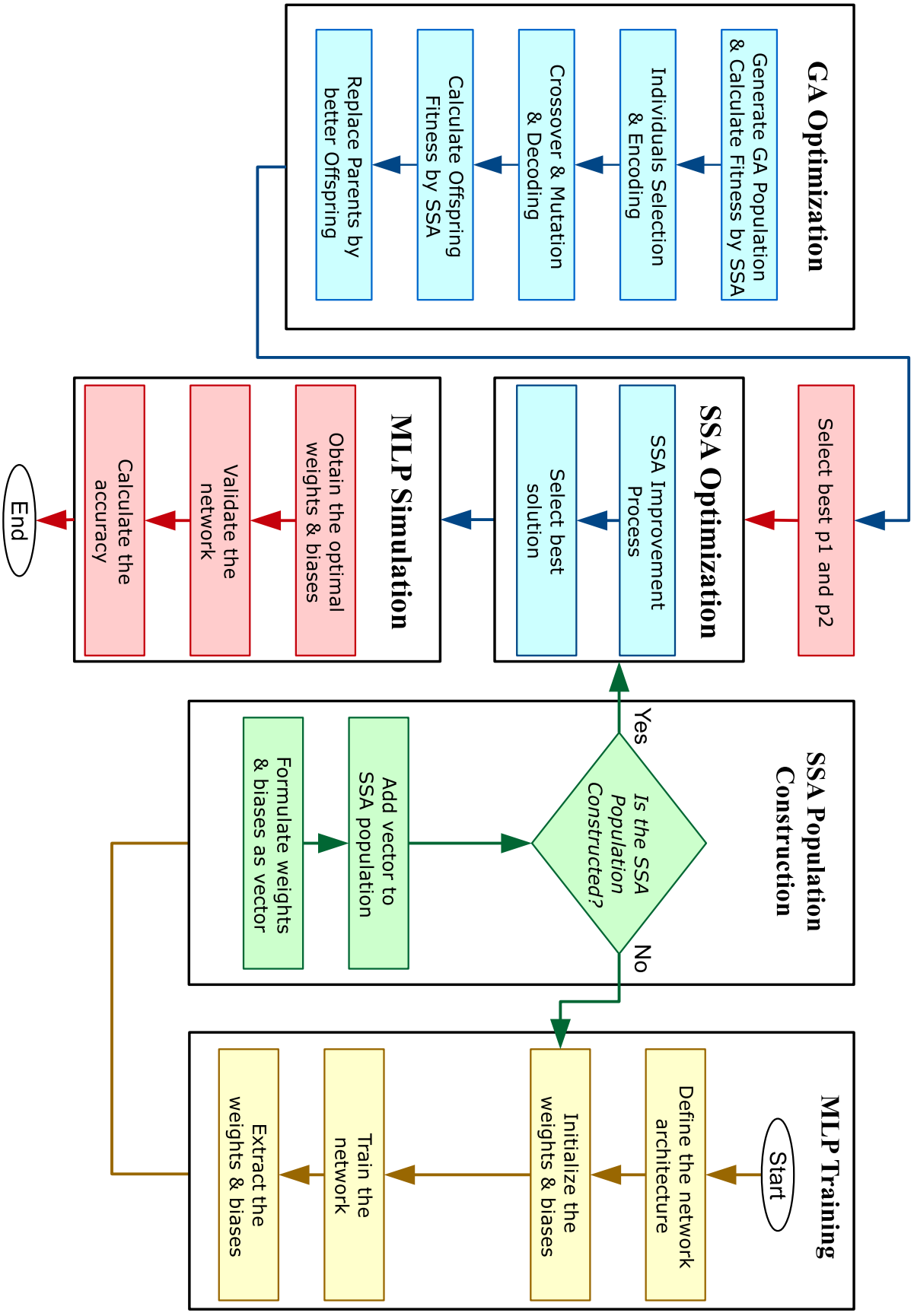

Fig. 4. Developed Technique

The optimal value obtained by using SSA determines the fitness function value for each subsequent GA-solution. The performance or efficacy of any particular solution with regard to the issue at hand is represented by this optimal value. Each solution may be tested for quality or appropriateness within the framework of the optimization process by using SSA to determine its fitness function values.

- *GA Termination Criteria*: To attain the maximum number of generations, repeat the previous procedures (apart from initiation) as many times as necessary. Each generation goes through several procedures like selection, crossover, mutation, and evaluation as the process continues repeatedly.
  The optimal GA-individual is chosen once the maximum number of generations has been reached. The SSA technique is then used to choose this individual as the candidate for its optimization process. This individual displays the best fitness or performance among all the individuals in the GA-population. The goal of choosing this most optimal GA-solution is to enhance the SSA algorithm's optimization capabilities by enhancing and refining the parameter values linked with it.
- *Select best parameters*: These optimal GA-individuals are chosen when the optimization procedure by the GA is finished and the best individuals have been identified. The SSA algorithm then uses the top candidates that were discovered by the GA optimization as new parameter values. By using these optimal parameters, the SSA algorithm is given the most precise and potent set of parameter values, improving both its performance and its capacity for optimization.
- *Optimization*: Using the optimal parameters discovered in the preceding rounds, the SSA-GA approach is used during optimization. This strategy tries to further optimize the SSA-GA-population and identify the optimal SSA-GA-solution. The MLP's weights and biases are then updated using the best SSA-GA-solution that was chosen. The MLP is provided with better parameters, boosting its prediction accuracy and overall performance by including this new set of perfect weights and biases.
- *Run the MLP simulation phase*: Using recently improved weights and biases, the MLP is run during the simulation phase. The validation data set is used to run this simulation. MLP uses updated weights and biases to produce predictions and estimates based on input data during simulation.
- *Accuracy calculation*: After the simulation, the MLP's estimation accuracy is calculated. Based on the supplied validation data, this accuracy measurement assesses the degree of accuracy and correctness of the MLP's forecasts and estimates. They may be quantitatively assessed by assessing the precision, the efficiency of the optimization procedure, and the influence of the new and optimized weights and biases on the MLP's estimate performance.

The SSA algorithm's parameters are dynamically adjusted by the GA algorithm by following these formal procedures. As a result, the SSA-GA technique's optimization capabilities are enhanced, increasing the accuracy of MLP predictions. By combining the modeling prowess of MLP and the optimization skills of SSA, this integrated strategy provides an efficient remedy for resolving the SEE problem.

Fig. 4, which shows a flow chart outlining the methodology's several phases, graphically illustrates the suggested and developed process. The proposed methodology is illustrated graphically in the flow chart, which shows the sequence and connections between the numerous parts and steps that make up the method.

IV. Experiment and Results

This part focuses on the thorough design and construction of an experiment that aims to solve the effort estimation problem, a crucial component of software development. The experiment makes use of MLP embedding together with SSA and GA, two potent metaheuristic techniques. The main goal of this experiment is to build a solid model that can calculate the effort needed for software development projects with accuracy.

The experiment's outcomes will be crucial in determining whether or not the created model is effective. We'll conduct a detailed analysis and comparison of the accuracy and performance of the proposed model with those of currently used effort estimation methods. The outcomes will also give important insights into the model's potential and capabilities for actual software development scenarios.

A. Datasets Used

The datasets used in this study are from credible sites like PROMISE and GitHub and are highly recognized benchmark datasets frequently used in research. These datasets illustrate a wide variety of features, traits, and scales, demonstrating their effectiveness. Each dataset is described in full in Table I, including the number of features, dimensions, effort unit, and source repository. Albrecht, Kitchenham, and Kemerer datasets each have seven, seven, and six features, making them the datasets with the lowest feature sizes. On the other hand, Maxwell and COCOMONASA-II have the most characteristics, with twenty-seven and twenty-one, respectively. While the China and Desharnais datasets are gathered from the GitHub and PROMISE repositories and measured in "person-hours," the Maxwell dataset uses "function points" as the measurement unit, in contrast to the other datasets, which all use "man-months" as the measurement unit. The additional datasets, like Albrecht and USP05, are measured in 'man-months" and were downloaded from the PROMISE and GitHub sources, respectively.

B. Parameter Setting

A number of careful tests were done to make sure the SSA algorithm worked as intended. Finding the ideal set of parameters—specifically, the population size and maximum iterations—was the main goal. A fair value was found for each parameter: a population size of 30 and a maximum iteration limit of 300, after careful deliberation and thorough investigation. Based on the outcomes of the experiments and how they affected the performance of the algorithm, these

TABLE I. DATASETS

Dataset	Features	Dimension	Unit	Source
Miyazaki-94	048	08	man-months	PROMISE
Kitchenham	145	07	man-months	PROMISE
Desharnais	081	12	man-hours	GitHub
COCOMONASA-I	060	17	man-months	PROMISE
Cosmic	042	11	man-months	PROMISE
Albrecht	024	08	man-months	PROMISE
USP05	203	08	man-months	GitHub
Maxwell	062	27	function points	PROMISE
Kemerer	015	07	man-months	PROMISE
COCOMONASA-II	093	24	man-months	PROMISE
COCOMO-81	063	17	man-months	PROMISE
China	499	16	man-hours	PROMISE

values were determined to be the best ones. Which supports what was used in the original research [6] that developed the SSA algorithm. This study's experiments were all carried out in the environment that was specifically mentioned before.

For the MLP network, this research uses a network with three layers: an input, an output, and a single hidden layer. It is the basic structure of any simple artificial neural network [17], [18]. The trial-and-error method [19], [20] was employed to select the best number of neurons in the hidden layer. Where the number with the best fitness value was considered. Therefore, each dataset has a specific number of neurons in the hidden layer. For instance, the following setting for the number of neurons in the hidden layer for each dataset was found: Miyazaki-94: 12 neurons; Kitchenham: 5 neurons; Desharnais: 15 neurons; COCOMONASA-I: 5 neurons; Cosmic: 10 neurons; Albrecht: 15 neurons; USP05: 12 neurons; Maxwell: 5 neurons; Kemerer: 15 neurons; COCOMONASA-II: 10 neurons; COCOMO-81: 12 neurons; China: 12 neurons. Finally, the activation functions used are as presented in Section II-B, and the learning rate value is 0.05.

Additionally, each experiment was performed 21 times in a row to guarantee accurate findings. As a result of this repetition, a sizable quantity of data was gathered, allowing for the computation of the average performance based on the best results attained over the several runs. In order to conduct the studies, MATLAB 2016a was used. The computing operations were carried out using a device that had 16 GB of RAM and an Intel Core i7 CPU running at a speed of 2.0 GHz. Utilizing this hardware setup gave us plenty of processing power and memory space to support the experimental methods successfully.

C. Performance Measures

Using six important statistical variables, a thorough assessment of the performance of the developed methodologies was carried out in this study. These metrics were intentionally chosen to offer a comprehensive evaluation of the techniques and cover many facets of error analysis. It's vital to remember that no one measurement can accurately represent all of the performance traits of the developed techniques. As a result, the use of these six metrics enables a flexible and thorough assessment of many elements of the approaches' effectiveness. Researchers can gain a more detailed picture of the benefits and drawbacks of the proposed methodologies by taking into account a variety of measures. The following are the six statistical tests used in this study:

Mean Square Error (MSE): The average of the squared discrepancies between the projected values and the actual values is calculated by the commonly used metric known as MSE. In addition to being very responsive to outliers, it quantifies the overall size of the errors.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(A_i - P_i)^2 \tag{7}$$

Root Mean Square Error (RMSE): RMSE is derived from MSE and, by calculating the square root of the MSE value, offers a more understandable measurement. It serves as a representation of the errors' standard deviation and may be used to compare models across various datasets.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(A_i - P_i)^2} \tag{8}$$

Relative Absolute Error (RAE): RAE quantifies the ratio of an absolute prediction error to an absolute error of a straightforward baseline model. By dividing the error by the total absolute errors in the base model, it normalizes the error.

$$RAE = \frac{\sum_{i=1}^{n}|A_i - P_i|}{\sum_{i=1}^{n}\left|A_i - \hat{A}_i\right|} \tag{9}$$

Root Relative Squared Error (RRSE): A RAE variant known as RRSE takes the square root of the relative squared error into account. Similar to RAE but including the squared components, it offers a relative estimate of the errors compared to a baseline model.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(A_i - P_i)^2}{\sum_{i=1}^{n}\left(A_i - \hat{A}_i\right)^2}} \tag{10}$$

Mean Absolute Error (MAE): The average of the absolute discrepancies between the expected and actual values is determined by MAE. Regardless of their orientation, it offers a clear indication of the average error magnitude.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|A_i - P_i| \tag{11}$$

Mean Magnitude Relative Error (MMRE): The average relative error between the anticipated and actual values is assessed using MMRE. It is the result of dividing the actual values by the average of the absolute differences between the expected and actual values.

$$MMRE = \frac{1}{n}\sum_{i=1}^{n}\frac{|A_i - P_i|}{A_i} \tag{12}$$

where A_i is the actual values, P_i is the predicted values, and $\hat{A}_i$ is the average of actual values.

These mathematical formulas make it possible to calculate the corresponding statistical measures, providing quantitative numbers for assessing how well the research's methodologies performed.

D. The Optimal SSA Parameter Values Found

The authors meticulously looked into many issue scenarios during the considerable testing done for this work in order to determine the best parameter values for the SSA method in each situation. The outcomes of this thorough investigation are shown in Table II, which highlights the precise SSA parameter values that are most successful in resolving the issues under consideration.

The proper values for the SSA parameters, namely p_1 and p_2, for each problem examined in this research project are clearly outlined in Table II. The fact that each problem requires a different set of parameter values in order to function at its best reveals the individuality of each one.

It is essential to understand that the values shown in Table II are neither constant nor unchangeable. Instead, they are prone to variation since the algorithms used are, by their very nature, probabilistic. There is a chance that the calculated parameter values might vary if the experiments were repeated. This difference might be traced to the algorithms' random features, which provide an element of ambiguity and variety to the optimization procedure.

As a result, the researchers stress how crucial it is to take the algorithms' stochastic character into account when interpreting the findings. The parameter values indicated in Table II are the best options based on the experimentation that was performed; however, they should be viewed as recommendations rather than strict guidelines. Recognizing the potential variations in these variables enables a thorough comprehension of the behavior of the algorithm and promotes a strong optimization procedure.

TABLE II. BEST OBTAINED SSA PARAMETERS AFTER TUNED BY GA ALGORITHM

Dataset	p_1	p_2
Albrecht	03	09
China	04	12
Cosmic	11	14
COCOMO-81	08	08
COCOMONASA-I	15	11
COCOMONASA-II	01	06
Desharnais	03	14
Kemerer	14	11
Kitchenham	14	14
Maxwell	03	14
Miyazaki-94	15	09
USP05	15	12

E. Comparison of SSA-GA vs. SSA-MLP and Standard MLP

There were 21 iterations of the experiment. The best findings were therefore averaged, and this was taken into consideration. Table III is a list of the outcomes. The findings of the study for several datasets utilizing MLP (Multi-Layer Perceptron) and two variants of SSA (Salp Swarm Algorithm): SSA-MLP and SSA-GA, are presented in the supplied data table. For each combination, the average MSE, standard deviation of MSE, worst MSE, and best MSE, as well as MMRE, RMSE, RRSE, MAE, and RAE, were calculated statistically.

The study presented here highlights the results of the recommended approaches and sheds light on two key findings of enormous relevance. First off, it is clear that the efficient optimization of weights and biases made possible by the seamless integration of the Salp Swarm Algorithm (SSA) with the Multi-Layer Perceptron (MLP) network has a significant impact on the predictive capacities of the MLP network. The performance and precision of the MLP predictions are significantly improved as a result of this integration.

The MLP network's training process is improved and made more efficient by using the SSA technique. The SSA algorithm presents a novel method of searching the solution space that is motivated by the organic movement patterns of salp swarms. This technique uses a series of dynamic equations to direct the salps in the direction of the best outcome. The network's predictive capacity is greatly increased as a result of the interaction between the MLP and the SSA algorithm, producing better outcomes and more accurate predictions.

The recommended improvements to the SSA algorithm, which were performed by fine-tuning its parameters with the help of the Genetic Algorithm (GA) have also been shown to significantly improve its optimization performance. The best set of parameters for the SSA may be found using the GA algorithm, which takes its cues from the mechanisms of natural selection and genetics.

The SSA algorithm's efficacy and efficiency in improving the weights and biases are significantly increased by exposing it to the GA's optimization capabilities. Superior optimization results are obtained as a result of the SSA method being able to adapt more precisely to the unique traits and needs of the current issue thanks to this parameter tweaking procedure. The amalgamation the GA algorithm with the SSA algorithm enables the latter to more thoroughly explore the solution space and to converge to optimal solutions in a more effective and efficient way.

In summary, the results obtained using the developed approaches highlight two important facts. First off, by enabling the efficient optimization of weights and biases, the SSA algorithm's integration with the MLP network has a significant positive impact on the performance of MLP predictions. Second, by applying the GA algorithm, the SSA algorithm has been greatly improved in terms of its optimization efficiency, which eventually results in higher-quality results and more precise forecasts. These discoveries open up new opportunities for additional study and application in a variety of fields, enhancing the area of predictive modeling and optimization approaches.

F. Comparing the SSA-MLP with State-of-the-Art Methods

A detailed and thorough comparison is made in this phase of the study, contrasting the performance of the recommended approach with leading-edge techniques that have been described in the body of existing literature. The comparison's objective is to show the created technique's effectiveness

TABLE III. RESULTS OBTAINED BY SSA-GA, SSA-MLP, AND STANDARD MLP

Dataset	Method	MSE				MMRE	RMSE	RRSE	MAE	RAE
		avg	std	worst	best					
Miyazaki-94	MLP	0.0035	1.39E-05	0.0042	0.0035	0.3260	0.0218	0.2440	0.0473	0.2130
	SSA-MLP	0.0010	5.44E-04	0.0020	0.0004	0.0634	0.0434	0.2760	0.0274	0.1420
	SSA-GA	0.0008	1.85E-05	0.0008	0.0008	0.0507	0.0022	0.1200	0.0075	0.1270
Kitchenham	MLP	0.0232	9.18E-04	0.0221	0.0220	1.7800	0.1350	0.5890	0.2380	0.6640
	SSA-MLP	0.0132	4.54E-03	0.0149	0.0084	1.1400	0.1220	0.3910	0.0626	0.5290
	SSA-GA	0.0092	5.46E-03	0.0147	0.0037	1.0200	0.1130	0.2140	0.0359	0.3520
Desharnais	MLP	0.0145	8.32E-04	0.0166	0.0148	0.5640	0.2230	0.5240	0.0848	0.5100
	SSA-MLP	0.0091	1.24E-03	0.0128	0.0053	0.2230	0.0192	0.2760	0.0486	0.3050
	SSA-GA	0.0028	1.14E-04	0.0046	0.0042	0.0637	0.0116	0.1080	0.0011	0.1320
COCOMONASA-I	MLP	0.0074	6.89E-05	0.0065	0.0064	1.6400	0.0329	0.2840	0.0773	0.4430
	SSA-MLP	0.0032	9.25E-05	0.0041	0.0041	0.7190	0.0142	0.0554	0.0048	0.0576
	SSA-GA	0.0026	1.19E-04	0.0015	0.0025	0.1340	0.0121	0.0344	0.0033	0.0356
Cosmic	MLP	0.0000	8.98E-10	0.0000	0.0000	0.0004	0.0004	0.0083	0.0182	0.0919
	SSA-MLP	0.0000	2.29E-08	0.0000	0.0000	0.0002	0.0000	0.0014	0.0000	0.0073
	SSA-GA	0.0000	3.14E-11	0.0000	0.0000	0.0001	0.0000	0.0004	0.0000	0.0007
Albrecht	MLP	0.0261	2.65E-05	0.0214	0.0219	0.3490	0.1750	0.3980	0.0895	0.4460
	SSA-MLP	0.0095	8.25E-03	0.0169	0.0028	0.1980	0.0194	0.0438	0.0304	0.0358
	SSA-GA	0.0075	6.97E-03	0.0183	0.0010	0.0854	0.0126	0.0306	0.0132	0.0134
USP05	MLP	0.0144	3.93E-04	0.0294	0.0152	8.7500	0.1390	0.7830	0.0663	1.8700
	SSA-MLP	0.0102	7.24E-03	0.0231	0.0031	4.9600	0.0170	0.3820	0.0174	0.6460
	SSA-GA	0.0072	6.56E-03	0.0122	0.0011	1.5800	0.0115	0.0596	0.0076	0.2340
Maxwell	MLP	0.0056	7.86E-04	0.0076	0.0060	1.0500	0.0717	0.4500	0.0704	0.4880
	SSA-MLP	0.0038	2.16E-03	0.0052	0.0009	0.1720	0.0034	0.0262	0.0033	0.0329
	SSA-GA	0.0007	2.46E-05	0.0007	0.0007	0.1210	0.0020	0.0172	0.0019	0.0112
Kemerer	MLP	0.0002	9.87E-07	0.0002	0.0002	0.2680	0.0132	0.0486	0.0201	0.0242
	SSA-MLP	0.0000	1.24E-06	0.0000	0.0000	0.0072	0.0042	0.0174	0.0036	0.0134
	SSA-GA	0.0000	6.42E-07	0.0000	0.0000	0.0047	0.0035	0.0109	0.0014	0.0063
COCOMONASA-II	MLP	0.0103	8.77E-05	0.0338	0.0113	1.7200	0.1230	0.4170	0.0578	0.4450
	SSA-MLP	0.0050	1.34E-04	0.0063	0.0061	4.3800	0.0462	0.3790	0.0276	0.1980
	SSA-GA	0.0041	1.03E-03	0.0067	0.0039	1.7600	0.0315	0.1120	0.0213	0.0551
COCOMO-81	MLP	0.0160	9.77E-05	0.0161	0.0248	0.1100	0.1470	0.5720	0.2560	0.4460
	SSA-MLP	0.0073	1.42E-04	0.0086	0.0076	3.3400	0.0438	0.3930	0.0401	0.2780
	SSA-GA	0.0040	1.05E-04	0.0031	0.0011	0.0955	0.0126	0.1110	0.0196	0.0642
China	MLP	0.0032	8.79E-05	0.0031	0.0019	0.7450	0.0687	0.3170	0.0288	0.3830
	SSA-MLP	0.0027	2.04E-03	0.0029	0.0004	1.3800	0.0321	0.2620	0.0260	0.3860
	SSA-GA	0.0013	1.14E-03	0.0014	0.0002	0.6300	0.0273	0.2070	0.0221	0.2540

and superiority in addressing the Software Effort Estimation (SEE) problem. The comparison study takes into account two different scenarios and thoroughly compares the created strategy to comparable techniques that have been specifically designed to tackle the SEE problem. The state-of-the-art in the field is represented by these chosen approaches, which also act as benchmark models for evaluating the improvements and contributions of the recommended approach.

The first comparison compares the developed technique to the strategy put forward by Kassaymeh et al. (2021), [21], while the second comparison compares the developed methodology to the strategies put forth by Khan et al. (2021), [22]. The benchmark datasets used in each of these comparative assessments are the same ones used in this study, and they use the same two evaluation metrics to assess performance: mean squared error (MSE) and mean magnitude of relative error (MMRE). Tables IV and V include the specific findings of these comparisons, respectively.

The SSA-GA technique clearly outperforms the SSA-BPNN method in terms of Mean Squared Error (MSE) performance metrics across all datasets in the first comparison, as shown by the results of the comparisons that were conducted. This shows that, when compared to the SSA-BPNN approach, the SSA-GA algorithm offers greater accuracy and precision in calculating software effort. this due to ability of the developed model that elicit the benefit of GA algorithm in adjusting its parameter according to problem in the hand.

Furthermore, in the second comparison, it is shown that the SSA-GA technique outperforms a number of other cutting-edge algorithms. The SSA-GA algorithm stands out as the best performer in terms of predictive abilities when compared to the Straw Berry (SB), Ant Colony Optimization (ACO), Cuckoo Optimization (CO), Genetic Algorithm (GA), Grey Wolf Optimizer (GWO), Particle Swarm Optimization (PSO), and Bat Algorithm (BA) algorithms.

TABLE IV. COMPARISON BETWEEN SSA-GA AGAINST STATE-OF-THE-ART METHODS [21]

Method	SSA-GA	SSA-BPNN
Miyazaki-94	**8.41E-04**	3.50E-03
Kitchenham	**9.20E-03**	2.29E-02
Desharnais	**2.76E-03**	1.57E-02
COCOMONASA-I	**2.60E-03**	7.40E-03
Cosmic	**7.57E-11**	1.34E-07
Albrecht	**7.52E-03**	1.61E-02
USP05	**7.23E-03**	1.44E-02
Maxwell	**6.53E-04**	6.80E-03
Kemerer	**2.47E-06**	1.62E-04
COCOMONASA-II	**4.07E-03**	1.03E-02
COCOMO-81	**4.01E-03**	1.60E-02
China	**1.30E-03**	3.00E-03

- comparison in term of MSE
- best results in bold

In particular, the SSA-GA method exhibits outstanding results on the COCOMO-81 and Maxwell datasets while taking into account the assessment metrics of mean magnitude of

relative error (MMRE). The SSA-GA method outperformed the other state-of-the-art algorithms analyzed in the study in terms of performance on these datasets, demonstrating the algorithm's greater capacity to estimate software effort precisely and with less relative error.

These superiority results are due to the fact that the SSA algorithm in the SSA-BPNN methods did not obtain parameter tuning, as is the case in the method developed in this paper (SSA-GA). This gives conclusive evidence of the desired benefit of introducing another algorithm (GA here) to adjust the parameters of the main algorithm (SSA) so that the developed algorithm becomes more flexible and effective in handling various prediction problems. In addition, the superior results of the developed algorithm play a pivotal role in proving the need to replace traditional training methods for artificial neural networks with metaheuristic algorithms to enhance prediction accuracy and raise the quality of the results.

TABLE V. COMPARISON BETWEEN SSA-GA AGAINST STATE-OF-THE-ART METHODS [22]

Method	COCOMO-81	Maxwell
SB	3.6100	2.7200
ACO	5.6100	4.5400
CO	4.0700	2.8600
GA	4.7800	3.2200
GWO	2.2100	1.4500
PSO	5.0600	3.7200
BA	4.7100	3.6400
SSA-GA	**0.0955**	**0.1210**

- comparison in term of MMRE
- best results in bold

Finally, these results highlight the SSA-GA algorithm's efficiency and competitiveness in the software effort estimation field. The algorithm's use of the Salp Swarm Algorithm and Genetic Algorithm allows it to effectively train the MLP network, improving accuracy and precision in software effort estimation. The SSA-GA method performs better than other advanced techniques, which makes it a promising and reliable alternative for resolving problems related to software work estimates.

V. DISCUSSION

The SSA-GA framework has the unique capacity to adjust its parameters dependent on the specific problem it is addressing by applying the GA algorithm. This flexibility is critical in improving the overall performance of the algorithm.

The GA method carefully chooses parameter values that are most appropriate for the SSA algorithm. The SSA-GA model assures that the SSA optimizer has optimal parameter values via such a selection procedure. As a consequence, the SSA optimizer becomes extremely trustworthy at avoiding the dangers of local optima, which can stymie exploration of the whole search space. Furthermore, by including the GA method, the SSA-GA model gets the ability to find and explore the most promising parts of the search space.

The ability to avoid local traps and explore the area of search effectively is critical for striking a balance between exploration and exploitation. Exploration entails investigating different parts of the search space to find possibly optimal solutions, whereas exploitation involves refining and improving the identified solutions. Using the SSA-GA model increases the likelihood of achieving this delicate balance.

On the other hand, the combination of the MLP with the tuned-SSA technique has several advantages, such as enhanced prediction accuracy and higher reusability. The MLP network that results from using the tuned-SSA technique to improve the MLP weights is more reconfigurable. This indicates that the improved MLP may be used with different SEE prediction tasks without requiring substantial adjustments or retraining, saving time and effort.

Additionally, the tuned-SSA technique's improvement of MLP weights results in a decrease in prediction error. The optimized process is successfully guided by the tuned-SSA technique, which enables the MLP to converge towards more precise predictions. This decrease in prediction error results in an improvement in prediction quality, which raises the MLP's dependability and utility.

The addition of the tuned-SSA algorithm also aids in adjusting the MLP network's rate of convergence. The process through which the MLP modifies its weights to reduce estimation error is known as convergence. The MLP weights are optimized using the tuned-SSA method in a way that promotes reasonable convergence. In situations or applications that need critical decisions, this improvement in convergence speed is essential.

A tuned SSA technique further minimizes the MLP's reliance on initial parameter values. Whereas the behavior and performance of MLP convergence can be greatly affected by the values of the initial weights. The MLP becomes less dependent on starting weight values when the Tuned SSA technique is used, which improves the MLP's stability and robustness.

In conclusion, By choosing the best parameter values, the combination of the SSA and GA algorithms improves the performance of the SSA method. With this combination, the algorithm is better able to explore interesting regions of the search space and break out of local optima. An enhanced balance between exploration and exploitation produces more trustworthy optimization results. In addition, using the tuned-SSA method to optimize the weights and biases of MLPs has a number of benefits. When used for various prediction tasks, the improved MLP network becomes more adaptable and requires less alterations. Additionally, it lowers prediction error, raising the accuracy of predictions. A more reliable and stable model is produced thanks to the MLP's faster convergence and decreased reliance on starting weight values.

VI. CONCLUSION AND FUTURE WORKS

This research explores the integration of the salp swarm algorithm (SSA) and the multilayer perceptron neural network (MLP) in order to tackle the software development effort estimation (SEE) problem. By adjusting the weights and biases of the MLP, the goal is to increase prediction accuracy. Furthermore, a suggested improvement is included by modifying the SSA's parameters using a genetic algorithm (GA). Twelve different SEE datasets with different feature sets are used to comprehensively assess the efficiency of the suggested method.

To evaluate the results of the developed SSA-GA methodology, there are two phases in the assessment process. The outcomes are first contrasted with those attained by traditional MLP and traditional MLP in combination with the original SSA. This comparison research enables a thorough comprehension of the effect of SSA on MLP prediction accuracy. Additionally, the impacts of parameter adjustments on the SSA's optimization performance and the MLP's prediction performance are examined.

In the second evaluation, the results of the developed SSA-GA methodology are contrasted with cutting-edge methods that have been used on datasets related to the SEE problem. The purpose of this comparison is to demonstrate how much better the recommended strategy is than the alternatives. This assessment offers a fair and direct comparison between the proposed methodology and other cutting-edge technologies by using identical datasets.

The main goal of these assessments is to show how SSA affects the precision of MLP predictions as well as how parameter adjustment affects SSA's and MLP's optimization and prediction performance, respectively. The findings of the two assessments consistently show that the proposed methodology is better than the competing techniques. This demonstrates how the SEE problem may be solved by combining SSA and MLP and optimizing the parameters with the GA algorithm, which increases prediction accuracy.

Possible future directions for this work might include expansion to additional software engineering domains and/or exploration of integration with other metaheuristic methods. The suggested approach may be integrated with other metaheuristic optimization methods such as particle swarm optimization (PSO), ant colony optimization (ACO), or differential evolution (DE) in the first potential direction. Investigating hybrid strategies that take advantage of several methods may result in even greater optimization performance and improved software effort estimation accuracy. While In the second scenario, the suggested approach may be investigated further and used in software engineering domains other than effort estimation.

ACKNOWLEDGMENT

The research reported in this publication was supported by the Deanship of Scientific Research and Innovation at Al-Balqa Applied University in Jordan, Grant Number: DSR-2021 #404.

REFERENCES

[1] P. Suresh Kumar and H. Behera, "Role of soft computing techniques in software effort estimation: an analytical study," in *Computational Intelligence in Pattern Recognition*. Springer, 2020, pp. 807–831.

[2] S. Kassaymeh, M. Alweshah, M. A. Al-Betar, A. I. Hammouri, and M. A. Al-Ma'aitah, "Software effort estimation modeling and fully connected artificial neural network optimization using soft computing techniques," *Cluster Computing*, pp. 1–24, 2023.

[3] S. Kassaymeh, S. Abdullah, M. Alweshah, and A. I. Hammouri, "A hybrid salp swarm algorithm with artificial neural network model for predicting the team size required for software testing phase," in *2021 International Conference on Electrical Engineering and Informatics (ICEEI)*. IEEE, 2021, pp. 1–6.

[4] S. N. Makhadmeh, M. A. Al-Betar, K. Assaleh, and S. Kassaymeh, "A hybrid white shark equilibrium optimizer for power scheduling problem based iot," *IEEE Access*, vol. 10, pp. 132 212–132 231, 2022.

[5] D. Mokeddem, "A new improved salp swarm algorithm using logarithmic spiral mechanism enhanced with chaos for global optimization," *Evolutionary Intelligence*, vol. 15, no. 3, pp. 1745–1775, 2022.

[6] S. Mirjalili, A. H. Gandomi, S. Z. Mirjalili, S. Saremi, H. Faris, and S. M. Mirjalili, "Salp swarm algorithm: A bio-inspired optimizer for engineering design problems," *Advances in Engineering Software*, vol. 114, pp. 163–191, 2017.

[7] D. Bairathi and D. Gopalani, "Numerical optimization and feedforward neural networks training using an improved optimization algorithm: multiple leader salp swarm algorithm," *Evolutionary Intelligence*, vol. 14, no. 3, pp. 1233–1249, 2021.

[8] S. Ahmed, M. Mafarja, H. Faris, and I. Aljarah, "Feature selection using salp swarm algorithm with chaos," in *Proceedings of the 2nd international conference on intelligent systems, metaheuristics & swarm intelligence*, 2018, pp. 65–69.

[9] X. Zhao, F. Yang, Y. Han, and Y. Cui, "An opposition-based chaotic salp swarm algorithm for global optimization," *IEEE Access*, vol. 8, pp. 36 485–36 501, 2020.

[10] S. Kassaymeh, M. Al-Laham, M. A. Al-Betar, M. Alweshah, S. Abdullah, and S. N. Makhadmeh, "Backpropagation neural network optimization and software defect estimation modelling using a hybrid salp swarm optimizer-based simulated annealing algorithm," *Knowledge-Based Systems*, vol. 244, p. 108511, 2022.

[11] L. L. Minku and X. Yao, "Ensembles and locality: Insight on improving software effort estimation," *Information and Software Technology*, vol. 55, no. 8, pp. 1512–1528, 2013.

[12] R. N. Charette, "Why software fails," *IEEE spectrum*, vol. 42, no. 9, p. 36, 2005.

[13] S. Kassaymeh, S. Abdullah, M. A. Al-Betar, and M. Alweshah, "Salp swarm optimizer for modeling the software fault prediction problem," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 6, pp. 3365–3378, 2022.

[14] A. S. V. Ojha, Varun Kumar; Abraham, "Metaheuristic design of feedforward neural networks: A review of two decades of research," *Engineering Applications of Artificial Intelligence*, vol. 60, pp. 97–116, 2017.

[15] L. P. Madin, "Aspects of jet propulsion in salps," *Canadian Journal of Zoology*, vol. 68, no. 4, pp. 765–777, 1990.

[16] S. Kassaymeh, S. Abdullah, M. A. Al-Betar, M. Alweshah, M. Al-Laham, and Z. Othman, "Self-adaptive salp swarm algorithm for optimization problems," *Soft Computing*, vol. 26, no. 18, pp. 9349–9368, 2022.

[17] F. Mumali, "Artificial neural network-based decision support systems in manufacturing processes: A systematic literature review," *Computers & Industrial Engineering*, p. 107964, 2022.

[18] E. Elahi, Z. Zhang, Z. Khalid, and H. Xu, "Application of an artificial neural network to optimise energy inputs: An energy-and cost-saving strategy for commercial poultry farms," *Energy*, vol. 244, p. 123169, 2022.

[19] Y.-C. Liu, A. Kholik, and S.-K. Lin, "A machine learning approach to explore tensile properties of low-temperature solders," in *2023 International Conference on Electronics Packaging (ICEP)*. IEEE, 2023, pp. 71–72.

[20] Y.-C. Liu, C.-H. Yang, and S.-K. Lin, "Sn-based solder design using machine learning approach," in *2022 International Conference on Electronics Packaging (ICEP)*. IEEE, 2022, pp. 43–44.

[21] S. Kassaymeh, S. Abdullah, M. Al-Laham, M. Alweshah, M. A. Al-Betar, and Z. Othman, "Salp swarm optimizer for modeling software reliability prediction problems," *Neural Processing Letters*, vol. 53, no. 6, pp. 4451–4487, 2021.

[22] M. S. Khan, F. Jabeen, S. Ghouzali, Z. Rehman, S. Naz, and W. Abdul, "Metaheuristic algorithms in optimizing deep neural network model for software effort estimation," *IEEE Access*, vol. 9, pp. 60 309–60 327, 2021.