Received: March 11, 2023. Revised: May 10, 2023.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

An Improving Long Short Term Memory-Grid Search Based Deep Learning Neural Network for Software Effort Estimation

Robert Marco[1]* **Sharifah Sakinah Syed Ahmad[2]** **Sabrina Ahmad[2]**

[1]*Department of Informatics, Universitas Amikom Yogyakarta, Yogyakarta, Indonesia*
[2]*Faculty of Information & Communication Technology, Universiti Teknikal Malaysia Melaka, Malaysia*
*Corresponding author's Email: robertmarco@amikom.ac.id

Abstract: One of the main reasons that hinders making software effort estimation remains a most of the unresolved problem due to the heterogeneous nature of software data with complex structures. In processing nonlinear data, the long short-term memory (LSTM) model is often used for the purpose of solving discriminative and generative problems. Taking into account the fact that the LSTM network have low computational efficiency, due to the need to set a large number of hyperparameters. However, training deep learning models requires expensive work in terms of specifying hyperparameter configurations in the model. The grid search (GS) optimization method is used to find the best hyperparameter values for deep learning networks, which have many different hyperparameters that affect how well the network architecture works. In this paper, we proposed the grid search method as a quick step toward optimizing the parameters of the LSTM model. An empirical study was conducted using five datasets. From our results, we have seen that the LSTM-grid search model consistently performs better across datasets in mean absolute error (MAE) and root mean squared error (RMSE) compare to existing work on using machine learning approach for software effort estimation. The main advantage of training the LSTM show that the grid search finds hyperparameters which results in faster convergent, the possibility of generalization, and better coefficient of determination in estimating software effort.

Keywords: Grid search, Long short-term memory, Hyperparameter tuning, Software effort estimation.

1. Introduction

Software effort estimation (SEE) is a method for estimating the effort (in person-hours per month) required to develop a software project [1]. Unfortunately, uncertainty and imprecision are inherent in the predictive environment of software efforts [2].

Over the decades, many SEE techniques have been proposed, but the effort has been exceeded in software project management [3]. Machine learning techniques are highly effective at simulating uncertainty to improve decision-making [4]. Unfortunately, using machine learning techniques without exposing them to the process of parameter optimization may not completely solve particular problems [5]. Thus, it is necessary to make hyperparameter adjustments to get better results in machine learning models [6]. Song et al. (2013) determined that parameter configuration is one of the factors that can influence model accuracy [7]. In addition, there is evidence that parameter tuning can affect which model is deemed superior to others [8].

Several machine learning techniques including regression tree (RT) [9], neural network (NN) [7, 6, 10, 11] support vector regression (SVR) [8, 12, 13] k-nearest neighbors (kNN) [7, 14] have been frequently used for prediction problems in the field of SEE. Unfortunately, RT requires space-consuming model storage, and data prediction is time-consuming and prone to overfitting on small training data sets [15]. Meanwhile, NN performs worse on smaller datasets than on complete datasets [15] and usually suffers from overfitting problems, and its explanatory ability is weak [7]. SVR is required for large data sets, long training periods, and high computation time.

Furthermore, kNN is ineffective in large data and relies heavily on parameter settings [16].

Although several methods have been proposed to predict software effort, there is no consensus on which technique makes the best estimate in all these situations [17], and still giving disappointing results [18]. Therefore, the study of SEE continues to be a challenging issue for researchers and project managers.

The problem is the difficulty of hyperparameter setting, which impacts the quality of the resulting predictive models [7]. They don't have clear defaults and can be agreed upon in different applications [16]. Meanwhile, hyperparameters setting manually not only requires a deep understanding of the model but is also impractical, time and cost-inefficient [5]. Several other studies have shown that the choice of information estimation techniques based on parameter tuning has a large effect on the accuracy of the SEE method [19]. In the worst-case scenario, incorrect parameter settings may result in poor performance [20].

Long short-term memory (LSTM) is proposed as a model for deep learning to enhance the relevance of training samples. The LSTM method as a predictor has the best accuracy results than other traditional methods [21]. The LSTM model is commonly employed to handle discriminatory and generative issues in nonlinear data processing [21], in the field of time series, such as [22-24], and the limited use of LSTM in the field of regression, such as [21, 25, 26]. LSTM networks are efficient and flexible in conserving long-term memory [27]. LSTM, on the other hand, has hyperparameters capable of being tuned to provide the model with good performance [28]. On the other hand, due to the increased complexity of the LSTM network, there are more hyperparameters to optimize during training, resulting in a greater computational burden [26].

One way to accelerate the training process is to use optimization methods. Hyperparameter search is an efficient automatic LSTM parameter adjustment technique in deep learning [27]. The grid search (GS) method was developed to optimize LSTM parameter values. Grid search is a comprehensive technique. It will attempt every option within the specified parameter range and select the best-performing parameter values as the model's super parameters [27, 29] with examines each hyperparameter combination using permutation and combination [30]. Other strategies have also been proposed earlier, such as random search [31], gradient optimization [32], and bayesian optimization [33]. Unfortunately, random search cannot determine the values that optimize network performance [34]. Bayesian optimization is useful for solving problems that have a limited number of hyperparameters and difficult to parallelize [6].

Grid search is feasible because the number of evaluations is lower and leads to a better-studied model through less computational time [24]. This method uses a large-scale search in a small dataset to sample a representative training dataset. Then, followed by the entire dataset is for fine-tuning, which helps to balance the cost of time and accuracy [27]. In grid search, a model is built for each possible parameter combination, resulting in many iterations, but will give the best combination [29].

This study proposes a deep neural network based on LSTM-grid search to overcome the uncertainty of software effort estimation. The main reason grid search is added to the LSTM model (LSTM+GS) is to optimize the hyperparameters. Values that can control the learning process are called hyperparameters. Tuning the hyperparameters makes sure that a model can solve a problem in the optimally way possible by lowering losses that have already been set and giving accurate results.

The remaining sections of this paper are structured as follows. The second section explain of related studies. In thrid section the theory of LSTM and grid search is discussed. The fourth section describes experiments design. The five section contains the experimental results and discussion. Finally, six section contains the conclusion of this paper.

2. Related studies

In recent years, many academics have shown increasing interest in applying hyperparameter tuning techniques to optimize learning algorithms.

Meanwhile, several works in the SEE field have used hyperparameter tuning optimization in online and offline learning. Using a grid search technique, Song et al. (2013) studied the regulatory impact of five machine learning techniques, including kNN, RT, RT+bagging, multilayer perceptron (MLP), and MLP+bagging. To discover the optimal value without assembling an ensemble. kNN, RT, and RT+bagging were insensitive to parameters tuning, whereas MLP and MLP+bagging were extremely sensitive. Bagging gives results closest to the optimal hyperparameter configuration [7].

Zakrani et al. (2018) optimized SVR using the grid search technique. Results suggest that our method can enhance the performance of the SVR method [13]. They used base learners (MLP, RF, adaboost regressor, and linear regression) and stacked ensemble learning with two ways to tune

hyperparameters: genetic algorithm (GA) and particle swarm optimization (PSO). The results of experiments show that prediction accuracy is better when hyperparameters are set using PSO [6].

Minku (2019) says that using linear regression on a logarithmic scale (LogLR) makes more accurate predictions [35]. In the meantime, Minku and Yao (2013) analyzed the RT, RT+bagging, and MLP+bagging approaches, which have been demonstrated to perform well on several data sets [9]. It is crucial to tune SVR parameters into the context of SEE. In particular, a taboo search has been suggested as a method for locating the optimal values for the SVR parameters [12]. Elish (2013) carried out an experiment in which they presented a heterogeneous ensemble that was constructed using five different machine learning approaches. These techniques included kNN, SVR, MLP, decision tree (DT), and radial basis function networks (RBFN)[14].

Villalobos-Arias and Quesada-López (2021) investigated ridge regression (RR), classification and regression tree (CART), and support vector regression (SVR) utilizing grid and random search in conjunction with six bio-inspired algorithms. According to the results, the Flash+Log+SVR model ranks first in accuracy across the majority of data sets, while the Hyperband+Log+RR model ranks first in stability across the majority of data sets [36]. Specifically, the best overall accuracy comes from a stacked ensemble that takes the average of the predicted effort values from random forest (RF), analogy based estimation (ABE), ordinary least squares (OLS), bagging, adaboost, and gradient boosting. All of these methods are optimized using grid search techniques [37].

In the field of software engineering, researchers have employed a variety of neural network techniques to software effort estimation. Kodmelwar et al. (2018) proposed modified neural network (MNN) based deep learning is formulated using the cuckoo search algorithm. The type of neural network that has been created is a convolutional network. After this stage, the optimization is carried out using hybrid particle swarm optimization (HPSO). The experimental results demonstrate that the proposed work employing MNN is more competent than the existing approach [38].

Choetkiertikul et al. (2019) suggest the long-deep recurrent neural network (LD-RNN) prediction model for estimating story points by combining LSTM and recurrent highway network, which are both strong deep learning architectures. The proposed approach consistently outperforms the general baseline according to the evaluation results. However, this method requires time-consuming during training [39].

Favero et al. (2020) applying the pre-trained BERT method with fine-tuning hyperparameter gives promising results with MAE values is 4.25 and standard deviations 0.17. This method has reliability, generalizability, speed, and low computational cost [40].

Khan et al. (2021) proposed a deep neural networks (DNN) model. This model is based on a meta-heuristic approach. They use gray wolf optimizer (GWO) and strawberry (SB). The results of the experiments show that GWO has a significant edge over other methods in estimation accuracy [10]. Meanwhile, in an empirical study, Kangwantrakool et al. (2020) investigated the effectiveness of using sequence models in unstructured SEE by manually hyperparameters tuning. The LSTM sequence model achieved the lowest MAE at 0.705 (ISEM) and 14.077 (ISBSG), respectively [41].

Ozturk (2021) proposes a novel approach to deep regression based on a binary search-based method implemented in the proposed feed-forward deep neural network algorithm (FFDNN). When paired with more advanced hyperparameter search methods, the deep learning approaches can attain a greater level of performance than other methods that rely on more traditional methods, like random and grid search techniques. In addition, Using binary search-based algorithms in FFDNN has also sped up hyperparameter optimization [11].

3. Our approach

In this section, we have discussed the introduction and studies related to the formulation of the problem and explained the list of notations used

Table 1. List of notations

Notations	Description
x_t/y_t	Input/output
h_t/H	Hidden vector
C_t	Cell state
W_i, W_f, W_o	Weight matrix (input, forget, output)
b_i, b_f, b_o	Bias vector (input, forget, output)
i_t, f_t, o_t	Input, forgert, output gate
$\sigma, tanh$	Activation function
Θ	Parameter search space
Λ	Search space
K	Latent dimension
X^n	Input variable dataset
Y^n	Target variable dataset
$\mathbb{R}$	Real number
$\mathcal{D}$	Dataset
L	Loss function
θ	Model parameter
$\hat{Y}$	Predict

in this work. A list of notations is presented in Table 1.

3.1 Problem statement

In the paper, the problem in SEE has been recognized as a regression issue with continuous output values. Let $\mathcal{D} = \{(X^n, Y^n)\}_{n=1}^N$ represent a training data set with N observations, where $X^n = [x_1^n, \dots, x_i^n]^T \in \mathbb{R}^i$ for $i = 1,2, \dots, i$, is the n-th training example consisting i features. The software project feature is given by $X_i = (x_1, x_2, \dots, x_i) \in \mathbb{R}^i$ where d represents the feature dimension of X_i, some examples of input features $(x_1, x_2, \dots, x_i)$ include the hardware platform, software development type, functional size, user interface, team expertise quality requirements, tool use, and so on. In addition, we introduce $Y^n \in \mathbb{R}^1$ where the target variable is effort measured in man-months or an equivalent unit (continuous target variable).

Donate F as a set of learning machines with LSTM and θ as their model parameters. The objective of the training procedure for point effort estimation is to determine the optimal function $f(.; \theta^*) \in F$, where $F: \mathbb{R}^d \to \mathbb{R}^1$ from input features to output effort is defined as:

$$f(X; \theta^*) = \hat{Y} \tag{1}$$

Where a testing software project (X, Y), and $\hat{Y}$ is the predicted effort value based on the input features X and the constructed SEE model, respectively.

Through the process of minimizing a loss function $L(.)$ with respect to the θ parameter of the model as:

$$L(\mathcal{D}) = \sum_{n=1}^N (\|f(X^n; \theta^*) - Y^n\|_*). \tag{2}$$

$L(\mathcal{D})$ is the mean absolute error and $\|.\|_1$ is employed, become a common instance of L in SEE. The training procedure establishes θ^* as the optimal model parameter. It is expected that the result between Y and $\hat{Y}$ has a small deviation, as $L(\|Y - \hat{Y}\|)$.

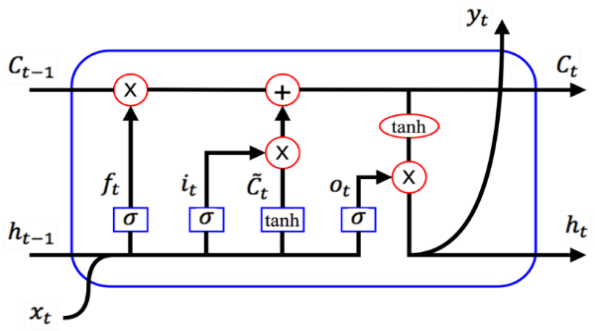


Figure. 1 LSTM architecture [42]

In practice, a vector is always a column vector except stated otherwise. A row vector is denoted by the transpose of its corresponding column vector, $\boldsymbol{X} = [X^1, \dots, X^N]^T \in \mathbb{R}^{N \times i}$. Where T represents the row (column) transpose. The matrix of training examples comprises all training information necessary to construct a SEE method with respect to a loss function.

3.2 Long short-term memory technique

According to Choetkiertikul et al. (2019), one of the popular variants of the improved basic recurrent neural network (RNN) is long short-term memory (LSTM) which uses loops to store information in the network [39], aiming to overcome the vanishing gradient and exploding gradient problems encountered with traditional RNN [42], overcame the issue of long-term dependence that had been occurring in RNN, and experiencing overfitting problems due to the many parameters that must be corrected [43].

A unit of LSTM consists of three gates (input, output, and forget gates) and a single cell. Forget gate specifies how much state memory will be reserved from the last time step for the current time step. The amount of the current input that is going to be stored in the current state memory is decided by the input gate. Lastly, the output gate determines how much of the current state memory should be issued in the current time step. Each gate possesses a weight matrix and a bias term, which will be automatically learned during the training phase [44]. Three gates regulate the flow of information into and out of the cell while the cell retains information acquired over varied time intervals. The previous layer's values are saved and utilized to determine the hidden state [45].

The architecture of the LSTM neural network is described in Fig. 1 [42], which illustrates the structure of memory cells. The t index refers to time or sequence. Where x_t, y_t, h_t, and C_t represent input, output, hidden vector, and cell state for t, respectively.

Each of these gates is a neural network whose input vector combines the hidden vector from the previous cell and the input vector. Let W_i, W_f, W_o be the weight matrix corresponding to the input, forget, and output gates, respectively. b_i, b_f, b_o the appropriate bias vectors. W_c and b_c are weight matrices and bias vectors that update cell states.

Our software effort prediction model applies a linear regression layer to the LSTM cell output layer. Let us denote the input as $X = (x_1, x_2, \dots, x_T)$,

hidden state cell as $H = (h_1, h_2, \dots, h_T)$, output sequence as $Y = (y_1, y_2, \dots, y_T)$, where $t = 1, 2, \dots, T$. LSTM will the computations as follows [46]:

$$h_t = H(W_{hy}x_t + W_{hh}h_{t-1} + b_h) \quad (3)$$

$$y_t = W_{hy}h_t + b_y \quad (4)$$

The above-described LSTM structure is then implemented through the following computations. The result of the input gate is i_t which is obtained as follows [42, 46]:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}C_{t-1} + b_i) \quad (5)$$

To calculate f_t, forget gate using the following equation [46]:

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}C_{t-1} + b_f) \quad (6)$$

The cell status is updated as follows:

$$C_t = f_t * C_{t-1} + i_t * \tanh(W_{xc}h_t + W_{hc}C_{t-1} + b_c) \quad (7)$$

The output gate uses Eq. (8) to get o_t and Eq. (9) to get the hidden vector (h_t).

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}C_t + b_o) \quad (8)$$

$$h_t = o_t * \tanh(C_t) \quad (9)$$

In Eqs. (5), (6) and (8), use σ as the sigmoid function defined in Eq. (10), while *tanh* as the hyperbolic tangent function defined in Eq. (11).

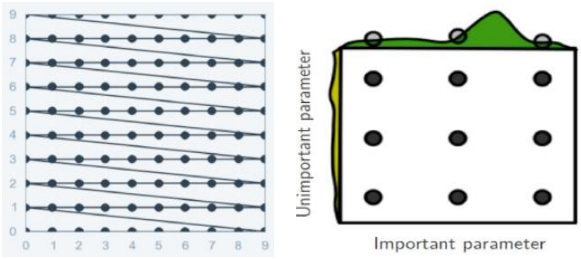
$$\sigma(x) = \frac{1}{1+e^x} \quad (10)$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (11)$$

Where σ and *tanh* are defined on the set of real numbers. For σ it ranges from 0 and 1, while *tanh* ranges from -1 to 1.

3.3 Grid search optimization

Grid search (GS) is the simplest approach, and it applies various candidate parameters to the model sequentially to find situations with optimal values that can be obtained quickly [47]. Grid search is the best way to optimize hyperparameters because it is easy to run and can be done in parallel. It also works well in low-dimensional spaces [48]. This strategy is feasible because the number of evaluations is lower and leads to a better-studied model through less computational time [24]. Although grid search is simple to implement, the computational cost is very expensive due to the large number of hyperparameters and the different levels of each [48]. Lin et al. (2008), using cross-validation techniques to the grid search method can prevent overfitting problems [49].



(a) (b)

Figure. 2 Visualization of grid search [50]: (a) grid search and (b) grid layout

Let Y be the target algorithm with k parameters to set, and the parameter θ_i be the value in the interval $[x_i, y_i]$ in parameter search space $\Theta = [x_1, y_1] \times \dots \times [x_k, y_k]$. $H: \Theta \to \mathrm{R}$ is transformed into a performance measurement function that assigns a numeric score to θ. Cross-validation is used to calculate the error H, and Θ consists of seven parameters: $n_{hiddenlayer}$, n_{neuron}, n_{epochs}, $n_{optimization}$, $batch_{size}$, $learning_{rate}$, and λ_d (dropout).

Fig. 2 is a visualization of the grid technique [50], The grid search attempts to assess each combination of hyperparameters and record precision. After evaluating all possible combinations, the model offers the most accurate parameter set. An example of a grid search with a parameter set of 3×3 is shown in Fig. 2-a.

Fig. 2-b illustrates how the nine experimental points will be tested using the grid search technique. The grid search for lower dimensional data is chosen because the number of iterations required to discover the optimal parameter set is smaller [50].

Where Λ is the search space and K is the latent dimension. Let Λ be the indexed set of configuration variables K. Grid search needs to get the optimal value from the set of values for each variable $(L^1, L^2, \dots, L^k)$, thus the number of trials on grid search is $S = \prod_{k=1}|L^k|$ elements.

4. Experiment design

Although LSTM with grid search methods are often used in other research fields, such as time series for forecasting [22-24, 47] and classification for text

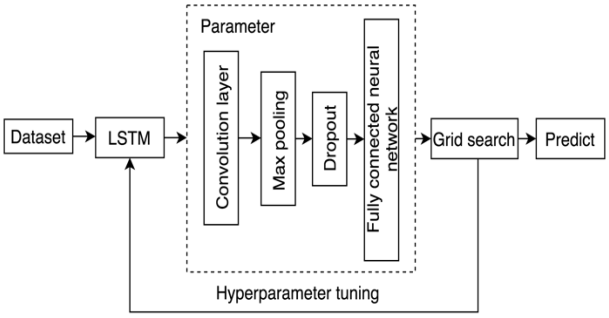


Figure. 3 Architecture LSTM+GS

Algorithm 1. The pseudocode for LSTM

input: the dataset $\mathcal{D} = \{(X^n, Y^n)\}_{n=1}^{N}$, $X \in \mathbb{R}^i$, where X is variable, Y is target, and n-th training example consisting features

1. **initialization:** unsupervised, *Y, N_epochs*
2. **data preprocessing:** normalize [0,1] and 3-fold cross-validation splitting the dataset into training and testing
3. **For** each $X \in \mathcal{D}$ **do**
4. inp←input(X)
5. $e \leftarrow$ Sequential (inp)
6. $g \leftarrow$ RNN(e)
7. $h \leftarrow$ LSTM(g)
8. $s \leftarrow$ attention(h)
9. $f \leftarrow$ ReLu(s)
10. out ← sigmoid(f)
11. **end for**
12. model← Model (inp,out)
13. model.compile(adam)
14. model.fit(*X,Y,N_epochs*)
15. $\hat{Y} \leftarrow$ model.predict(X)
16. **return** the predict $\hat{Y}$
17. **end for**
18. Output predicted ($\hat{Y}$)

Algorithm 2. The pseudocode of training grid search hyperparameter tuning

input: the dataset $\mathcal{D} = \{(X^n, Y^n)\}_{n=1}^{N}$, $X \in \mathbb{R}^i$, where X is variable, Y is target, and n-th training example consisting features; Algorithm (LSTM); Grid Search (GS); Hyperparameter (Θ)

1. **Res** ← { };
2. **for** ***i*** to *N* **do**
3. $\theta \leftarrow$ select hyperparameter (GS, LSTM, Θ)
4. model ← train (LSTM, θ, X_{train}, Y_{train})
5. Res ← eval (model, X_{test}, Y_{train})
6. **end for**
7. Θ ← Ad just (Θ, eval(res))
8. **end for**
9. **Output** Optimized Parameters

mining [29, 51]. To our knowledge, no researcher has combined LSTM with grid search for applications in the software effort estimation context.

4.1 Proposed model

Our proposed work focuses on software effort estimation using deep neural networks based on LSTM-grid search (LSTM+GS). The main novelty of our approach lies in its robust deep learning architecture. In this proposed work, the neural network in the LSTM model is fully connected in combination with the grid search. The primary objective of grid search is to find optimal hyperparameters for producing more accurate prediction results. Several of the layers in the suggested architecture, including input dataset, LSTM, convolution layer, max pooling layer, dropout, fully connected NN, grid search, and predict, make up the proposed architecture.

LSTM-based predictions are used to eliminate vanishing gradient and exploding gradient problems that may occur with RNN-based predictions. Maximum pooling is a pooling process that looks at each patch of each feature map to find the one with the greatest value. Dropout, on the other hand, preventing models from being overfit. Fully connected layer is simply, feed forward neural networks forming the last few layers in the network. Lastly, we use grid search on the whole process, as shown in Fig. 3.

The above steps of the LSTM model for our proposed regression problem are summarized in the pseudocode of Algorithm 1.

Therefore, we will present in more detail the grid search modeling in the pseudocode of Algorithm 2. for effort prediction.

4.2 Hyperparameter setting

The dimension of the LSTM by having four hidden layers having (5, 10, 15, 20, 30) neurons. The LSTM output data will be collected using max pooling in the pooling layer, and nonlinear processing will be done with the tanh function. The final step is to use a dense layer with an output dimension of 1 to get the output results. adam optimizer is often used to find the best value for the MAE loss function in regression models during the training process. Next, learning rate is set to (0.1, 0.01, 0.001, 0.0001), and the dropout rate is (0.1, 0.2, 0.4, 0.5, 0.6).

On the other hand, this method also uses the number of epochs (100, 200, 300, 400, and 500) and the batch size (64, 128, 200, 256, and 512). This study also attempts to add a 128-dimensional dense layer after the pooling layer. Comparative analysis was

Table 2. Grid search hyperparameter values

Parameter	Search space
Hidden layers	(5, 10, 15, 20, 30)
Loss	(mae, mse, logcosh, squared_hinge, hinge,)
Optimization	(Adam, RMSProp, Nadam)
Activation	(linear, relu, tanh, sigmoid)
Dropout	(0.1, 0.2, 0.4, 0.5, 0.6)
Learning rate	(0.1, 0.01, 0.001, 0.0001)
Epoch	(100, 200, 300, 400, 500)
Batch size	(64, 128, 200, 256, 512)

Table 3. Hyperparameter of the model comparison

Model	Parameter	Search space
LSTM[52]	loss	mse
	learning rate	0.0001
	activation	relu
	dropout	0.5
	epoch	500
	batch size	(128, 256)
CART[14]; CART+ bagging[7]; CART+ adaboost[53]	criterion	(mse, mae)
	splitter	(best, random)
	min samples split	(10, 100)
	max depth	(4, 125, 300)
	min samples leaf	(4, 10)
	max leaf nodes	(5,100)
	max features	(auto, log2, sqrt)
kNN[14]	n_neighbors	3
	weight function	uniform
	distance	euclidian
MLP[14]	hidden layer size	(16, 50,100)
MLP+ bagging[7]	epoch	(200, 300, 500)
	activation	(relu, tanh)
MLP+ bayesian[54]	solve	(adam, sigmoid)
	alpha	(0.0001, 0.05)
MLP+ adaboost[55]	learning rate	(0.005, 0.3)
	batch size	(128, 256)
SVR[14]	kernel	(poly, rbf, sigmoid)
SVR+ bayesian[54]	gamma	(0.01, 0.9)
	C	(1, 100)
SVR+ adaboost[54]	epsilon	(0,1)
RF[56]	criterion	(mse, mae)
RF+ bayesian[54]	n_estimator	(10, 100)
	min samples split	(10, 100)
RF+ adaboost[54]	max depth	(4, 125, 300)
	min samples leaf	(4, 10)
	max leaf nodes	(5,100)
	max features	(1, 7)
	criterion	(mse, mae)
	splitter	(best, random)
	min samples split	(10, 100)

conducted with the ReLU activation function on the fully connected layer.

In the grid search, each combination of the predefined list of hyperparameter values in the LSTM is evaluated to determine the optimal value based on the cross-validation score. Grid search, though time-consuming, will yield the optimal combination. Table 2 illustrates the hyperparameters utilized by this model.

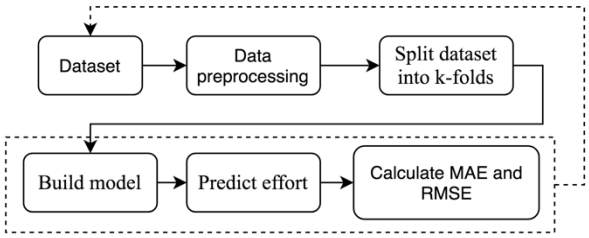


Figure. 4 Procedure all methods

4.3 Benchmark SEE model

In this section, the LSTM-grid search (LSTM+GS) model that we propose will be compared with with six baseline algorithms, with default parameter setting or use parameter optimization, shown in the Table 3.

4.4 Dataset preprocessing

We chose to use five data sets from the PROMISE repositories, which are often used in SEE studies and are open to the public [57, 58]. The general procedure diagram for all methods is described as in Fig. 4.

In detail, this section will be discussed on how to carry out the preparation and preprocessing stages of the dataset. First, feature reduction: the use of the year feature has been removed because the test scenario in this study primarily targets offline scenarios, which is thus an irrelevant feature. The next, categorical conversion: converts categorical features to numeric values for these datasets used ordinal encoding. The reason for using this method is that ordinal encoding provides a unique number code for each category [59]. Finally, normalization: each of the input features should be normalized with the interval [0,1] for X_i using the min-max normalization. Table 4 presents descriptive statistics on the data sets in terms of number of records, attributes, size, effort, mean, standard deviation (Std), skewness (Skew), and kurtosis (Kurt) of the actual recorded effort value in each data set.

Since machine learning and the LSTM model are sensitive to input scaling, the data are normalized using feature scaling within the range [0,1]. The data is divided into trains and tests while maintaining a temporary sequence of observations. The test data are utilized to evaluate the accuracy of the proposed prediction model, but are not utilized during the

Table 4. Dataset description

Dataset	Number record	Number attributes	Size unit	Effort unit	Mean	Std	Skew	Kurt
China [60]	499	19	Function point	Person-hours	331.375	1850.14	2.458	18.95
Kemerer [61]	15	8	KSLOC	Person-months	366.230	561.459	0.408	3.069
Kitchenham [62]	145	10	Function point	Person-hours	1675.802	6052.215	0.357	8.560
Maxwell [63]	62	27	Function point	Person-hours	335.977	2539.394	0.729	8.147
Nasa93 [64]	93	24	LOC	Person-months	41.195	295.891	3.919	30.46

training phase. For machine learning models, standard practice dictates a ratio of 70 percent (training set) and 30 percent (testing set) for random splitting of datasets [22].

4.5 Performance analysis

In this experiment, four performance measures were used to evaluate and compare the performance of various regression models. The regression metrics imported from the *sklearn package*, in Eqs. (12) to (15).

$$MAE = \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{n} \quad (12)$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}} \quad (13)$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2} \quad (14)$$

$$PRED(x) = \frac{1}{N}\sum_{n=1}^{N} \begin{cases} 1, if\ MRE_i \le x \\ 0, otherwise \end{cases} \quad (15)$$

5. Result and discussion

This SEE study will take offline scenarios. The training and testing project is part of the stationary training example without orders or changes in the offline scenario. Experiments were conducted using a computing platform based on Intel Core i9-9900k 5 GHz, Asus Maximus XI Hero, SSD M.2 Samsung 1Tb, HDD 4Tb WD Black, PSA Seasonic Focus 1000Watt, Ram 64Gb Vgen, and OS Ubuntu 22.04 LTS. The development environment is notepad plus, python 3.0 (32-bit), anaconda web programming interface, several libraries on Scikit-learn, and NumPy.

5.1 Model performance

LSTM performance was assessed over 1000 epochs with early stopping enabled, monitoring validation accuracy using 0.001 min_delta and 30 patience. This method is highly effective at preventing overfitting. In Figs. 5 through 9, the

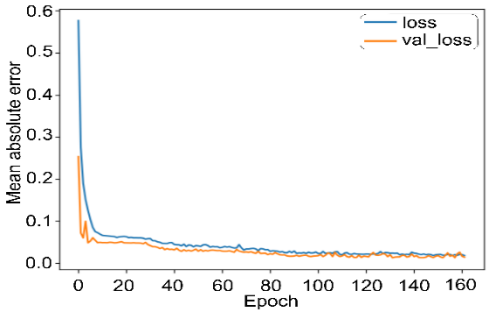


Figure. 5 Training loss on china dataset

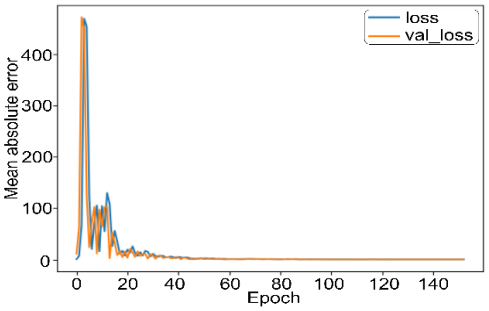


Figure. 6 Training loss on kemerer dataset

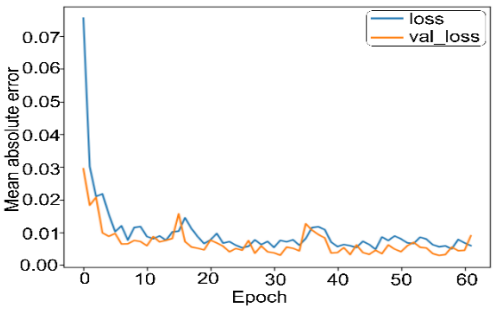


Figure. 7 Training loss on kitchenham dataset

diagnostic plot of our RNN-based LSTM model demonstrates that training and validation losses decrease as the number of epochs increases. The China dataset shows the loss and accuracy of the model for 160 epochs, Kemerer for 140 epochs, Kitchenham for 60 epochs, Maxwell for 150 epochs, and Nasa93 for 100 epochs.

We can see that some data sets on training and validation loss decrease drastically with increasing iteration time and become stable after reaching values above 10-20 epochs. The value of the loss function tends to be stable, which indicates that LSTM+GS

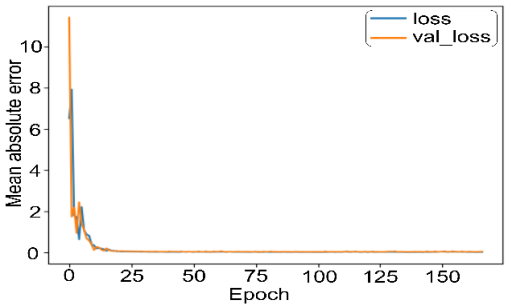


Figure. 8 Training loss on maxwell dataset

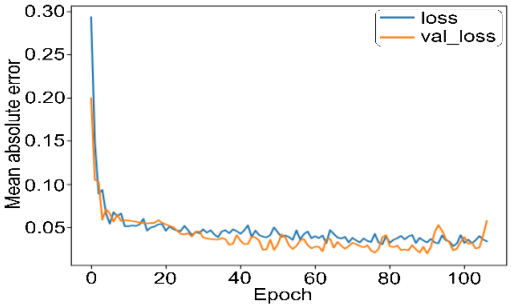


Figure. 9 Training loss on nasa93 dataset

Table 5. Performance of the LSTM+GS model

Dataset	MAE	RMSE	R^2	Pred(25)
China	554.407	1412.777	0.894	0.894
Kemerer	42.573	68.970	0.829	0.883
Kitchenham	321.106	418.956	0.976	0.976
Maxwell	2228.565	3533.907	0.843	0.856
Nasa93	172.686	283.801	0.923	0.924

has entered a state of convergence. Overall, this shows that the LSTM+GS training process is very stable. The results show that the loss function is close to constant, and it can be concluded that the model has converged. Our LSTM+GS model takes less time to train and find generalizations from the data set.

5.2 Results of hyperparameter optimization

Based on our experimental results, the hyperparameter model was selected through grid search optimization, which after being evaluated resulted in better performance. Optimization is run 10 times to determine the best parameter value with a certain search space. For these five data sets, we used the same 3-fold cross validation method as reported in [65] to divide the data into training and test samples. The LSTM+GS model consists of four layers. Adam and ReLU optimizer applied in this experiment. The nonlinear ReLU performs best as an activation function of each hidden layer. We investigate ReLU as a hidden layer activation function to solve the missing gradient issue brought on by Sigmoid [66]. Our model employs Adam performed the best and faster convergence [65, 66].

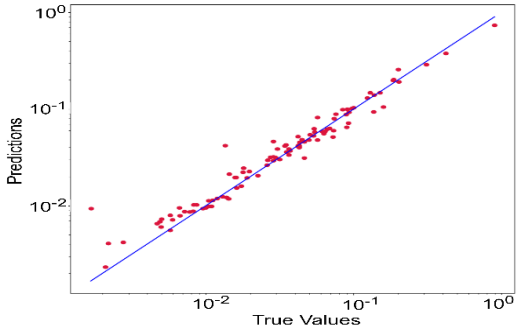


Figure. 10 Predicted effort on china dataset

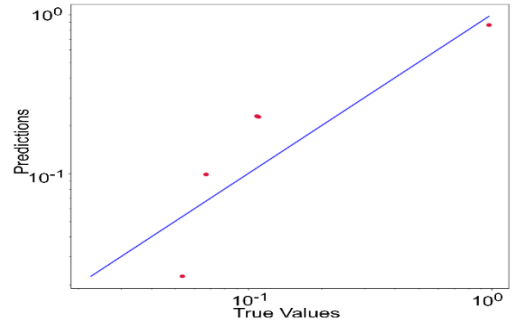


Figure. 11 Predicted effort on kemerer dataset

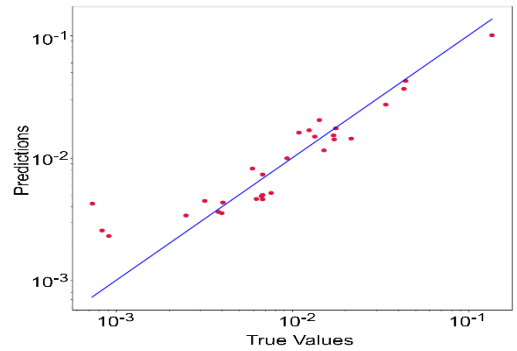


Figure. 12 Predicted effort on kitchenham dataset

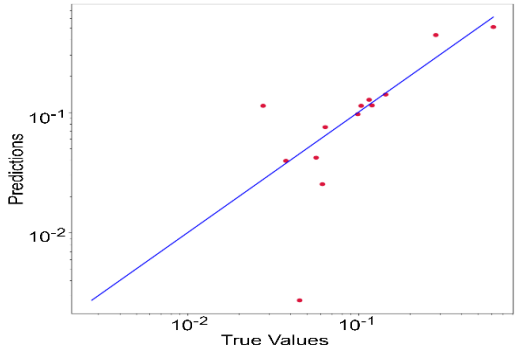


Figure. 13 Predicted effort on maxwell dataset

When using the MAE, RMSE, R^2, and Pred(25) performance evaluation metrics, lower MAE and RMSE values, higher R^2 and Pred(25) values indicate better results. Table 5 shows the MAE, RMSE, R2, and Pred(25) values obtained by applying the LSTM+GS model to the all dataset.

In Table 5 the results that have the best predictive

Table 6. Performance evaluation of MAE value for all dataset

Methods	MAE Values				
	China	Kemerer	Kitchenham	Maxwell	Nasa93
LSTM+GS	**554.407**	**42.573**	**321.106**	**2228.565**	**172.686**
LSTM	2313.504	108.127	1420.223	5215.056	*539.213*
CART	1032.260	113.200	758.241	8496.230	353.978
CART+bagging	625.302	103.751	502.794	3219.196	183.393
CART+adaboost	924.450	100.766	509.517	7313.692	283.515
kNN	1301.996	113.088	784.264	4536.974	404.235
MLP	3212.537	134.102	*2047.095*	8882.655	509.788
MLP+bagging	3249.591	138.870	2046.867	*8918.697*	507.688
MLP+bayesian	1580.818	67.022	598.185	5366.700	460.198
MLP+adaboost	*3249.693*	139.132	2044.106	8869.343	508.736
SVR	2592.019	*176.603*	1403.841	5425.094	507.548
SVR+bayesian	2228.305	131.786	1380.342	3054.978	343.846
SVR+adaboost	2586.333	172.058	1485.784	5778.088	495.916
RF	626.885	98.953	503.183	3462.259	162.123
RF+bayesian	578.960	98.953	491.404	3712.819	259.569
RF+adaboost	887.454	93.516	493.440	5052.004	267.609

Table 7. Performance evaluation of RMSE value for all dataset

Methods	RMSE values				
	China	Kemerer	Kitchenham	Maxwell	Nasa93
LSTM+GS	**1412.777**	**68.970**	**418.956**	**3533.907**	**283.801**
LSTM	4175.544	141.899	3096.038	10279.394	882.479
CART	2771.457	138.921	1489.455	*15187.327*	904.200
CART+bagging	2145.435	115.858	1094.731	5694.656	378.570
CART+adaboost	1973.087	123.108	775.290	14469.232	587.221
kNN	2584.747	120.958	1784.355	6203.140	1003.895
MLP	6578.298	219.793	*3519.173*	13079.703	1069.583
MLP+bagging	*6604.430*	*222.710*	3519.076	13106.443	*1083.958*
MLP+bayesian	3530.631	70.705	1174.686	9808.457	799.903
MLP+adaboost	6604.422	220.833	3517.262	13069.911	1079.508
SVR	5997.487	188.363	2895.880	10461.773	1061.127
SVR+bayesian	5618.617	148.643	2855.422	4775.870	711.095
SVR+adaboost	5990.275	187.379	2867.960	11009.078	1059.301
RF	2116.267	113.632	1109.433	6137.315	349.281
RF+bayesian	1914.092	113.632	1054.704	4838.835	602.169
RF+adaboost	2014.713	106.894	837.827	9828.878	418.193

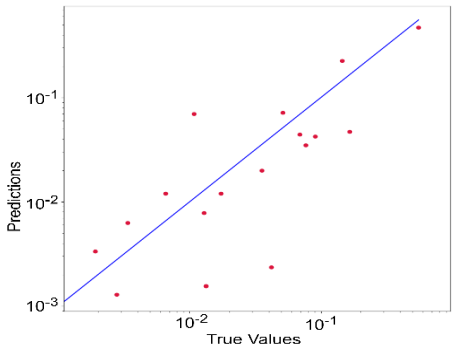


Figure. 14 Predicted effort on nasa93 dataset

performance in our model are the kitchenham dataset with a higher values of R^2 and Pred(25) with a value of 0.976 and 0.976, respectively. For nasa93 dataset the performance values are 0.923 and 0.924, respectively. Whereas, the china dataset the values seem slightly better i.e 0.894 and 0.894, respectively. Furthermore, maxwell dataset the performance values are 0.843 and 0.856. Finally, the kemerer dataset the performance values are 0.829 and 0.883, respectively. We observe that optimizing the hyperparameter by using a grid search can lead to an increase in the overall prediction performance of the model.

Figs. 10 to 14 illustrate the distribution of prediction efforts obtained from the LSTM+GS vs. actual effort around baseline. This scatter was produced by the "scatter" function of the matplotlib Python library. This function mechanically reorders minimum to maximum values by default. As shown

in the figure, the majority of actual and predicted values across all datasets are above the baseline, where actual and predicted values are identical ($\hat{Y} = Y$). It can be observed that the data points are very slightly spread out from the baseline. Therefore the correlation is higher, especially in the case of real project data sets.

5.3 Comparison with existing methods

In this subsection, the results of the proposed general RNN-based LSTM+GS model are compared with the fifteen methods in the SEE context that have been described in subsection 4.3. Tables 6 and 7 describe the performance values of the model using MAE and RMSE, where the best values are stated in bold. On the other hand, poor values are in italics.

Table 6 shows the results of a comparison of the LSTM+GS model with other techniques on five different datasets using the MAE performance measure. As can be seen, the proposed LSTM+GS model performs better than the other fifteen techniques across all datasets in terms of MAE, with a china dataset of 554.407, kemerer of 42.573, kitchenham of 321.106, maxwell of 2228.565, and nasa93 of 172.686. Meanwhile, the MLP+adaboost method has the worst MAE performance on the china dataset of 3249.693. Meanwhile, the SVR method has the worst performance on the kemerer dataset of 176.603. MLP has the worst performance on the kitchenham dataset of 2047.095, and MLP+bagging on the maxwell dataset of 8918.697. Finally, the baseline LSTM performed worst on the nasa93 dataset of 539.213.

Table 7 shows the results of the comparison of the LSTM+GS model with other techniques on five different datasets using the RMSE performance measure. As can be seen, the proposed LSTM+GS model performs better than the other fifteen techniques across all datasets in terms of RMSE, with the china of 1412.777, kemerer of 68.970, kitchenham of 418.956, maxwell of 3533.907, and nasa93 of 283.801. Meanwhile, the MLP+bagging method has the worst RMSE performance on the China dataset of 6604.430, Kemerer of 222.710, and Nasa93 of 1083.958. Meanwhile, MLP has the worst performance on the Kitchenham dataset of 3519.173. Finally, CART on the maxwell 15187.327 dataset.

It can be observed that the LSTM+GS model has the lowest MAE and RMSE values in the all dataset used with the best accuracy. This shows that the use of the hyperparameter tuning method used in this study is a grid search, and we observe that optimizing the hyperparameter to minimize the predefined loss leads to an increase in the overall model accuracy. The proposed model outperforms all other considered baseline algorithms. Therefore, incorporating grid search as a hyperparameter tuning technique improves the performance of our model, because the accuracy value of LSTM+GS is better than the baseline LSTM. On the other hand, baseline LSTM require a large number of hyperparameter settings and are prone to overfitting small training data sets. Although, LSTM has the worst performance on the nasa93 dataset, on the other hand LSTM outperforms MLP, MLP+bagging, MLP+adaboost, and SVR performance on other datasets.

In our analysis, kNN is one of the simplest approaches and works well in our analysis. Meanwhile, kNN outperformed the performance of MLP, MLP+bagging, MLP+bayesian, MLP+adaboost, SVR, SVR+bayesian, and SVR+adaboost in almost all datasets. Meanwhile, CART is not very sensitive to parameter tuning and performs poorly on small data. It can be observed, that CART+bagging and CART+adaboost have slightly lower MAE and RMSE values than baseline CART. Overall, CART+bagging and CART+adaboost have performance close to RF. On the other hand, MLP is often not one of the best approaches like the others in our analysis. It is proven that MLP+bagging has the worst performance on the china, kemerer, maxwell, and nasa93 datasets. Meanwhile, MLP+adaboost has the worst performance on the china dataset. RF is a promising technique for SEE, and it is sensitive to parameter tuning (RF+bayesian and RF+adaboost) with performance that may exceed that of the CART method.

The accuracy values (MAE and RMSE) of all models for the five datasets are depicted in Figs. 15 and 16. We observe that the LSTM+GS model is more accurate than the other benchmark models for all five datasets. Consequently, modifying the hyperparameter using a grid search can enhance the performance of our model, as the accuracy value is higher than the LSTM baseline.

5.4 Statistical performance evaluation

In this section, we further carried out a Friedman's ANOVA analysis on the regression performance for non-parametric tests used to test significance between two or more models [67]. The results of the ANOVA and post hoc tests for statistical tests can be seen in Table 8.

The ANOVA test for the 15 methods has a significant value of 0.000 (p-value<0.05). Meanwhile, the F-value is 3.693 and the F-table$_{(15,\ 319)}$ value is 1.710. Because the F-value 3.693>F-table 1.710, is

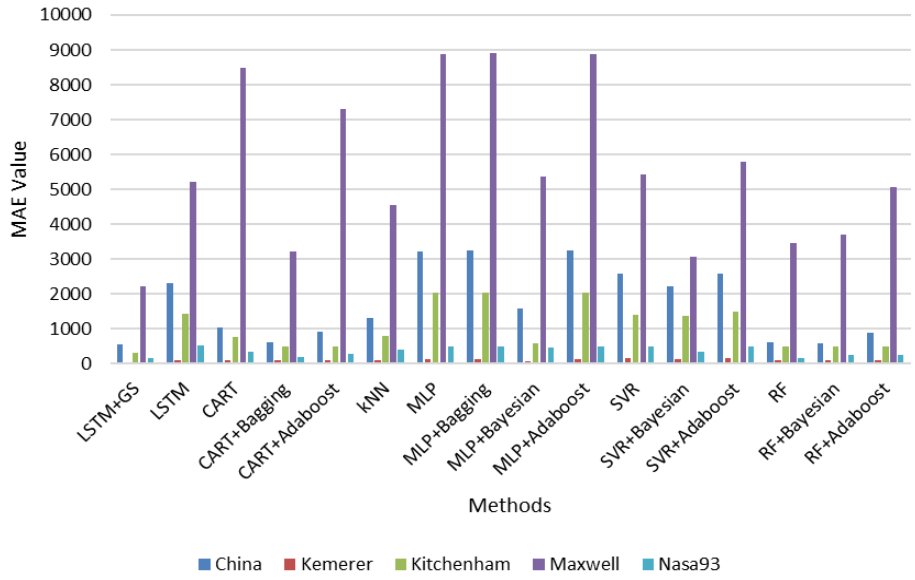


Figure. 15 Model comparison on MAE values across datasets

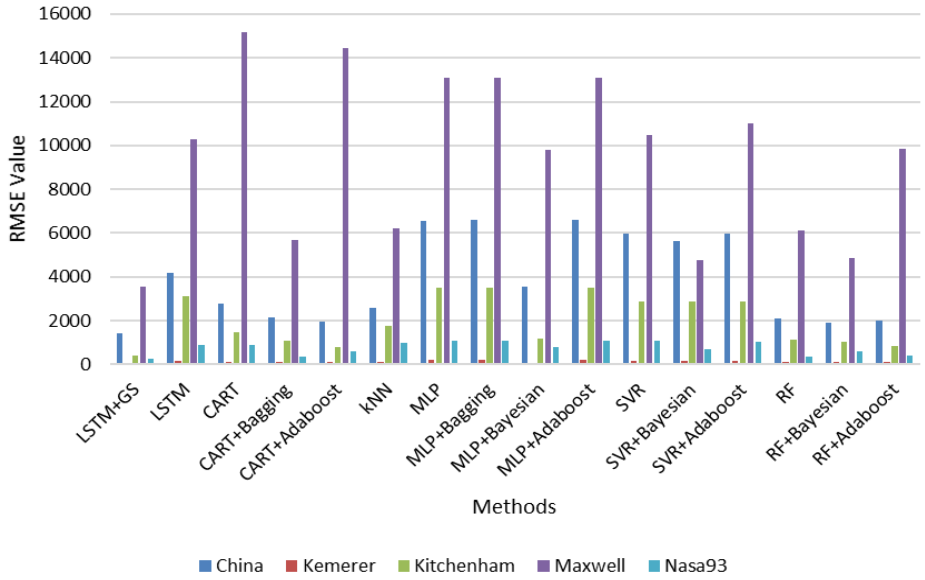


Figure. 16 Model comparison on RMSE values across datasets

Table 8. Comparison of predictor machine performance using Friedman's ANOVA test

	df	Mean Square	F-value	Sig.
Between Groups	15	16604496.01	3.693	.000
Within Groups	304	4496655.80		
Total	319			

the basis for decision making in the F test. Thus, it can be stated that the two or more models are statistically significant. In this case, two-way ANOVA shows a significant difference between the variation of factors (models). In conclusion, our LSTM+GS model performs better and more robustly than other considered baseline algorithms in the SEE field by providing significantly improved predictive results.

We further conducted a duncan multiple range test (DMRT) aims to determine the clustering of performance schemes in each algorithm and whether the difference in performance (better/worse) is statistically significant or not, which is presented in the following Table 9.

Based on the DMRT test, it is known that the results of the comparison of the prediction accuracy are divided into two groups. The first group is LSTM+GS, CART+adaboost, RF, CART+bagging, RF+bayesian, RF+adaboost, CART, kNN, and SVR+bayesian. The second group is MLP+bayesian, LSTM, SVR, SVR+adaboost, MLP, MLP+bagging,

Table 9. Model performance comparison using DMRT test

Methods	N	Subset for alpha = 0.05	
		1	2
LSTM+GS	50	664.7259	
CART+adaboost	50	732.9039	
RF	50	763.8111	
CART+bagging	50	779.0536	
RF+bayesian	50	802.0740	
RF+adaboost	50	818.6823	
CART	50	986.1421	
kNN	50	992.7523	
SVR+bayesian	50	1297.7148	
MLP+bayesian	50	1766.7781	1766.7781
LSTM	50	1870.2694	1870.2694
SVR	50	1987.7859	1987.7859
SVR+adaboost	50	2006.4632	2006.4632
MLP	50		3118.4453
MLP+bagging	50		3127.9692
MLP+adaboost	50		3132.3377
Sig.		.104	.080

and MLP+adaboost.

The DMRT test results show that the variants are LSTM+GS, CART+adaboost, RF, CART+bagging, RF+bayesian, RF+adaboost, CART, kNN, and SVR+bayesian are clusters that are not significantly different.

5.5 Threat to validity

The construct validity, we will also discuss the three main threats in the method we developed using LSTM+GS. The first threat pertains to the data set's suitability for training an RNN-based LSTM+GS model, which requires large quantities of data to mine historical patterns. We used five datasets from the software engineering repository, which are small datasets.On the other hand, we also train the LSTM method to perform well on small datasets. The second threat relates to the problem of overfitting the use of our LSTM+GS model, and the third threat concerns the issue of errors in alternative machine learning options in the context of SEE for comparison. We have randomized the training examples in each epoch. This helps to increase generalizability. Next, the use of cross-validation techniques. Then, an early stop is implemented, wherein training is terminated when the test set yields the lowest error rate. This method is highly effective at preventing overfitting. A further threat is posed by the selection of alternative machine learning techniques in the context of SEE for comparison. There may be a need for a more precise spectrum of techniques for the effort prediction problem. Based on a survey of the relevant literature, we have chosen the most effective solution for this issue.

6. Conclusion

In this studies, we propose a LSTM-grid search-based deep neural network for software effort estimation. We used a robust strategy by redesigning the RNN-based LSTM to work well on small data sets. LSTM network optimization involves several hyperparameters. The application of grid search as an optimization approach because it has relatively few function evaluations and optimizes much faster. This helps enhance deep learning as a variation of NN to generate robust models. Furthermore, the designed LSTM-grid search model is compared with other baseline algorithms under consideration using the 3-fold cross validation method and through five datasets: china, kemerer, kitchenham, maxwell, and nasa93. The evaluation criteria used are MAE and RMSE. Our study shows that the proposed model outperforms all other baseline algorithms. This study also shows that parameter tuning can result in improved model performance.

Our proposed method is in the offline scenario, the most common setting in the SEE community. Therefore, adapting the online scenario will be fruitful work in the future. We may find interesting findings by investigating the sensitivity to parameter tunings, encouraging their practical use.

Conflicts of interest

The authors declare no conflict of interest.

Author contributions

Conceptualization, R. Marco; methodology, R. Marco, S. S. S. Ahmad and S. Ahmad; validation, S. S. S. Ahmad and S. Ahmad; formal analysis, R. Marco, S. S. S. Ahmad and S. Ahmad; investigation, R. Marco, S. S. S. Ahmad and S. Ahmad; resources, R. Marco, S. S. S. Ahmad and S. Ahmad; data curation, R. Marco, S. S. S. Ahmad and S. Ahmad; writing—original draft preparation, R. Marco; writing—review and editing, S. S. S. Ahmad and S. Ahmad; visualization, R. Marco; supervision, S. S. S. Ahmad and S. Ahmad; funding acquisition, R. Marco, S. S. S. Ahmad and S. Ahmad.

References

[1] L. Song, L. L. Minku, and X. Yao, “A novel automated approach for software effort estimation based on data augmentation”, In: *Proc. of International Conf. on European Software Engineering Conference and Symposium on the Foundations of Software*

Engineering, San Francisco, USA, pp. 468–479, 2018.

[2] S. Ezghari and A. Zahi, "Uncertainty management in Software effort estimation using a consistent fuzzy analogy-based method", *International Journal of Applied Soft Computing*, Vol. 67, pp. 540–557, 2018.

[3] N. Ramakrishnan, H. A. Girijamma, and K. Balachandran, "Enhanced Process Model and Analysis of Risk Integration in Software effort estimation", In: *Proc. of International Conf. on Smart Systems and Inventive Technology*, Stanford, California, pp. 419–422, 2019.

[4] R. Alizadehsani, M. Roshanzamir, S. Hussain, A. Khosravi, A. Koohestani, M. H. Zangooei, M. Abdar, A. Beykikhoshk, A. Shoeibi, A. Zare, M. Panahiazar, S. Nahavandi, D. Srinivasan, A. F. Atiya, and U. R. Acharya, "Handling of uncertainty in medical data using machine learning and probability theory techniques: a review of 30 years (1991–2020)", *International Journal of Annals of Operations Research*, Vol. 128, pp. 1–42, 2021.

[5] M. M. Ozturk, "The impact of parameter optimization of ensemble learning on defect prediction", *International Journal of Computer Science of Moldova*, Vol. 27, No. 1, pp. 85–128, 2019.

[6] S. K. Palaniswamy and R. Venkatesan, "Hyperparameters tuning of ensemble model for software effort estimation", *International Journal of Ambient Intelligence and Humanized Computing*, Vol. 12, pp. 6579–6589, 2020.

[7] L. Song, L. L. Minku, and X. Yao, "The impact of parameter tuning on software effort estimation using learning machines", In: *Proc. of International Conf. on Predictive Models in Software Engineering*, Maryland, USA, pp. 9:1-9:10, 2013.

[8] L. V. Arias, C. Q. López, J. G. Coto, A. Martínez, and M. Jenkins, "Evaluating hyper-parameter tuning using random search in support vector machines for software effort estimation", In: *Proc. of International Conf. on Predictive Models and Data Analytics in Software Engineering*, New York, United States, pp. 31–40 , 2020.

[9] L. L. Minku and X. Yao, "An analysis of multi-objective evolutionary algorithms for training ensemble models based on different performance measures in software effort estimation", In: *Proc. of International Conf. on Predictive Models in Software Engineering*, Maryland, USA, pp. 1–10, 2013.

[10] 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", *International Journal of IEEE Access*, Vol. 9, pp. 60309–60327, 2021.

[11] M. M. Öztürk, "A tuned feed-forward deep neural network algorithm for effort estimation", *International Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 00, No. 00, pp. 1–25, 2021.

[12] A. Corazza, S. Di Martino, F. Ferrucci, C. Gravino, F. Sarro, and E. Mendes, "Using tabu search to configure support vector regression for effort estimation", *International Journal of Empirical Software Engineering*, Vol. 18, No. 3, pp. 506–546, 2013.

[13] A. Zakrani, A. Najm, and A. Marzak, "Support Vector Regression Based on Grid-Search Method for Agile Software Effort Prediction", In: *Proc. of International Conf. on Information Science and Technology*, Marrakech, Morocco, pp. 492–497, 2018.

[14] M. O. Elish, "Assessment of voting ensemble for estimating software development effort", In: *Proc. of International Conf. on Computational Intelligence and Data Mining*, Singapore, pp. 316–321 , 2013.

[15] E. Kocaguneli, T. Menzies, J. Hihn, and B. H. Kang, "Size doesn't matter? On the value of software size features for effort estimation", In: *Proc. of International Conf. On Predictive Models in Software Engineering*, Lund, Sweden, pp. 89–98, 2012.

[16] L. L. Minku and X. Yao, "Ensembles and locality: Insight on improving software effort estimation", *International Journal of Information and Software Technology*, Vol. 55, No. 8, pp. 1512–1528, 2013.

[17] M. Hosni, A. Idri, A. Abran, and A. Bou, "On the value of parameter tuning in heterogeneous ensembles effort estimation", *International Journal of Soft Computing*, Vol. 22, No. 18, pp. 5977-6010, 2017.

[18] J. J. C. Gallego, P. R. Soria, and B. M. Herrera, "Analogies and Differences between Machine Learning and Expert Based Software Project Effort Estimation", In: *Proc. of International Conf. On Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, Butterworth-Heinemann*, United States, pp. 269–276, 2010.

[19] P. Phannachitta, "On an Optimal Analogy-based Software Effort Estimation", *International Journal of Information and Software Technology*, Vol. 125, No. June 2019, pp.

106330, 2020.

[20] A. Arcuri and G. Fraser, "Parameter tuning or default values? An empirical investigation in search-based software engineering", In: *Proc. of International Conf. On Search Based Software Engineering, Szeged*, Hungary, pp. 33–47, 2011.

[21] F. Tan, "Regression analysis and prediction using LSTM model and machine learning methods", In: *Proc. of International Conf. On Artificial Intelligence and Information Systems*, Chongqing, China, Vol. 1982, 2021.

[22] S. Bouktif, A. Fiaz, A. Ouni, and M. A. Serhani, "Optimal Deep Learning LSTM Model for Electric Load Forecasting using Feature Selection and Genetic Algorithm : Comparison with Machine Learning Approaches", *International Journal of Energies MDPI*, Vol. 11, No. 1636, pp. 1–20, 2018.

[23] D. Liu, S. Lee, Y. Huang, and C. J. Chiu, "Air pollution forecasting based on attention-based LSTM neural network and ensemble learning", *International Journal of Expert Systems*, No. November, pp. 1–16, 2019.

[24] N. Bakhashwain and A. Sagheer, "Online Tuning of Hyperparameters in Deep LSTM for Time Series Applications", *International Journal of Intelligent Engineering and Systems*, Vol. 14, No. 1, pp. 212–220 , 2021, doi: 10.22266/ijies2021.0228.21.

[25] M. Qin, "Lattice LSTM Model for Function Point Based Software Cost Measurement", In: *Proc. of International Conf. On Information Technology and Artificial Intelligence*, Chongqing, China, Vol. 8, pp. 731–735, 2019..

[26] T. Kim and S. Cho, "Neurocomputing Optimizing CNN-LSTM neural networks with PSO for anomalous query access control", *International Journal of Neurocomputing*, Vol. 456, pp. 666–677, 2021.

[27] Y. Dai and J. Huang, "A Sales Prediction Method Based on LSTM with Hyper-Parameter Search", In: *Proc. of International Conf. On Industrial Applications of Big Data and Artificial Intelligence*, Shenzhen, China, Vol. 1756, 2021.

[28] B. Z. Aufa, S. Suyanto, and A. Arifianto, "Hyperparameter Setting of LSTM-based Language Model using Grey Wolf Optimizer", In: *Proc. of International Conf. On Data Science and Its Applications*, Bandung, Indonesia, pp. 1–5, 2020.

[29] I. Priyadarshini and C. Cotton, "A novel LSTM–CNN–grid search-based deep neural network for sentiment analysis", *International Journal of Supercomputing*, Vol. 77, No. 12, pp. 13911–13932, 2021.

[30] M. Cao, V. O. K. Li, and V. W. S. Chan, "A CNN-LSTM Model for Traffic Speed Prediction", *International Journal of IEEE Xplore*, pp. 1–5, 2020.

[31] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization", *International Journal of Machine Learning Research*, Vol. 13, pp. 281–305, 2012.

[32] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee, "Choosing multiple parameters for support vector machines", *International Journal of Machine Learning*, Vol. 46, pp. 131–159, 2002.

[33] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms", In: *Proc. of International Conf. On Neural Information Processing Systems*, New York, United States, pp. 2951–2959, 2012.

[34] Á. S. Illana, D. P. Guaita, D. C. García, J. D. S. Herráez, M. Vento, J. L. R. Cerdá, G. Quintás, and J. Kuligowski, "Model selection for within-batch effect correction in UPLC-MS metabolomics using quality control - Support vector regression", *International Journal of Analytica Chimica Acta*, Vol. 1026, pp. 62–68, 2018.

[35] L. L. Minku, "A novel online supervised hyperparameter tuning procedure applied to cross-company software effort estimation", *International Journal of Empirical Software Engineering*, Vol. 24, pp. 3153–3204, 2019.

[36] L. V. Arias and C. Q. López, "Comparative study of random search hyper-parameter tuning for software effort estimation", In: *Proc. of International Conf. On Predictive Models and Data Analytics in Software Engineering*, New York, United States, pp. 21–29, 2021.

[37] P. Phannachitta and K. Matsumoto, "Model-based software effort estimation - A robust comparison of 14 algorithms widely used in the data science community", *International Journal of Innovative Computing, Information and Control*, Vol. 15, No. 2, pp. 569–589, 2019.

[38] M. K. Kodmelwar, S. D. Joshi, and V. Khanna, "A Deep Learning Modified Neural Network Used for Efficient Effort Estimation", *International Journal of Computational and Theoretical Nanoscience*, Vol. 15, No. 11, pp. 3492–3500, 2018.

[39] M. Choetkiertikul, H. K. Dam, T. Tran, T. Pham, A. Ghose, and T. Menzies, "A Deep Learning Model for Estimating Story Points", *International Journal of IEEE Transactions on*

Software Engineering, Vol. 45, No. 7, pp. 637–656, 2019.

[40] E. M. D. B. Fávero, D. Casanova, and A. R. Pimentel, "SE3M: A Model for Software Effort Estimation Using Pre-trained Embedding Models", *International Journal of Information and Software Technology*, Vol. 147, No. 106886, 2022.

[41] T. Kangwantrakool, K. Viriyayudhakorn, and T. Theeramunkong, "Software Development Effort Estimation from Unstructured", *International Journal of Intelligent Information and Communication Technology and Its Applications to Creative Activity Support*, No. 4, pp. 739–747, 2020.

[42] T. E. Idriss, A. Idri, I. Abnane, and Z. Bakkoury, "Predicting Blood Glucose using an LSTM Neural Network", In: *Proc. of International Conf. On Computer Science and Information Systems*, Leipzig, Germany, pp. 35–41, 2019.

[43] D. Karmiani, R. Kazi, A. Nambisan, A. Shah, and V. Kamble, "Comparison of Predictive Algorithms : Backpropagation , SVM , LSTM and Kalman Filter for Stock Market", In: *Proc. of International Conf. On Amity International Conference on Artificial Intelligence*, Dubai, United Arab Emirates, pp. 228–234 , 2019.

[44] H. Wang, W. Zhuang, and X. Zhang, "Software Defect Prediction Based on Gated Hierarchical LSTMs", *International Journal of IEEE Transactions on Reliability*, Vol. 7, No. 2, pp. 711–727, 2021.

[45] P. S. Kumar, H. S. Behera, K. Anisha Kumari, J. Nayak, and B. Naik, "Advancement from neural networks to deep learning in software effort estimation: Perspective of two decades", *International Journal of Computer Science Review*, Vol. 38, p. 100288, 2020.

[46] Y. Li and H. Cao, "Prediction for Tourism Flow based on LSTM Neural Network", *International Journal of Procedia Computer Science*, Vol. 129, pp. 277–283, 2018.

[47] G. Jung and S. Y. Choi, "Forecasting foreign exchange volatility using deep learning autoencoder-LSTM techniques", *International Journal of Complexity*, Vol. 2021, 2021.

[48] D. M. Belete and M. D. Huchaiah, "Grid search in hyperparameter optimization of machine learning models for prediction of HIV/AIDS test results", *International Journal of Computers and Applications*, 2021.

[49] S. Lin, K. Ying, S. Chen, and Z. Lee, "Particle swarm optimization for parameter determination and feature selection of support vector machines", *International Journal of Expert Systems with Applications*, Vol. 35, pp. 1817–1824, 2008.

[50] G. Behera and N. Nain, "GSO-CRS: grid search optimization for collaborative recommendation system", *International Journal of Sadhana - Academy Proceedings in Engineering Sciences*, Vol. 47, No. 3 , pp. 1–12 , 2022.

[51] E. Balouji, I. Y. H. Gu, M. H. J. Bollen, A. Bagheri, and M. Nazari, "A LSTM-based deep learning method with application to voltage dip classification", In: *Proc. of International Conf. On Harmonics and Quality of Power*, Ljubljana, Slovenia, Vol. 2018-May, pp. 1–5 , 2018.

[52] F. B. Ahmad and L. M. Ibrahim, "Software Development Effort Estimation Techniques Using Long Short Term Memory", In: *Proc. of International Conf. On Computer Science and Software Engineering*, Duhok, Iraq, No. x , pp. 182–187, 2022.

[53] F. Qi, X. Y. Jing, X. Zhu, X. Xie, B. Xu, and S. Ying, "Software effort estimation based on open source projects: Case study of Github", *International Journal of Information and Software Technology*, Vol. 92, pp. 145–157, 2017.

[54] R. Marco, S. Sharifah, S. Ahmad, and S. Ahmad, "Bayesian Hyperparameter Optimization and Ensemble Learning for Machine Learning Models on Software Effort Estimation", *International Journal of Advanced Computer Science and Applications*, Vol. 13, No. 3, pp. 419–430, 2022..

[55] S. Shukla, S. Kumar, and P. R. Bal, "Analyzing effect of ensemble models on multi-layer perceptron network for software effort estimation", In: *Proc. of International Conf. On IEEE World Congress on Services*, Milan, Italy, pp. 386–387, 2019.

[56] A. Zakrani, M. Hain, and A. Namir, "Software development effort estimation using random forests: An empirical study and evaluation", *International Journal of Intelligent Engineering and Systems*, Vol. 11, No. 6, pp. 300–311, 2018, doi: 10.22266/ijies2018.1231.30.

[57] A. K. Bardsiri, S. M. Hashemi, and M. Razzazi, "Statistical Analysis of the Most Popular Software Service Effort Estimation Datasets", *International Journal of Telecommunication, Electronic and Computer Engineering*, Vol. 7, No. 1, pp. 87–96 , 2015.

[58] B. Vasilescu, A. Serebrenik, and T. Mens, "A historical dataset of software engineering conferences", In: *Proc. of International Conf. On Mining Software Repositories*, San Francisco, CA, USA, pp. 373–376 , 2013.

[59] S. Viaene, G. Dedene, and R. A. Derrig, "Auto claim fraud detection using Bayesian learning neural networks", *International Journal of Expert Systems with Applications*, Vol. 29, No. 3, pp. 653–666, 2005.

[60] A. G. P. Varshini, K. A. Kumari, D. Janani, and S. Soundariya, "Comparative analysis of Machine learning and Deep learning algorithms for Software Effort Estimation", In: *Proc. of International Conf. On Data Analytics, Intelligent Systems and Information Security*, Pollachi, India, Vol. 1767, No. 1, 2021.

[61] J. Keung, "Empirical Evaluation of Analogy-X for Software Cost Estimation" In: *Proc. of International Conf. On Empirical Software Engineering and Measurement*, New York, United States, pp. 294–296 , 2008.

[62] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering", *International Journal of Transactions on Software Engineering*, Vol. 28, No. 8, pp. 721–734, 2002.

[63] K. Maxwell, "Applied Statistics for Software Managers", *Prentice-Hall, Englewood Cliffs*, NJ, 2002.

[64] B. Kitchenham and S. Linkman, "Estimates, uncertainty, and risk", *International Journal of Software*, Vol. 14, No. 3, pp. 69–74, 1997.

[65] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization", In: *Proc. of International Conf. On Learning Representations*, San Diego, pp. 1–15, 2015.

[66] Y. Yang, K. Zheng, C. Wu, and Y. Yang, "Improving the classification effectiveness of intrusion detection by using improved conditional variational autoencoder and deep neural network", *International Journal of Sensors (Switzerland)*, Vol. 19, No. 11, 2019.

[67] Y. Bai, J. Xie, D. Wang, W. Zhang, and C. Li, "Computers & Industrial Engineering A manufacturing quality prediction model based on AdaBoost-LSTM with rough knowledge", *International Journal of Computers & Industrial Engineering*, Vol. 155, No. March, p. 107227, 2021.