Journal of Infrastructure, Policy and Development 2025, 9(4), 11428.
https://doi.org/10.24294/jipd11428

Article

Predicting time execution in fog computing using neural network and input evaluation with Interpretive Structural Modeling (ISM-ANN)

Ely Cheick Maaloum[1,*], Franklin Manene[2], Vitalice Oduol[3]

[1] Department of Electrical Engineering, Pan African University Institute for Basic Sciences, Technology and Innovation (PAUSTI), Jomo Kenyatta University Of Agriculture And Technology, Juja, Kiambu County 00200, Kenya
[2] Department of Electrical and Electronics Engineering, Dedan Kimathi University of Technology, Nyeri 10100, Kenya
[3] Department of Electrical and Information Engineering, University of Nairobi, Nairobi 00200, Kenya
[*] **Corresponding author:** Ely Cheick Maaloum, cheick.ely@students.jkuat.ac.ke

CITATION

Maaloum EC, Manene F, Oduol V. (2025). Predicting Time Execution in Fog Computing Using Neural Network and Input Evaluation with Interpretive Structural Modeling (ISM-ANN). Journal of Infrastructure, Policy and Development. 9(4): 11428. https://doi.org/10.24294/jipd11428

ARTICLE INFO

Received: 27 January 2025
Accepted: 24 March 2025
Available online: 19 December 2025

Abstract: Fog computing (FC) has been presented as a modern distributed technology that will overcome the different issues that Cloud computing faces and provide many services. It brings computation and data storage closer to data resources such as sensors, cameras, and mobile devices. The fog computing paradigm is instrumental in scenarios where low latency, real-time processing, and high bandwidth are critical, such as in smart cities, industrial IoT, and autonomous vehicles. However, the distributed nature of fog computing introduces complexities in managing and predicting the execution time of tasks across heterogeneous devices with varying computational capabilities. Neural network models have demonstrated exceptional capability in prediction tasks because of their capacity to extract insightful patterns from data. Neural networks can capture non-linear interactions and provide precise predictions in various fields by using numerous layers of linked nodes. In addition, choosing the right inputs is essential to forecasting the correct value since neural network models rely on the data fed into the network to make predictions. The scheduler may choose the appropriate resource and schedule for practical resource usage and decreased make-span based on the expected value. In this paper, we suggest a model Neural Network model for fog computing task time execution prediction and an input assessment of the Interpretive Structural Modeling (ISM) technique. The proposed model showed a 23.9% reduction in MRE compared to other methods in the state-of-arts.

Keywords: backpropagation algorithm; deep learning; fog computing; neural networks; resource allocation; time execution

1. Introduction

Fog computing extends cloud computing by shifting computation and storage closer to data sources, including IoT devices, sensors, and mobile units. While cloud computing employs primarily distant data centers for processing, fog computing is an approach that decentralizes computing by using the resources at the periphery of the network (Li, 2021). This architecture enables real-time data processing, which helps minimize latency and bandwidth utilization, making it ideal for smart cities, self-driving cars, and industrial IoT. It enables the deployment of fog nodes at the edge of the network, closeness to end users, mobility support, and real-time execution (Bitam, Zeadally, and Mellouk, 2018). Fog computing uses multiple layers of communication and can delegate tasks at various levels of the network(Arlot and Celisse, 2010). Predicting task execution time is essential for efficient resource allocation (Shahid et al., 2020) and improved task scheduling to the appropriate fog nodes (Gourisaria, Patra,

and Khilar, 2016). In addition, predicting the time needed by a task to be executed on a fog server could avoid the processing delays and reduce the need of the idle thus leads to energy effsiciency (Gourisaria, Patra, and Khilar, 2016). Prediction techniques are divided into three categories: analytical, simulation emulation, and empirical assessment(Ahmad and Qahmash, 2021). (Aboudib, Gripon, and Coppin, 2016) Introduced a machine learning model for efficient virtual resource selection in fog environments. This model enables improved workload prediction and resource allocation, identifying random forests as particularly effective. However, it faces limitations in overfitting and computational complexity, as well as a need for broader data diversity and real-world validation. (Attri, Dev, and Sharma, 2013) proposed a method to predict the performance of complex data applications within the fog computing paradigm, focusing on energy-efficient strategies like task offloading and load balancing. The study also explores renewable energy integration but notes limitations in scalability and complexity for diverse IoT workloads. (Meng et al., 2016) Presents a regression-based model to estimate resource utilization, reducing power consumption and improving task execution accuracy through a Two-Phase Regression (TPR) method. While it enhances resource allocation and speculative execution in Hadoop MapReduce, it struggles with irregular tasks and adds setup complexity due to Hadoop modifications, potentially impacting scalability. (Oliveira, Thomas, and Espadanal, 2014) Presented a job placement approach designed for hybrid high-performance computing environments. (Chang, Lin, and Chen, 2011) Presented an approach for predicting the execution time for parallel application scenarios. (Fan et al., 2014) Proposed an execution time prediction approach for effective CPU provisioning. Existing literature identifies several challenges in predicting task execution times in fog computing environments. Many current models experience issues with overfitting, high computational demands, and limited scalability across various IoT applications. For instance, while Nemirovsky's machine learning approach enhances workload prediction, it requires broader datasets for effective real-world application, and Avito’s task offloading strategy, though energy-efficient, encounters complexity in incorporating renewable energy. Additionally, regression-based methods, useful in resource allocation and task scheduling, show reduced accuracy with irregular tasks and face deployment challenges that affect scalability. This study aims to bridge these gaps by proposing an ISM-ANN prediction model, designed to improve task time prediction accuracy, resource efficiency, and ease of implementation in fog computing settings. We identified the suitable parameters for prediction model through ISM and proposed a neural network prediction model, which is classified as analytical modelling, is the first kind of prediction approach in fog computing context. The remainder of the paper is organized as follows: Section 2: identifies critical parameters in the execution task time prediction field. The approach and assessment standards for forecasting the model's accuracy are covered in Section 3. The suggested ISM-ANN prediction model is explained in Section 4. The simulation and validation results of the suggested model compared to other cutting-edge techniques are shown in Section 5. The conclusion and next steps to improve the suggested prediction model are presented in Section 6.

1.1 Problem statement

It is essential to accurately predict the time to complete a task in Fog Computing environments. Thus, efficient prediction helps the fog server Manager (SM) allocate resources more dynamically to ensure the tasks are executed on time and with minimum resource waste. The prediction of time execution of tasks mostly depends on the identification and selection of the inputs. Thus, an ISM method was proposed for identification and selection of the inputs.

1.2 Significance of the study

The main objective of this research is to propose a more reliable model for predicting time execution in Fog Computing environments. To achieve this, the study employs Interpretive Structural Modeling (ISM) to analyze and rank the critical input parameters. This systematic approach will assist in isolating the key parameters within the context that contributes to the time taken to complete a task. Subsequently, a Neural Network model will be created with the parameters that have been evaluated to generate precise time estimations. Finally, the study seeks to augment the findings from ISM by incorporating them into the Neural Network model with a view of improving its performance as a tool to facilitate optimal completion of tasks in the Fog Computing.

The main contributions of this paper are:

- Novel ISM-ANN Approach: The paper proposes a unique combination of ISM and ANN, where ISM is used to select the most impactful parameters, thereby improving the prediction accuracy of task execution time.
- Improved Prediction Accuracy: The proposed model achieves a 23.9% reduction in Mean Relative Error (MRE) compared to state-of-the-art methods, demonstrating its superiority in execution time estimation.
- Efficient Resource Allocation: Accurately predicting the time execution time of tasks, our approach enhances the task scheduling, reduces latency, and optimized resource usage in fog computing environments.
- Scalability and Performance: The model is validated using the GWA-13-METRANA dataset with 4,263,784 records and tested across 80 fog nodes, proving its scalability and applicability to real-world scenarios.
- Validation with Cross-Validation Techniques**:** The study employs K-fold cross-validation and multiple error metrics (MRE, RMSE, MMRE, MBRE) to ensure the robustness and reliability of the proposed prediction model.

1.3 Scope of the study

This study explores the use of Neural Networks (NN) to predict task execution times in fog computing environments, addressing the challenges of heterogeneity in devices and the need for low latency and real-time processing. In order to assess input factors for more precise forecasts and effective resource use, the study makes use of the Interpretive Structural Modelling (ISM) methodology. This study is a simulation-based analysis. The simulation employs 80 fog nodes, leveraging a subset of the GWA-13-METRANA dataset with approximately 4,263,784 records. The parameters identified and utilized in the neural network model include variables related to CPU

speed, memory usage, task deadlines, bandwidth, network throughput, and additional factors that influence task execution time in fog computing. Task scheduling and resource allocation in fog computing were improved by the suggested model, which was tested using K-fold cross-validation and showed a 23.9% decrease in mean relative error (MRE) when compared to other approaches. Due to its simulation setting, the study may have limitations in real-world generalizability, particularly when scaling to more diverse or unpredictable fog node configurations.

2. Identification of the input using ISM

The accuracy of a neural network model is heavily influenced by the input parameters provided to the network (Li, 2021). Initially, in the context of fog computing, various parameters are identified and compiled. Subsequently, a review of the literature reveals redundancy among these parameters, identified through correlation analysis. Finally, parameters with minimal impact on the prediction are eliminated.

3. Description of proposed model

This paper develops a robust Neural Network task time execution model in fog computing architecture. Thus, first, input parameters are identified using the ISM approach. Irrelevant variables are removed, and those with high correlation and impact are selected to enhance the accuracy of the neural network model in predicting task execution time on a fog node.

3.1. Interpretive Structural Model (ISM) approach

This paper develops a robust Neural Network task time execution model in fog computing architecture. Thus, firstly, an identification of the input parameters is achieved using ISM approach, therefore, some of irrelevant variables were removed and the ones with high correlations and impact were fed into a neural network model to improve its accuracy to predict the time execution of a task on a fog node

3.1.1. Steps involved in ISM

- Identify Variables: The parameters relevant to task execution in fog computing were identified (**Table 1**).
- Develop Structural Self-Interaction Matrix (SSIM): The SSIM measures the direct connections between variables, such as how Memory Required (P2) interacts with CPU Speed (P6), and is converted into a binary reachability matrix (Chang, Lin, and Chen, 2011).
- Reachability Matrix: The SSIM was converted into a binary reachability matrix, which helped identify direct and indirect influences between parameters. They also aid in developing the neural network model by recognizing the key features and enhancing the predictive accuracy.
- Level Partitioning: The variables were divided into hierarchical levels based on their influence, with parameters such as Memory Required (P2) and CPU Speed (P6) identified as the most impactful (**Table 2**).

- Draw the ISM graph: based on parameters levels and final reachability matrix a graph is generated.

Table 1. Prediction model parameter.

S.No	Parameters	Notation	References
1	Number of instructions	P_1	(Arlot and Celisse, 2010; Abdelaziz et al., 2018; Fan et al., 2014)
2	Memory required	P_2	(Arlot and Celisse, 2010; Shukla, Kumar, and Singh, 2021)
3	Input file size	P_3	(Shahid et al., 2020; Pham et al., 2017)
4	CPU speed	P_4	(Chang et al., 2011; Fan et al., 2014)
5	CPU usage	P_5	(Shukla et al., 2021)
6	CPU rate	P_6	(Shahid et al., 2020)
7	Output file size	P_7	(Bishop, 2006)
8	Deadline	P_8	(Arlot and Celisse, 2010)
9	Memory usage	P_9	(Shahid et al., 2020)
10	Bandwidth	P_{10}	(Pham et al., 2017)
11	Network throughput	P_{11}	(Pham et al., 2017)
12	Disk throughput	P_{12}	(Gourisaria et al., 2016)
13	Cost	P_{13}	(Shahid et al., 2020)

14	Delay	P14	(Shahid et al., 2020)

3.2. Time execution prediction

In distributed computing models such as Fog Computing, estimation of the time taken to execute tasks is beneficial in right allocation of resources to get optimum outcomes in terms of system performance (Hasteer, Bansal, and Murthy, 2017). The prediction of time execution is another aspect that has been addressed through different approaches ranging from statistical to heuristic approaches. The processing power available on the fog node (CPU Speed) and the available network bandwidth for transmitting data, influence task execution time in distributed systems, where faster CPU speeds and higher bandwidth can significantly reduce the time required for task completion (Shukla, Kumar, and Singh, 2019). A prediction model for task execution time prediction on a fog server is provided once the proper inputs have been chosen. A deep neural network (Oliveira, Thomas, and Espadanal, 2014) with backpropagation (Chang, Lin, and Chen, 2011) will be used. Techniques based on machine learning (ML) or deep learning (DL) are used to achieve the best outcomes effectively. These methods provide scheduling systems with more accurate models and reasoning conclusions. Additionally, heuristic algorithms' decision-making accuracy is increased by ML approaches (Sabireen and Neelanarayanan, 2021). Only tasks terminated before their deadlines is met may be carried out by virtual machines (VMs) due to restrictions imposed by deadline-based scheduling. A neural network uses a sequence of hidden layers to identify patterns between inputs and outputs. The neural network model is trained through backpropagation. As a result, each input data point is multiplied by the weight before being introduced into the hidden layers that contain neurons that process the data. The hidden layer forwards the information to the output layer, which uses a sigmoid function to compute the activation function and produces an output. The error, defined as the difference between the actual and predicted execution time of a task, is computed and propagated backward to adjust the weights. The error is calculated based on the activation function, the weight of neurons and the sum of weighted values (equation (1)):

$$\Delta \mathrm{Wgij} = \frac{\partial \mathbf{Error}}{\partial Activ_{funi}} * \frac{\partial \mathbf{Activ_{funi}}}{\partial \mathbf{Sum_{wgti}}} * \frac{\partial \mathbf{Sum_{wgi}}}{\partial Wgi} \quad (1)$$

ΔWgij: Represents the variation in the weight between neuron i and j.

$\frac{\partial \mathbf{Error}}{\partial Activ_{funi}}$: Represents the gradient of the error with respect to the activation function of neuron i.

$\frac{\partial \mathbf{Activ_{funi}}}{\partial \mathbf{Sum_{wgti}}}$ Derivative of the error with respect to the activation function.

$\frac{\partial \mathbf{Sum_{wgi}}}{\partial Wgi}$: Represents the gradient of the activation function of neuron i with respect to the weighted sum of inputs to that neuron.

After having the model trained, we then validate the accuracy of the proposed model with various well-known statistical parameters such as: Mean Relative Error (MRE), Root Mean Square Error (RMSE), and Mean Magnitude Relative Error to

Estimate (MMREE), Mean of Balanced Relative Error (MBRE), and finally Magnitude Relative Error (MMRE).

3.2.1. Mean relative error

Mean Relative Error measures the average relative difference between predicted execution time and actual execution time.

$$MRE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{t_{actual_i}-t_{predicted_i}}{t_{actual}}\right| \times 100 \quad (2)$$

Where n is the number of tasks, t_actual is the actual time execution of the task on fog server, and t_predicted is the predicted time execution of the task on the server.

3.2.2. Root mean square error deviation

Root Mean Square Deviation measures the square root of the average squared difference between predicted and actual execution times.

$$\text{RMSD} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(t_{actual_i} - t_{predicted_i})^2} \quad (3)$$

3.2.3. Mean magnitude relative error

MMRE is another form of relative error, focusing on the magnitude and giving an average relative difference between the actual and predicted execution times.

$$MMRE = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|t_{actual_i}-t_{predicted_i}\right|}{t_{actual_i}} \quad (4)$$

3.2.4. Mean magnitude relative error to estimate

MMRE evaluates the average magnitude of the relative error between the predicted and actual execution times. It is similar to MRE but focuses on the magnitude of the difference.

$$MMER = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|t_{actual_i}-t_{predicted_i}\right|}{t_{actual_i}} \times 100 \quad (5)$$

3.2.5. Mean of balanced relative error

MBRE balances the error by dividing the absolute difference by the maximum of the actual and predicted execution times. This ensures fair treatment of underestimation and overestimation.

$$MBRE = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|t_{actual_i}-t_{predicted_i}\right|}{\max\left(t_{actual_i},t_{predicted_i}\right)} \quad (6)$$

4. Methodology

Due to the fact that various tasks are sent from different devices in IoT layer, and there are heterogeneous fog servers in the fog computing configuration, so each task and node will have its own characteristics. Each task is characterized by its own arrival time (AT) the time of receiving it, deadline time (DT) which depends on the length of each task, and task size (TS) which reflects the number of instructions in it. On the other hand, the fog servers (FS) are characterized by the speed of the processor, the minimum and maximum time required to execute a task in the FS, minimum and

maximum cost required to execute a task in the FS, maximum length storage for the task in the FS, minimum time required to transfer a task in the FS, and energy constant to execute the task in the FS and so on. In this paper, we propose a Neural Network to predict the time execution of a task on FS. The accuracy of any ANN depends majorly the inputs fed. Therefore, we propose an ISM approach to define and select those inputs. Let assume a set of N tasks T = (T_1, T_2, T_3 ⋯.T_n) and a set of fog servers FS = (FS_1, FS2, FS_3⋯,FS_n) Predicting the time execution on each available fog server FSi is our primary goal in order to choose the best FS for actual execution. A subset of the prediction data for distinct tasks across diverse heterogeneous resources is used to train the model. The prediction model will be tested using the remaining dataset. Several statistical measures are calculated to assess the correctness of the model.

4.1. Selection of inputs parameters using ISM

We used ISM approach to select the appropriate input to be fed into the network. Various relevant inputs (P1, P2, …P14) were defined (**Table 1**). To form the Structural self-Interaction Matrix (SSIM), proportional relationships between the variable are determined based on experts' opinions and literature. Based on the relationship between the parameter Pi and Pj, different symbols (V, A, X, O) have been put in each cell:

- V: is placed in the cell if (P_i, P_j) if the parameter Pi directly influences Pj
- A: is placed in the cell if (P_i, P_j) if the parameter P_j directly influences P_i
- X: mutual influence, is placed in the cell (P_i, P_j) if both parameters (P_i and P_j) influence each other.
- O: no direct influence, is placed in the cell (P_i, P_j) if the parameters (P_i, P_j) are not directly connected

(**Table 2**) shows the structural self-Interaction Matric between the parameters and how do they influence each other's. By eliminating transitive relationships between parameters, the original reachability matrix is converted into the final reachability matrix (**Table 3**). If parameter Pi is linked to Pj and Pj is related to Pk, then Pi should likewise be related to Pk. This is known as a transitive dependence. The interplay of the reachability set (Ri) and antecedent set (Ai) is then used to split this matrix into levels. Whereas Ai contains both the parameter and the parameters that may affect it, Ri includes both the parameter and the other parameters it can affect. (**Figure 1**) displays the interdependency and connection among variables, which helps to reduce the transitive dependency in later steps. (**Figure 2**) presents the detailed directed graph constructed from the Final Reachability Matrix (FRM), highlighting the directional influence among the parameters P1–P14. The diagraph shows the driving and dependence relationships explicitly, including self-loops representing internal influence and directed edges showing how each parameter propagates its effect through the system. A hierarchical structure is established using the final level partitioning, which helps identify independent issues and maps their interdependencies based on their respective levels within the structural model. (**Table 4**) represents the cluster matrix, which is derived by the analyzing the (FRM) and categorizing the parameters into different clusters based on their Driving Power (DrP) and Dependence Power (DePr). The classification follows these steps:

1. Identifying the key Variables:
   - Independent variables (Drivers): High DrP, low DePr (Strong influence, less influenced)
   - Linkage variables: High Drp, High DePr (both influence/influenced)
   - Dependent variables: Low DrP, High DePr (highly influenced, weak influencers).
   - Autonomous variables: Low DrP, Low DePr (weak interaction).
2. Clustering based on Influence-dependence relationship:
   - Sort parameters by their DrP and DePr
   - Assign them to the cluster based on thresholds

Table 2. Structural self interaction matrix.

P_i	P_{14}	P_{13}	P_{12}	P_{11}	P_{10}	P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	P_1
P_1	V	V	X	V	X	X	X	V	V	X	A	A	O	
P2	V	X	X	X	X	X	X	X	X	V	O	V		
P3	V	X	V	V	V	V	V	X	X	X	A			
P4	V	X	V	V	V	V	X	X	X	X				
P5	A	V	A	A	A	A	A	A	A					
P6	V	X	O	O	X	V	X	X						
P7	V	X	O	O	X	V	X							
P8	V	X	X	O	X	V								
P9	V	X	O	O	V									
P10	X	X	V	V										
P11	X	X	V											
P12	X	X												
P13	X													
P14														

Table 3. Final Reachability Matrix (FRM).

Parameters	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Driving Power(Drp
P_1	1	1*	1*	1	1	0	0	1*	1*	1	1*	0	1*	0	10
P_2	1	1	1	1	1	1*	1*	1	1	1	1*	1*	0	1*	13
P_3	1	1*	1*	1	1	1*	1*	1	0	1	0	1	1	1	12
P_4	1	0	0	1	1	1*	1*	1	1*	1	1	1	1*	1	12
P_5	1	0	0	1*	1	0	1*	1	1*	1	1	1	0	1	10
P_6	1*	0	0	1*	0	1	1*	1	1*	1	1*	1	1*	1*	11
P_7	0	0	0	1*	1*	0	1	0	0	0	0	0	0	0	3
P_8	1*	0	0	1*	0	0	0	1	0	1	1*	0	0	0	5
P_9	0	0	0	1	1	0	1*	1*	1	1	0	1	1	1	7
P_{10}	0	0	0	1*	1*	0	0	1*	0	1	0	0	0	0	4
P_{11}	1*	1*	0	1*	1*	0	0	1*	0	1	1	1*	1*	1	7
P_{12}	0	0	0	0	0	0	1*	0	0	0	0	1	0	1*	3
P_{13}	1	0	0	1	1	0	1*	1*	1*	0	0	1*	1	1	7
P_{14}	0	0	0	0	1*	0	0	0	0	0	0	1*	0	1	3

Dependence Power(Depr)	11	4	3	12	11	4	9	11	9	10	10	10	9	10

Table 4. Extraction of driving dependence powers from FRM.

Natation	**Parameters**	**Driving Power (DrP)**	**Dependence Power(DePr)**
p_1	Number of instructions	10	11
P_2	Memory required	13	4
P_3	Input file size	12	3
P_4	CPU speed	12	12
P_5	CPU usage	10	11
P_6	CPU rate	11	4
P_7	Output file size	3	9
P_8	Deadline	5	11
P_9	Memory usage	7	9
P_{10}	Bandwidth	4	10
P_{11}	Network throughput	7	10
P_{12}	Disk throughput	3	10
P_{13}	Cost	7	9
P_{14}	Delay	3	10

Table 5. Clustering of parameters into categories.

Cluster	**Criteria**	**Parameters**
Independent	High DrP, Low DePr	P2, P3, P6
Linkage	High DrP, High DePr	P1, P4, P5
Dependent	Low DrP, High DePr	P7, P8, P10, P12, P14, P9, P11, P13
Autonomous	Low DrP, Low DePr (Weak Interaction)	

Table 6. Final cluster matrix.

Natation	**Parameters**	**Driving Power (DrP)**	**Dependence Power(DePr)**	**Cluster**
p_1	Number of instructions	10	11	Linkage
P_2	Memory required	13	4	Independent
P_3	Input file size	12	3	Independent
P_4	CPU speed	12	12	Linkage
P_5	CPU usage	10	11	Linkage
P_6	CPU rate	11	4	Independent
P_7	Output file size	3	9	Dependent
P_8	Deadline	5	11	Dependent
P_9	Memory usage	7	9	Dependent

P10	Bandwidth	4	10	Dependent
P11	Network throughput	7	10	Dependent
P12	Disk throughput	3	10	Dependent
P13	Cost	7	9	Dependent
P14	Delay	3	10	Dependent

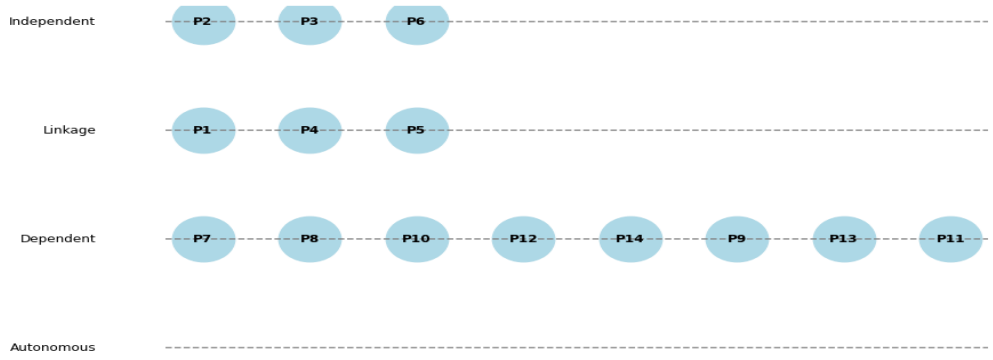

Figure 1. Hierarchical architecture.

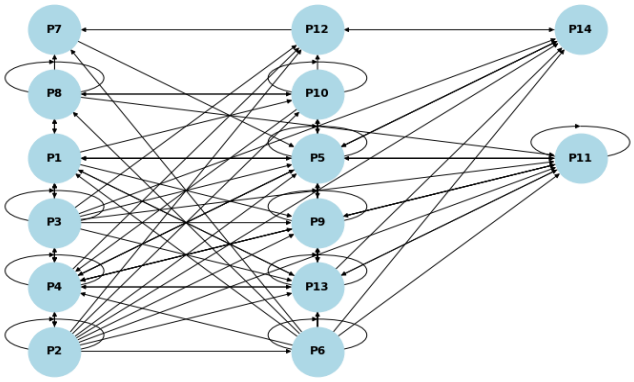

Figure 2. Parameters diagraph.

The lack of an autonomous variable as in the (**Table**s **5** and **6**) indicates that every parameter affects the system in some way. The parameters P_2, P_3, and P_6 have high Driving Power (DrP) values and high Dependence Power (DePr) indicating that they significantly influence other parameters within the system. However, they are not significantly influenced by other parameters. Parameters P_1, P_4, and P_5 are linkage parameters, which are highly influential and highly dependent, indicating that dynamic optimization of the system's overall performance will result from their proper selection. On the other hand, P_7, P_8, P_9, P_{10}, P_{11}, P_{12}, P_{13}, and P_{14} dependent parameters, meaning they are affected by the performance of other parameters in the model.

4.2 Artificial neural network model

Three parameters are selected as inputs with strong prediction power of accurate output assessed using the ISM approach: Number of Instruction (P1), CPU speed (P4), and CPU usage (P5). To predict the task time execution on a server, the data is fed into a neural network (equation (7)):

$$\boldsymbol{t}_{\boldsymbol{predicted}_i} = \frac{\boldsymbol{Number\ of\ Instruction}}{\boldsymbol{CPU}_{\boldsymbol{speed}} \times \boldsymbol{CPU\_usage}} \quad (7)$$

The data undergoes normalization and is transformed into a linear format before being input into the network. This process involves applying the natural logarithm and adjusting the values using specific coefficients (equation (8)):

$$\ln\left(\boldsymbol{t}_{\boldsymbol{predicted}}\right) = lnA + \ln(P_1) - \ln(P_4) - \ln(P_5) \quad (8)$$

The optimized logarithm transformation function of the predicted execution time is defined in (equation (9))

$$OTPT = W0 + W1 \times P'_1 + W2 \times P'_2 + W3 \times P'_3 \quad (9)$$

Where $\begin{cases} P_{'1} = \ln(P_1) \\ P'_2 = \ln(P_2) \\ P'_3 = \ln(P_4) \end{cases}$ and $\begin{cases} W0 : \ln(A) \\ W1 = W2 = W3 = 1 \end{cases}$

The Wi are the weight coefficients which are all initialized with 1. The Optimized Time Prediction of Task (OTPT) function is then compared to the Actual Time Execution (ATE) to determine the prediction error. We tried to minimize this error by updating the weights using the backpropagation algorithm (equation (10))

$$W'_i = W_i + E_i \quad (10)$$

Where W'_i is the updated weights and E_i is the prediction error. The optimization is repeated until acquiring the least error possible.

The model is designed with four input parameters, three hidden layers, and a single output. All inputs are processed through the hidden layers, which are interconnected to facilitate training on a given dataset. A sigmoid activation function is applied within the hidden layers to introduce nonlinearity into the data. The output layer generates a single value representing the Predicted Execution Time (PET) for each resource as shown in (**Figure 3**). Once the training phase is completed, the model is able to predict task execution times as illustrated in (**Figure 4**). Test datasets are fed into the model to validate its predictions, and the process continues until the mean square error (MSE) approaches zero. (Appendex 1.2) states the training steps in detail. To assess the model's accuracy, various performance metrics, including Mean Relative Error (MRE), Root Mean Square Error (RMSE), Mean Magnitude of Relative Error (MMRE), Mean Magnitude of Error Rate (MMER), and Mean Balanced Relative Error (MBRE), are calculated. The effectiveness of the proposed model is evaluated by comparing its performance with existing approaches.

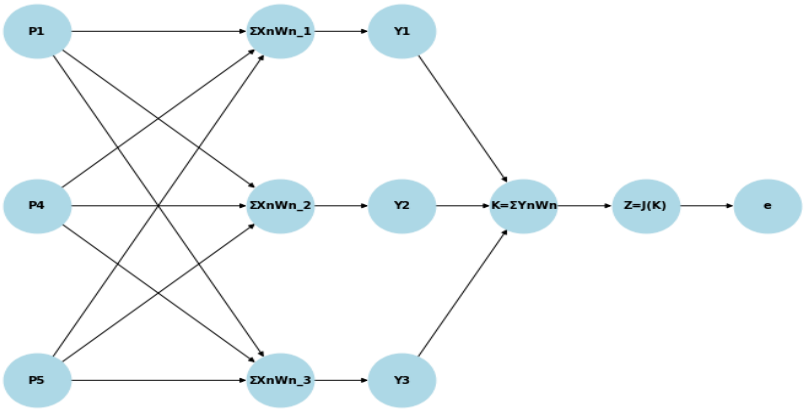

Figure 3. Proposed ANN Time Execution Model.

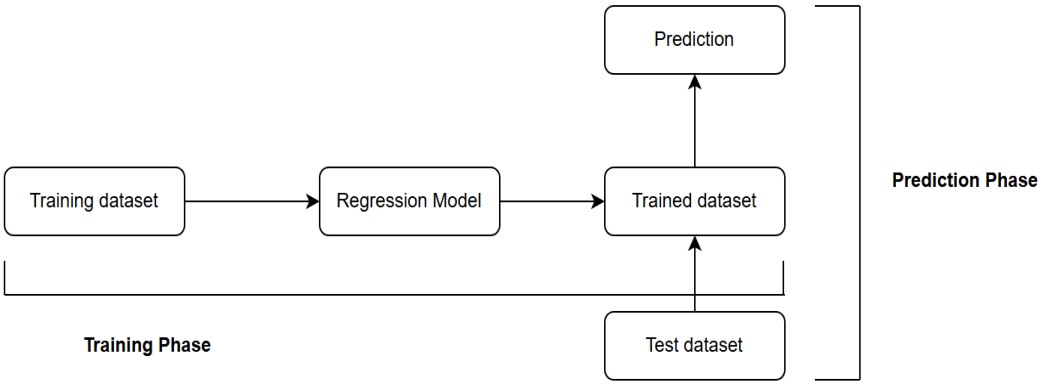

Figure 4. Proposed ANN Training and Testing.

5. Results and discussion

To validate the proposed model's performance, simulations were conducted using Python, and the outcomes were compared against those of an existing approach. The simulations were executed on a Linux (Ubuntu 24.04.1 LTS) operating system, running on an Intel® Core™ Ultra 7 155H with 16 GB of RAM and a 500 MB SDD. The GWA-13-METRANA dataset was used to train and validate the prediction model. Specifically, a subset of the dataset was utilized to record actual execution times, which were then used to train the model. The remaining portion of the dataset was reserved for testing the model's predictive capabilities. Various values and ranges of parameters were also considered during this process. The characteristics of the model and other parameters are shown in (**Table 7**).

Table 7. parameters used in the experiment.

Type	Parameter	Rang
	hidden layers	03
Artificial Neural Network	Learning rate	0.01
	Number of epochs	27000
Tasks	Number of tasks	7000
Fog servers	Number of Nodes	80

An 80% subset of the GWA-13-METRANA dataset is used to train the artificial neural network, while the remaining 20% is used for testing. **Figure 3** shows the graph between task predicted and actual time. The **Figure 3** indicates that there is a noticeable peak in both the actual and predicted execution times around 2000 milliseconds, where the execution time reaches around 700 units. Initially, the predicted values (dashed red line) closely follow the actual values (solid blue line), but there is a significant spike around 2000ms. After this point, the predicted and actual execution times converge and show minimal difference as the execution time increases beyond 2000ms, exhibiting a steady trend. Each dataset was subjected to K-Fold Cross validation after being divided into five equal training shanks. (**Table 8**) displays the cross-validation findings.

The ISM-ANN proposed prediction model is contrasted with linear regression (Chang, Lin, and Chen, 2011) and (Abdelaziz et al., 2018). A prediction-based method for resource selection was introduced by (Duong et al., 2016). In comparison to their findings, which were 15.49% and 8.14% MRE, our proposed model reaches up to 5.68% MRE. A Comparison between the prediction model and current methods in terms of MMER, MMRE, RMSE, and MRE. MRE is addressed in the (**Table 9**).

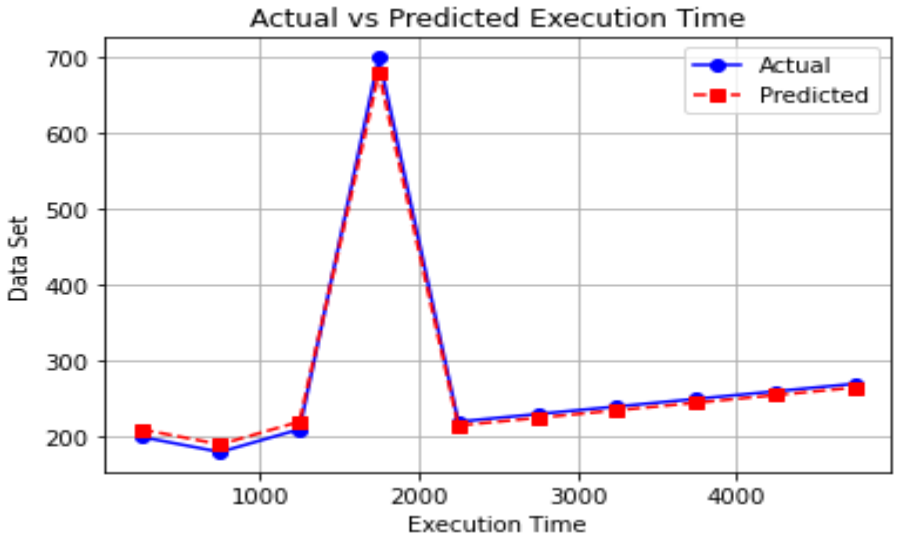

Figure 3. Task actual and predicted execution time comparison.

Table 8. Performance metrics table.

Performance metric	Proposed Execution model	(Chang, Lin, and Chen, 2011)	(Abdelaziz et al., 2018)
MRE	5.68	9.53	8.14
RMSE	13.2	20.3	14.22
MMER	0.069	0.11	0.081
MMER	0.087	0.126	0.106
MBRE	0.0379	0.273	0.505

Table 9. Comparative analysis of prediction model with existing approaches.

Metrics	Training set	Folder (f1)	Training set	Folder (f2)	Training Set	Folder (f3)	Training set	Folder (f4)	Training Set	Folder (f5)
MRE	6.07	9.9	8.3	3.12	8.2	3.29	6.7	7.91	7.9	4.17
RMSE	13.3	12.9	13.1	13.2	13.0	13.1	12.8	13.4	13.1	13.12
MMRE	0.070	0.068	0.072	0.066	0.071	0.068	0.067	0.069	0.068	0.069
MMER	0.087	0.085	0.088	0.086	0.087	0.089	0.084	0.086	0.088	0.087
MBRE	0.039	0.036	0.038	0.038	0.037	0.037	0.036	0.038	0.037	0.0379

6. Conclusion

In this paper, we proposed an ISM-ANN-based approach for predicting task execution time in fog computing environments, addressing the challenge of workload management in distributed networks. By leveraging Interpretive Structural Modeling (ISM) for optimal input selection and a Neural Network (ANN) for execution time prediction, our model significantly enhances accuracy, achieving a 23.9% reduction in Mean Relative Error (MRE) compared to state-of-the-art techniques. This improvement facilitates more efficient task scheduling, optimized resource utilization, and reduced latency, contributing to the advancement of fog computing architectures. Despite these advancements, our study has some limitations, including reliance on simulations, scalability challenges, dataset dependency, and a lack of energy-efficient considerations. Future work will focus on real-world deployment, scalability optimization, energy-aware scheduling, adaptive learning mechanisms, and hybrid modeling techniques. These enhancements will further improve execution time prediction accuracy, making fog computing more efficient and sustainable for large-scale distributed applications.

Author contributions: Conceptualization was carried out by Ely Cheick Maaloum, Franklin Manene, and Vitalice Oduol; methodology was developed by Ely Cheick Maaloum and Franklin Manene; software was implemented by Ely Cheick Maaloum; validation was performed by Ely Cheick Maaloum, Franklin Manene, and Vitalice Oduol; formal analysis and data curation were done by Ely Cheick Maaloum; investigation and visualization were done by Ely Cheick Maaloum;

resources were provided by Franklin Manene and Vitalice Oduol; the original draft was written by Ely Cheick Maaloum, with review and editing by Franklin Manene and Vitalice Oduol; supervision and project administration were handled by Franklin Manene and Vitalice Oduol. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the African Union (AU) through the Pan African University Institute for Basic Sciences, Technology and Innovation (PAUSTI).

Acknowledgment: Here, you can acknowledge any support given which is not covered by the author contribution or funding sections. This may include administrative and technical support, or donations in kind (e.g., materials used for experiments).

Conflict of interest: The authors declare no conflict of interest.

References

Li, R. (2021). Use Linear Weighted Genetic Algorithm to Optimize the Scheduling of Fog Computing Resources. Complexity, 2021(1). Portico. https://doi.org/10.1155/2021/9527430

Bitam, S., Zeadally, S., & Mellouk, A. (2017). Fog computing job scheduling optimization based on bees swarm. Enterprise Information Systems, 12(4), 373–397. https://doi.org/10.1080/17517575.2017.1304579

Pham, T. P., Durillo, J. J., & Fahringer, T. (2017). Predicting workflow task execution time in the cloud using a two-stage machine learning approach. IEEE Transactions on Cloud Computing, 21(1), 1–13.

Arlot, S., & Celisse, A. (2010). A survey of cross-validation procedures for model selection. Statistics Surveys, 4, 40–79. https://doi.org/10.1214/09-ss054

Shahid, M. H., Hameed, A. R., ul Islam, S., et al. (2020). Energy and delay efficient fog computing using caching mechanism. Computer Communications, 154, 534–541. https://doi.org/10.1016/j.comcom.2020.03.001

Gourisaria, M. K., Patra, S. S., & Khilar, P. M. (2016). Minimizing energy consumption by task consolidation in cloud centers with optimized resource utilization. International Journal of Electrical and Computer Engineering, 6(6), 3283–3292. https://doi.org/10.11591/ijece.v6i6.12251

Ahmad, N., & Qahmash, A. (2021). SmartISM: Implementation and Assessment of Interpretive Structural Modeling. Sustainability, 13(16), 8801. https://doi.org/10.3390/su13168801

Artificial Neural Networks and Machine Learning—ICANN 2016. (2016). In A. E. P. Villa, P. Masulli, & A. J. Pons Rivero (Eds.), Lecture Notes in Computer Science. Springer International Publishing. https://doi.org/10.1007/978-3-319-44781-0

Attri, R., Dev, N., & Sharma, V. (2013). Interpretive structural modelling (ISM) approach: An overview. Research Journal of Management Sciences, 2(2).

Meng, X., Bradley, J., Yuvaz, B., et al. (2016). MLlib: Machine learning in Apache Spark. Journal of Machine Learning Research, 17(1), 1–7.

Oliveira, T., Thomas, M., & Espadanal, M. (2014). Assessing the determinants of cloud computing adoption: An analysis of the manufacturing and services sectors. Information & Management, 51(5), 497–510. https://doi.org/10.1016/j.im.2014.03.006

Chang, R.-S., Lin, C.-F., & Chen, J.-J. (2011). Selecting the most fitting resource for task execution. Future Generation Computer Systems, 27(2), 227–231. https://doi.org/10.1016/j.future.2010.09.003

Fan, Y., Wu, W., Xu, Y., et al. (2014). Improving MapReduce performance by balancing skewed loads. China Communications, 11(8), 85–108. https://doi.org/10.1109/cc.2014.6911091

Shukla, A., Kumar, S., & Singh, H. (2020). MLP-ANN-Based Execution Time Prediction Model and Assessment of Input Parameters Through Structural Modeling. Proceedings of the National Academy of Sciences, India Section A: Physical Sciences, 91(3), 577–585. https://doi.org/10.1007/s40010-020-00695-9

Duong, T. N. B., Zhong, J., Cai, W., et al. (2016). RA2: Predicting Simulation Execution Time for Cloud-Based Design Space Explorations. In Proceedings of the 2016 IEEE/ACM 20th International Symposium on Distributed Simulation and Real Time Applications (pp. 120–127). https://doi.org/10.1109/ds-rt.2016.9

Bishop, C. M. (2006). Pattern recognition and machine learning. Springer.

Hasteer, N., Bansal, A., & Murthy, B. K. (2017). Assessment of cloud application development attributes through interpretive structural modeling. International Journal of System Assurance Engineering and Management, 8(2), 1069–1078. https://doi.org/10.1007/s13198-017-0571-2

Ramesh, V. P., Baskaran, P., Krishnamoorthy, A., et al. (2019). Back propagation neural network based big data analytics for a stock market challenge. Communications in Statistics - Theory and Methods, 48(14), 3622–3642. https://doi.org/10.1080/03610926.2018.1478103

Shukla, A., Kumar, S., & Singh, H. (2019). Fault tolerance based load balancing approach for web resources. Journal of the Chinese Institute of Engineers, 42(7), 583–592. https://doi.org/10.1080/02533839.2019.1638307

Sabireen, H., & Neelanarayanan, V. (2021). A review on fog computing: Architecture, fog with IoT, algorithms and research challenges. ICT Express, 7(2), 162–176. https://doi.org/10.1016/j.icte.2021.05.004

Chang, R.-S., Lin, C.-F., & Chen, J.-J. (2011). Selecting the most fitting resource for task execution. Future Generation Computer Systems, 27(2), 227–231. https://doi.org/10.1016/j.future.2010.09.003

Abdelaziz, A., Elhoseny, M., Salama, A. S., et al. (2018). A machine learning model for improving healthcare services on cloud computing environment. Measurement, 119, 117–128. https://doi.org/10.1016/j.measurement.2018.01.022

Bitam, S., Zeadally, S., & Mellouk, A. (2018). Fog computing job scheduling optimization based on bees swarm. Enterprise Information Systems, 12(4), 373–397. https://doi.org/10.1080/17517575.2017.1304579

Appendix 1

1. K-Fold Cross Algorithm.

Input: Data, Labels, Model, K (number of folds) K= 5

Output: Final Average Accuracy

1. **Divide** Data into K equal folds: fold_1, fold_2, ..., fold_K

2. **Initialize** accuracy_list = []

3. **For each** fold k = 1 to K:

   Set Validation_Set = fold_k

   Set Training_Set = Data - fold_k`

   Train Model on Training_Set

   Predict on Validation_Set

   Calculate Accuracy = accuracy(Predictions, Validation_Set_Labels)

   f. Append Accuracy to accuracy_list

4. Final_Accuracy = mean(accuracy_list)

5. **Return** Final_Accuracy

2. Backpropagation Training Algorithm.

Input: Task (T) with Number of instructions (NI) CPU Usage (CU), CPU Speed (CS) From Dataset(DS), and Fog Servers (FS)**Output:** Predicted Execution Time (PET) for T on FS

1. **Initialize** the weights (W_i) for inputs variables and set the learning rate ($0 < ṇ < 1$).
2. **Repeat** steps 3 to 10 until convergence.
3. **For each** record in DS, perform the following:
    4. **Calculate** the Predicted Execution Time $\boldsymbol{t_{predicted}}$as:

$$OTPT = \ln(A) + W \times \ln(NI) - W \times \ln(CS) - W \times \ln(1 - CU)$$

    5. **Calculate** the Error (E) as:

E= $OAET_1 - OTPT_1$ where $OAET_1 = \ln(\boldsymbol{t_{actual}})$ and $OTPT_1 = \ln(\boldsymbol{t_{predicted}})$

    6. **Update** the weights using gradient descent:
        - $dw1 = \eta \times \frac{dE}{dWgtij} = E \times \ln(NI)$
        - $dw1 = \eta \times \frac{dE}{dWgtij} = E \times \ln(CS)$
        - $dw1 = \eta \times \frac{dE}{dWgtij} = E \times \ln(1\text{-}CU)$
    7. **Compute** the bias update as:

$$db = \eta \times \frac{dE}{dWgtij}$$

    8. **Recalculate** the weights:

$$w_i = w_i + dw_i;$$

    9. **Recalculate** the bias:

$$b = b + db$$

4. **Repeat** this steps until the error reaches the desired minimum level.