Particle Swarm Optimization (PSO) Long Short-Term Memory Network Based Cloud Resource Usage Prediction Model for Cloud Data center

Ayantu Tesfaye Kenea

A Thesis Submitted to The Department of Computer Science and Engineering
School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the degree of masters in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

June 2022

Adama, Ethiopia

Particle Swarm Optimization (PSO) Long Short-Term Memory Network Based Cloud Resource Usage Prediction Model for Cloud Data center

Ayantu Tesfaye Kenea

Advisor: Dr. Biruk Yirga

A Thesis Submitted to Department of Computer Science and Engineering School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the degree of masters in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

June 2022

Adama, Ethiopia

DECLARATION

I hereby declare that this Master Thesis entitled "**Particle Swarm Optimization (PSO) Long Short-Term Memory Network Based Cloud Resource Usage Prediction Model for Cloud Data center**" is my original work. That is, it has not been submitted for the award of any academic degree, diploma or certificate in any other university. All sources of materials that are used for this thesis have been duly acknowledged through citation.

Ayantu Tesfaye Kenea	_____	_____
Name of student	Signature	Date

RECOMMENDATION OF ADVISORS/ SUPERVISORS

I, the major advisor/supervisor of this research thesis, hereby certify that I have closely advised/supervised the student while developing this thesis and read the draft thesis/dissertation entitled "**Particle Swarm Optimization (PSO) Long Short-Term Memory Network Based Cloud Resource Usage Prediction Model for Cloud Data center"** prepared under my guidance by Ayantu Tesfaye Kenea. Therefore, I recommend the submission of the thesis to the department for further review and evaluation.

Major Advisor/Supervisor _____ _____ _____

Signature Date

Co-advisor/Co-supervisor _____ _____ _____

Signature Date

APPROVAL PAGE

I/we, the advisors of the thesis entitled "**Particle Swarm Optimization (PSO) Long Short-Term Memory Network Based Cloud Resource Usage Prediction Model for Cloud Data center**" and developed by *Ayantu Tesfaye Kenea*, hereby certify that the recommendation and suggestions made by the board of examiners are appropriately incorporated into the final version of the thesis.

_____	_____	_____
Major Advisor	Signature	Date
_____	_____	_____
Co-advisor	Signature	Date

We, the undersigned, members of the Board of Examiners of the thesis by *Ayantu Tesfaye Kenea* have read and evaluated the thesis entitled "**Particle Swarm Optimization (PSO) Long Short-Term Memory Network Based Cloud Resource Usage Prediction Model for Cloud Data center**" and examined the candidate during open defense. This is, therefore, to certify that the thesis is accepted for partial fulfillment of the requirement of the degree of Master of Science in Computer Science and Engineering.

_____	_____	_____
Chairperson	Signature	Date
_____	_____	_____
Internal Examiner	Signature	Date
_____	_____	_____
External Examiner	Signature	Date

Finally, approval and acceptance of the thesis is contingent upon submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and School Graduate Committee (SGC).

_____	_____	_____
Department Head	Signature	Date
_____	_____	_____
School Dean	Signature	Date
_____	_____	_____
Office of Postgraduate Studies, Dean	Signature	Date

ACKNOWLEDGMENT

I'd like to express my gratitude to the almighty God for providing me with everything I require in life. Because I owe it everything to you both, my life coach, my mother Shitaye Abe, and my father Tesfaye Kenea. Thank you very much! I am grateful to my husband Habte Lejebo for his moral and emotional support throughout my life. I'm also grateful to my other family members and friends for their continued support.

A particular thank you goes out to everyone, especially my adviser, Dr. Biruk Yirga, for his encouragement and support in the form of helpful remarks and recommendations. I will never forget his encouragement and, more importantly, his prompt support and advice till the thesis was completed. I'd like to express my gratitude to the member of the university staff: For his unwavering support and assistance, Dr.-Ing Frezwed Lemma. Finally, but certainly not least, to everyone in the (Cloud) SIG lab, it has been a pleasure sharing the lab with all of you over the previous two years. Thank you so much for your support!

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATION

ARIMA	Autoregressive Integrated Moving Average
CDC	Cloud Data Center
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
GRU	Gated Recurrent Unit
LSTM	Long-Short-Term Memory
MAE	Mean Absolute Error
MSE	Mean Squared Error
PSO	Particle Swarm Optimization
QoS	Quality of service
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SARIMA	Seasonal Auto Regressive Integrated Moving Average
SLA	Service Level Agreement
VM	Virtual Machine

Abstract

The deployments of cloud data centers are growing at an exponential rate. As the excessive use of computer resources and the accompanying energy and environmental impacts have a close correlation and therefore it is vital to be more concerned and careful about the emission of carbon footprints while improper usage of data center resources takes place. Virtual machine consolidation techniques are used by cloud service providers to efficiently manage computer resources and reduce energy consumption. Aggressive virtual machines (VM) consolidation, on the other hand, may result in service performance reduction and, as a result, serious violations of SLA. The most important challenge in appropriately managing the cloud resources such as memory and central processing units while limiting energy consumption is the accurate forecasting of Virtual Machine (VM) workloads. Various researches have been done to manage cloud resources and lowering resource costs through VM workload prediction using deep learning techniques. However, workload prediction accuracy still needs improvement. This research paper proposes a hybrid particle swarm optimization (PSO) and state-of-the-art long short-term memory (LSTM) deep learning model to solve this challenge. The proposed prediction model has been efficiently optimized using the PSO, allowing it to better forecast incoming workloads. To compare various metrics and extend the applicability of the model, the method first employs multivariate analysis to derive the sequential and contextual properties of historical workload data using the PSO based LSTM network from the original high-dimensional workload data. The model then combines the various hyperparameters (optimizers, metrics, and losses), and layers (Normalization, Recurrent, and Activation layers) to achieve accurate workload prediction. The experiment was done on a Bitbrain workload trace to show that the proposed method had a higher prediction accuracy. The Root Mean Squared Error (RMSE) findings show that the PSO-LSTM model beats state-of-the-art workload prediction approaches such as CNN, CNN-LSTM and LSTM in terms of VM workload forecasting accuracy in cloud computing environments up.

***Keywords**: Virtual Machine (VM), Workload, PSO, LSTM, Cloud, Datacenter*

CHAPTER ONE

INTRODUCTION

1.1 Background of the Study

Green technology and sustainability are now two of the most widely discussed topics. Massive climate change in recent years is one of the most significant and visible indicators that the world is rapidly becoming ill. Saving the atmosphere, and eventually the earth, is critical. Despite the benefits that information communication technology (ICTs) provides, they also contribute to environmental issues by consuming massive amounts of energy and emitting carbon dioxide (Shuja et al., 2017). Green computing is an evolving area in which problems such as the effect of information technology on climate change (i.e., global warming), cloud data center resource use, and other relevant challenges are being investigated and studied. Therefore, managing computing resources has been the most critical task for the ICT industry such as cloud computing service providers.

The Cloud computing is a utility computing model which is convenient to access the pool of computing resources such as physical machines, servers, applications, computing, storage, networks and various other services. The cloud computing model provides majorly three services such as software, platform and infrastructure as a service based on pay per usage model. The elasticity feature of cloud computing enables the users to dynamically change the resource request periodically based on the demand. Due to fluctuating demands, the cloud manager must be able to leverage resources by provisioning and de-provisioning the resources to meet the current request (Anupama et al., 2021). Under-provisioning causes Service Level Agreements (SLA) violation, Quality of Service (QoS) dropping and the customer dissatisfaction. This may lead to the loss of customers and a decrease in revenue. On the other hand, Over- provisioning wastes energy and resources and it even increases costs like network, cooling and maintenance. Therefore, the resources management is a complicated process in cloud and an efficient resource management technique is required (Masdari & Khezri, 2020).

In order to increase resource usage and decrease energy consumption, virtual machine consolidation is the most commonly used technique in cloud data centers. However, Incorrect VM

consolidation, on the other hand, can result in frequent live migration, resulting in serious quality of service deterioration and resource overhead. The most difficult aspect of VM consolidation is accurately predicting future VM workloads. Accurate VM workload estimation leads to sufficient virtual machine resource allocation for future workloads.

Consequently, accurate workload prediction is a crucial factor in conducting effective proactive resource management schemes and allocating on-demand resources to the user requests (Masdari & Khoshnevis, 2020). Every application requires resources to complete its execution and these resources comes in the virtual form. The Cloud Data Centers (CDC) comprises of various resources like CPU, Memory, bandwidth, software etc., which are allocated to the users on demand to complete their task execution. As per the previous works, it is well noted that resources provisioned to execute an application is always greater than actual resources required to complete it (Anupama et al., 2021).

The vital need to precisely estimate virtual machine workloads is one of the most pressing concerns facing cloud computing in terms of service provisioning (VMs). This will be advantageous in terms of resource allocation efficiency, as the consumer will only have to pay for what they require and utilize.

To anticipate future values, time series analysis can be utilized to detect a repeated pattern in input workload (Ruan et al., 2021). As a result, it might be used to forecast cloud resources. Time series analysis techniques based on machine learning, such as regression and neural networks, are one type of time series analysis that focuses on direct future prediction (Bhatia et al., 2019).

Therefore, this research conducts studies about the workload prediction of cloud data center resource i.e., CPU usage in general and GWA-T-12 Bitbrains distributed cloud data center as a case base for primary observation, analysis and implementation of the proposed Long Short-Term Memory (LSTM) deep learning model that how proactive workload prediction can improve efficient resource utilization and reduce energy consumption of a cloud datacenter. On the other side to evaluate how deep learning techniques such as CNN, LSTM and CNN-LSTM can be used to consolidate virtual machines to fewer physical machines through live migration based on accurate workload estimation.

1.2 Motivation of the Study

One of the most difficult aspects of cloud computing is precisely predicting future resource usage in order to meet future needs. Because cloud resource usage is constantly changing and exhibit a complex dynamic behavior (Barthwal et al., 2019). CPU and memory utilization is one of the most important metrics for measuring the performance of virtual machines and is a popular metric for researchers to determine when predicting future virtual performance. In virtualized environments, CPU and memory is usually the resource with the highest level of demand and is therefore a major cause of resource shortages on cloud VM (Chen & Wang, 2019).

This motivates the researcher to conduct this research study to generate accurate forecasts for time series problems, making it a good option for predicting CPU and memory utilization more precisely than previous methods.

1.3 Statement of the Problems

In cloud computing, virtual machine (VM) workload impacts resource provisioning. Cloud services are hosted in data centers with a huge number of servers. Cloud users request resources (for example, CPU and memory capacity) from service providers, who allocate the resources by building virtual infrastructures on top of actual hardware called Virtual Machines (VMs), which simulate the behavior of a computing system with a specific set of resource characteristics.

Cloud computing resource provisioning involves the adaptive and precise prediction of cloud workloads, to manage cloud resources efficiently, reduce resource costs and accurate prediction of virtual machine resource needs is critical. When resource use exceeds a specific threshold, most auto scaling systems add/remove Virtual Machines (VMs). The necessity to precisely anticipate the application resource requirements for time-varying workloads makes realizing the full potential of the cloud utilizing proactive auto-scaling difficult. As a result, precise forecasting of virtual machine resource requirements is crucial for effectively managing cloud resources and lowering resource costs. The majority of existing resource prediction methods are based on statistical analysis with a shallow structure or on a neural network with a single hidden layer. As a result of the workload data's randomness and nonlinearity, the prediction model has a difficult time finding intrinsic features (Qiu et al., 2016).

Deep learning has been proven to be suitable for massive data processing, with numerous successful applications in domains such as computer vision and pattern recognition. They've been demonstrated to do better on a variety of tasks. Deep learning appears as a candidate approach based on this, promising more accurate prediction (Ouhame et al., 2021). Therefore, A deep learning PSO-Long Short-Term Memory (LSTM) resource prediction model is proposed to address this problem.

1.4 Research Questions

RQ1. In what ways does workload prediction improve cloud data center resources usage?

RQ2. How can PSO-LSTM neural network be used to increase the accuracy of cloud client resource usage predictions?

RQ3. How to implement the designed architecture so that the cloud resource utilization prediction would be better in terms of accuracy?

1.5 The Objective of the Study

1.5.1 General Objective

The main objective of this study is to accurately predict the VM workload utilization for different time intervals benefiting the cloud management for proper utilization of the available resources.

1.5.2 Specific Objectives

In order to achieve the general objective, following step by step specific objectives were achieved to complete the research

- Deep recurrent neural network architecture for the prediction of cloud client resource utilization is designed.
- The architecture of a deep recurrent neural network is implemented.
- Deep recurrent neural network algorithm with collected real-world cloud dataset is trained.
- The trained model with the test set to test the best parameter.
- The accuracy of the proposed architecture is evaluated.

1.6 Significance of the Study

It has been seen that cloud data centers in developed countries are criticized for extensive emission of greenhouse gases. However, with the growing demand of cloud services developing countries

are also required to start thinking about green or energy efficient cloud computing data center so as to avoid criticism and ensure environmentally sustainable systems.

This research helps cloud system providers enhance overall cloud resource provisioning by anticipating future resource demand. When the service provider allocates either too many or too little resources, inefficiency in the allocation of an optimal number of services leads in over-expenditure. Service providers can better manage their capacity and dynamically scale the number of active machines to achieve an SLA for an energy-efficient resource provisioning system by forecasting client needs.

1.7 Scope of the Study

The scope of the proposed work within the given time and resources includes-

- Developing a prediction model for the cloud client resource utilization using LSTM architecture for better accuracy.
- Train and test the model using Bitbrain dataset collected from a distributed cloud data center.
- The developed model considers prediction of the CPU usage of cloud clients.

1.8 Limitation of the study

This thesis does not cover the following due to time and resource limitations.

- The lack of high-performance hardware capable of deep learning was a major stumbling block to the achievement of this thesis. Deep learning is a computationally difficult task that requires high-performance technology such as the Graphics Processing Unit (GPU).
- Paying for a real-time cloud service provider to receive real-time (live) data from the host is too expensive. It assists in consistently predicting the best results of the proposed model.
- Furthermore, budget, deadline, and resource constraints limited greater practicability.

1.9 Contribution of the Study

The experiment finding shows, the proposed approach can increase workload anticipation performance as compared to existing workload forecasting methods.

The contributions of this research study are:

- Propose a Particle Swarm Optimization (PSO) based LSTM multivariate model for CPU usage forecasting for cloud data center.
- Comparison of the proposed model with other different models such as CNN, CNN-LSTM and LSTM
- Conduct experiments and evaluate the proposed model with the Bitbrain real world workload traces.

1.10 Thesis Organization

The research thesis is framed in seven chapters. The following is a brief description of these chapters:

Introduction (Chapter One) Covers general background of the domain, target problem statement, research questions to be answered at the end, major goal and objectives of the research with scope, significance and limitations.

Literature Review (Chapter Two) Covers review of theoretical literature and related research works: The theoretical basis of the deep learning models and cloud workload prediction methods are presented here along with the critical review remarks of the intersection between the two concepts. Solutions and techniques researched or developed by salient researchers prior to this research are reviewed here from books, journal papers, conference papers and research documents like thesis and dissertations.

The methodology of the study (Chapter Three) Covers research design, methods and methodology used in this research study. It covers detailed basis of selection of methods, methodology with reference to focused problem parameters and suitability assessments.

Proposed Solution (Chapter Four) Covers data collection and analysis related to target problem under study. In this section the research methodology for data collection, sample size, sampling frame required to address the research problem and answer the primary research questions are covered. This section also presents detailed discussion about the proposed LSTM model and the optimization technique used to optimize the model's parameter i.e., layer weights.

Implementation (Chapter Five) Covers Modeling and Experimental Analysis. In this section the Proposed PSO based LSTM model has been designed, developed and explained. Here the experiments have been done using Spyder development environment for scientific programming via anaconda distribution and the results are discussed in detail.

The result, Evaluation, and Discussion (Chapter Six) This chapter describes the experiments carried out to evaluate four different prediction models, namely CNN, LSTM, CNN-LSTM, and the proposed PSO based Long Short-Term Memory (LSTM) using different workload data from a real distributed cloud datacenter.

Conclusion and Future Work (Chapter Seven) Covers Conclusion and Future work. In this section; the new knowledge outcomes about virtual machine's cloud workload prediction in general and Bitbrain dataset as a case have been summarized with brief explanation. In addition to it few recommendations along with future research ideas are covered.

CHAPTER TWO

LITERATURE REVIEW AND RELATED WORKS

2.1 Introduction

Cloud computing is an Internet-based computing architecture in which hosted services are delivered over the Internet on a pay-as-you-go basis (Alwafi Ridho Subarkah, 2018). CC is a utility computing model that makes it simple to access a pool of computing resources such physical equipment, servers, applications, compute, storage, networks, and other services (Anupama et al., 2021). Cloud technology offers a variety of benefits, including flexibility, disaster recovery, mobility, storage, on-demand resources, and more. Despite these benefits, the technology has some flaws, such as efficient resource scalability, power consumption, virtual machine location, security, privacy, and resource use(J. Kumar et al., 2020). Because different workloads may have distinct resource consumption footprints and can be further separated by their time fluctuations, Cloud computing data centers use virtualization technology to schedule workloads on lesser numbers of servers that could have been better employed. However, appropriately distributing and planning resources to maximize resource efficiency and ensure QoS remains a difficulty(Chen & Wang, 2019).

Many cloud platforms and virtualized data centers provide service elasticity as a standard feature. This is described as the capacity to adjust the system to changing workloads by autonomously providing and deprovisioning resources so that the available resources at any one time closely match the present service demand. There are two benefits of adopting service elasticity techniques. On the one hand, it provides consumers with Quality of Service (QoS), which can be stated in a variety of ways depending on the service type, including response time, throughput (e.g., requests/s), service availability, and so on.(Moreno-vozmediano et al., 2019).

Resource management is the process of assigning resources to a collection of apps in a way that fulfills the applications', cloud providers', and cloud users' performance goals. In general, there are three participants in a cloud environment: a Cloud Provider, a Cloud User, and an End User(Banerjee et al., 2021). They each have different goals and duties to fulfill. The cloud provider's goals are to focus on resource efficiency and effectiveness while adhering to SLAs with

cloud users. The performance, availability, and cost-effective scaling of applications are more important to cloud users. The cloud provider manages the resources by assigning them in a way that meets SLAs from the standpoint of cloud resource management. The resources are used by the cloud user to host applications. Workloads are generated by end users and processed by cloud resources(Alwafi Ridho Subarkah, 2018).

The cloud resource use forecasting approach anticipates the prospective resources required in such a way that the resource provider has enough of time to assign resources before a bottleneck emerges. Even if the workload suddenly increases, the resource delivery manager auto-scales up the resource system and designs virtual machines based on the projected prospective resource demand, while also allowing demand reduction by releasing reserved resources(K. D. Kumar & Umamaheswari, 2018).

Proactive prediction approaches have been used to address the difficulty of predicting resources, which is the estimation of the correct amount of resources required. The proactive approach forecasts future resource requirements so that the resource provisioning manager has enough time to provide resources before a bottleneck occurs. Even if the workload suddenly increases, the resource provisioning manager automatically scales up the resources structure and prepares virtual machines based on the predicted future demand for resources; at the same time, if the demand decreases, the resource provisioning manager will release allocated resources. As a result of the prediction methodologies, the Service Level Agreement (SLA) is met, power waste is prevented, and demand provisioning of re- sources is met(K. D. Kumar & Umamaheswari, 2018).

2.1 Auto-scaling Techniques

Elastic applications adapt to changing workloads by dynamically provisioning and releasing computer resources. An auto-scaling system is required to offer elasticity by automatically adjusting resources to handle the workload of the applications. Without human interaction, the auto-scalers make dynamic scaling decisions(Alwafi Ridho Subarkah, 2018).

Simple threshold-based rules, reinforcement learning, queueing theory, control theory, and time-series analysis are all examples of auto-scaling procedures. Each of the categories usually contains a wide range of strategies that can be used in a reactive or proactive manner(Lorido-Botran et al., 2014). Reactive means that the system reacts to variations in demand that are observed. Reactive

auto-scaling may be unable to respond to quick changes in workload, particularly in the event of a sudden increase (a new virtual machine usually takes 5 to 10 minutes to be operational). Because new resources take time to purchase and configure, proactive auto-scaling is a more promising strategy because it reacts to expected future demand.

Auto-scalers can also be classified based on the underlying technique utilized to integrate the auto-scaler into the system(Lorido-Botran et al., 2014).

- **Threshold-based rules:** The threshold-based rules base their scaling decisions on some performance measurements (such as average input request, average response time, and average CPU utilization) and a pre-defined threshold. This technique is considered straight forward; however, it has a difficult time selecting the appropriate performance criteria.
- **Reinforcement learning:** This system learns the best available option in each condition based on previous experience (trial and error method). After executing each action, the auto-scaler receives data from the system that indicates the quality of the action executed; as a result, the auto-scaler has a strong tendency to execute the best actions. The main disadvantages of this technique are the lengthy learning periods and poor early results.
- **Queuing theory:** Modeling the system is used in queueing theory to determine the system's future resource requirements. It entails establishing connections between jobs entering and exiting the system. A basic queuing architecture can be used to create elastic applications, in which the load balancer is represented by a single queue that distributes requests across a number of VMs. To solve queuing models, simulation is frequently utilized. Average queue waiting time and average response time were employed in several of the performance metrics. The non-flexibility of the queuing theory model means that the parameters and metrics must be recalculated on a regular basis.
- **Control theory:** Control theory necessitates the creation of an application model. It entails creating a controller that modifies the required resources to meet the application's requirements. There are several types of control systems, including open-loop controllers, which compute input to the system using current state data without requiring feedback to determine whether the goal has been met. The feedback controller monitors the output of the system and corrects any deviations from the desired value. Finally, a feed-forward controller that anticipates and reacts to problems before they occur. The most common

control system is feedback controllers. They change the input variables to match the output variable's goal value. However, developing a reliable performance model that maps input to output variables remains a difficult task.

- **Time series analysis:** A specific performance metric is collected frequently at set intervals for prediction of future values in the time series analysis technique. Time series analysis can be used to anticipate future values by identifying repeated patterns in the input workload.

2.2 Time Series Analysis and Forecasting

A time series analysis is a method of studying a collection of data points over a period of time. Instead of capturing data points intermittently or arbitrarily, time series analyzers record data points at constant intervals over a predetermined length of time.

- **Moving Average (MA) methods:** The approach can be used to eliminate noise or generate predictions. The weighted average of the last history window of the succeeding values is calculated by the prediction formula. There are also several versions, such as the simple moving average, which gives equal weight to all data. When there is no trend, it is merely useful for forecasting. Each observation value is given a distinct weight in a weighted moving average. In which the most recent observations are given more weight. When it comes to predicting, moving averages methods are still unable to handle big patterns.
- **Exponential smoothing (ES):** The weighted average of past data is computed using ES, which is comparable to MA. It does, however, take into account the time series' previous history. It identifies weights that are falling exponentially over time. ES comes in a variety of flavors, including Single ES, Double ES, and Triple ES. In the case of time series with shifting trends, a single (i.e. simple) exponential smoothing is insufficient. When there is a linear trend, though, double ES is a solid option.
- **Auto-regression of order p, AR(p):** The formula for making predictions is based on a linear weighted sum of previous values. The basic goal is to find the best weights or auto-regression coefficients possible. The AR coefficients are calculated using a variety of methods. The most prevalent are those based on auto-correlation coefficient calculations and Yule-Walker equations(Alwafi Ridho Subarkah, 2018).

- **Auto-Regressive Moving Average (ARMA):** Auto-regression and moving average are combined in ARMA. It is appropriate for stationary time series, which do not have a trend or seasonality. For non-stationary time series, however, the Auto-regression Integrated Moving Average (ARIMA) model, which is an extension of the ARMA model, can be utilized.

2.2.1 Machine Learning Techniques

The newest proposed ideas are based on machine learning methods. Machine learning-based algorithms predict the application's behavior in multiple dimensions. They're employed for more than just predicting possible resource behavior(Le Duc et al., 2019). Machine learning approaches are also commonly utilized to anticipate VM workloads(Gao et al., 2020). Regression and neural networks are the most widely used machine learning algorithms. Regression is a frequently used technique for estimating the relationships between variables that is based on a statistical model. A neural network is a collection of nodes (neurons) that are connected by a series of layers that include an input layer, an output layer, and a hidden layer or levels in between The neural network outperforms the MA and simple ES approaches in terms of accuracy(Sridharan, 2021).

The other type of time series analysis focuses on discovering patterns in time series and then using these patterns to forecast future demand. It is normally divided into four categories: Trend, Seasonality, Cyclical, and Randomness.

In general, time series analysis is thought to be the most important step in proactive auto-scaling. The key disadvantage is that the accuracy of the prediction is dependent on the application demand.

2.2.2 Time Series Forecasting with Deep Learning

Deep learning is a subset of machine learning techniques that use many layers of nonlinear information processing to extract supervised and unsupervised features, as well as analyze and classify patterns(Niu et al., 2020).

Time series forecasting differs from conventional regression predictive modeling in that the sequence introduces an order reliance on the observation that must be retained throughout model training and prediction. The literature revealed a variety of deep learning architectures that could be employed for time series forecasting. Multilayer Perceptron and Long Short-Term Memory are the two most frequent(Rao et al., 2019).

2.2.2.1 Multilayer Perceptron

Artificial Neural Networks (ANN) or Multilayer Perceptron (MLP) is one of the most extensively used machine learning approaches for prediction. The ability of neural networks to learn the representation of training data and relate this representation to the output variable we need to predict is their strength. The hierarchical structure of the networks, which permits expressing features at multiple scales, gives ANN its predictive potential. Neurons are the fundamental building blocks of neural networks. They are simple computational models with weighted input that produce an output value using an activation function (Shahvaroughi Farahani & Razavi Hajiagha, 2021). The activation function is a straightforward mapping between the neuron's summed weighted input and output. Neurons are divided into layers in a Multilayer Perceptron, and these layers are grouped into networks called network topology.

To construct a model that can be used to generate predictions, neural networks must be trained on the dataset. Stochastic gradient descent is a popular artificial neural network training algorithm. Each row of data is exposed as input to the network and processed upwards activation function to create an output. Forward pass is the name of this phase. The second phase is the back propagation pass, in which the network's output is compared to the predicted value in the dataset and the error is calculated. They subsequently traveled back across the network, updating the weights based on how much mistake they contributed to(Saxena & Singh, 2021).

Because of their capacity to solve issues stochastically, MLPs are beneficial for a wide range of machine learning problems. They do, however, have certain limitations when it comes to challenges involving sequence prediction. A unique sort of neural network built for sequence problems will be introduced in the following section(Shahvaroughi Farahani & Razavi Hajiagha, 2021).

2.2.2.2 Long Short-Term Memory

Recurrent Neural Networks (RNNs) are a form of neural network that is specifically developed to solve sequence problems. The architecture of an RNN can be thought of as adding loops to a normal feedforward Multilayer Perceptron network. Another way to think about RNNs is as a network with memory that can store and retain information while also taking advantage of the ordered nature of observations within input sequences (Zaremba et al., 2014).

Back-propagation is a technique for training feedforward neural networks that break down due to loop connections in RNNs. As a result, a novel technique called Back-propagation Through Time (BPTT) is used to train RNNs, in which the network's structure is unrolled (copies of neurons are formed) into a full network.

Figure 2.1 depicts the unrolling of a typical RNN (left) (right). This section of NN (A) examines an input xt and returns a value ht. In RNN, a loop allows data to be transmitted from one step to the next.

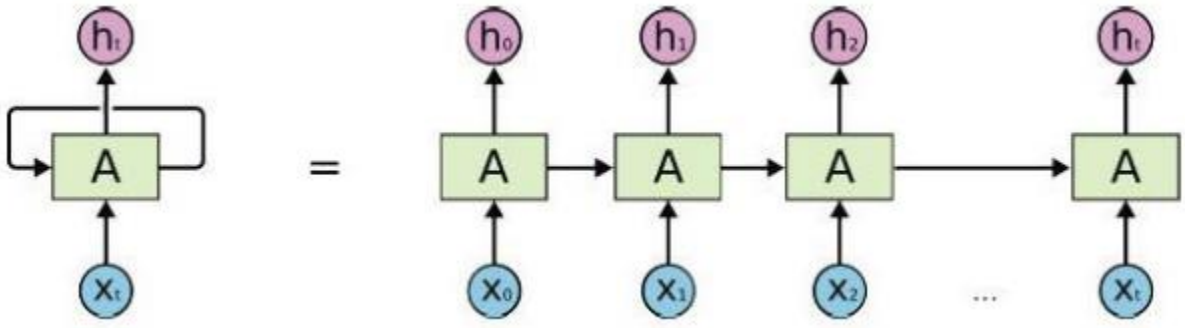

Figure 2.1 back-propagation technique (Merangin et al., 2018)

When using back-propagation in very deep learning and unrolled RNN, the gradients created to update the weights can become unstable. They can become very small, resulting in disappearing gradients, or very big, resulting in exploding gradients.

The Rectifier function, or unsupervised layer pre-training, is used to tackle this problem in a deep Multilayer Perceptron network. It is solved or decreased in RNN by employing a novel architecture known as Long Short-Term Memory.

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) that was created to solve the problem of vanishing gradients. Instead of neurons, the LSTM uses memory blocks that can be joined into layers. There are three basic gates in each block: a forget gate, an input gate, and an output gate(Dang-Quang & Yoo, 2021).

Those gates control how much information is discarded, updated, and outputted in order to manage the state and output of each block. Many time series jobs are intractable by MLP using the fixed size window approach, however LSTM can solve them(Gers et al., 2001). It also has the ability to perform challenging "long-time-lag" jobs that prior RRN algorithms have never been able to solve.

Figure 2.2 compares the construction of an LSTM cell with four NN layers (button) to a typical RNN with a single layer (top).

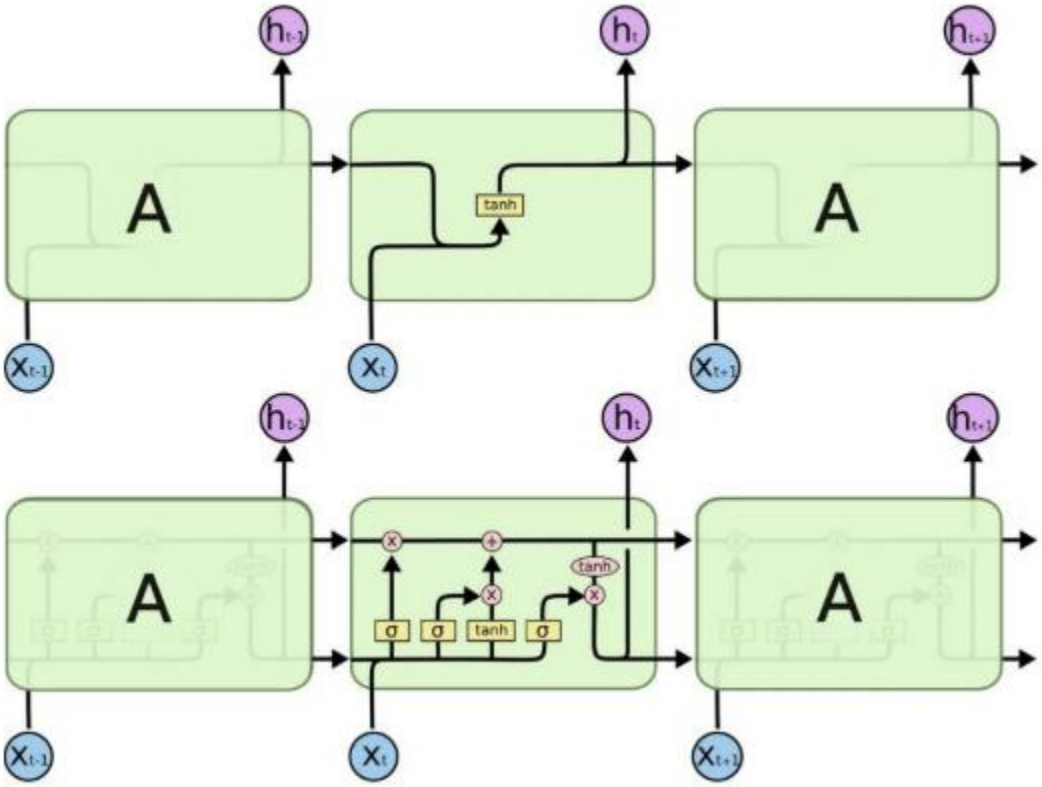

Figure 2.2 compares the construction of an LSTM cell with four NN layers (Merangin et al., 2018)

2.3 Review of Related Research Works

Almost all cloud computing providers such as Google, Amazon, Alibaba and Microsoft Azure are recognizing benefits and efficiency of using deep learning techniques for predicting cloud resources so that they can reduce the power consumption of their data centers and make the cloud data center environmentally sustainable.

The study (Hsu et al., 2014) describes cloud computing as a grouping of Physical Machines (PMs) that can be utilized to run multiple Virtual Machines (VM). It enables the widespread adoption of virtual machine consolidation as a critical tool for maximizing resource usage and lowering energy consumption. However, incorrect VM consolidation might lead to frequent live migration, resulting in substantial quality of service degradation and resource overhead.

The most difficult difficulty in addressing insufficient resource utilization and energy consumption in cloud computing is accurate forecasting of future VM workloads. Accurate VM workload prediction enables reasonable allocation of virtual machine resources to anticipated workloads. The majority of today's cloud apps are made up of a collection of interconnected service components that execute various tasks (Kalyampudi et al., 2021). These service components are spread over a number of heterogeneous, non-independent virtual machines (Zhang et al., 2018). As a result, rather than presenting a single VM workload, we describe a workload prediction approach based on multiple VM workload datasets.

Many methods for predicting VM workloads have been proposed. Mahendra Pratap Yadav et al. (Yadav et al., 2021) presents to estimate future server load, a predictive study of time series forecasting using deep learning method Long Sort-Term Memory was performed. Three measures have been used to assess the LSTM's prediction accuracy: RMSE, MSE, and MAE.

Yazdanian Peyman and Sharifian Saeed (Yazdanian & Sharifian, 2021) propose a hybrid E2LG algorithm for predicting highly nonlinear cloud workloads, which decomposes the cloud workload time-series into its constituent components in different frequency bands using an empirical mode decomposition method, reducing the complexity and nonlinearity of the prediction model in each frequency band.

Dang-Quang Nhat Minh and Yoo Myungsik (Dang-Quang & Yoo, 2021) developed a Kubernetes-based system architecture with a proactive custom autoscaler that uses a deep neural network model called Bidirectional Long Short-term Memory (Bi-LSTM) to dynamically handle workload during run time.

Using Machine learning approaches, Banerjee Sounak, Roy Sarbani, and Sunirmal Khatua (Banerjee et al., 2021)suggested a multi-step-ahead workload prediction strategy. The framework surpasses previous state-of-the-art approaches in terms of long-term workload prediction and greatly increases resource usage while allowing for significant energy savings.

Karim Md Ebtidaul et al. (Karim et al., 2021) offer the BHyPreC, a hybrid Recurrent Neural Network (RNN)-based prediction model. On top of the stacked Long Short-Term Memory (LSTM) and Gated Recurrent Unit is Bidirectional Long Short-Term Memory (Bi-LSTM) (GRU). It's used to anticipate how much CPU a cloud VM will need in the future. S. R. Shishira and A.

Kandasamy (Shishira & Kandasamy, 2021)established a novel conceptual paradigm for cloud computing platforms that allows for better workload prediction. By selecting the CPU label, Memory storage, and Disk, the suggested Feature extraction model enhances prediction accuracy.

Anupama K, Shivakumar B R, and Nagaraja R provide a novel approach to forecasting seasonal and non-seasonal workloads in their paper (Anupama et al., 2021). They suggested a hybrid prediction model that combines statistical and machine learning techniques to achieve this. For non-seasonal workloads, the test recommends using Long Short-Term Memory Networks (LSTM) or the AutoRegressive Integrated Moving Average (ARIMA) model. The model calculates the actual resource needed for various time intervals such as daily, hourly, and minute usage.

For predicting multivariate workload, Soukaina Ouhame, Youssef Hadi, and Arif Ullah (Ouhame et al., 2021) suggested a convolutional neural network and long short-term memory model based on the central processing unit, memory, and network usage.

Langston Nashold and Rayan Krishnan (Nashold & Krishnan, 2020) use Long Short-Term Memory (LSTM) and Seasonal Auto Regressive Integrated Moving Average to estimate CPU use over both short and long time scales (SARIMA).

Md. Nahid Hasan Shuvo et al. (Hasan Shuvo et al., 2020) introduced LSRU, an unique hybrid-method for enhancing resource utilization forecasting prediction accuracy, which is primarily a combination of two models, namely Gated Recurrent Unit (GRU) and Long Short Term Memory (LSTM) (LSTM).

According to the findings of the experiments stated above, deep learning can increase the accuracy of workload prediction for virtual machines on a cloud system. The majority of studies have improved cloud resource prediction accuracy. However, there is still a need to increase cloud resource prediction accuracy.

Table 2.1 Summary of focused and related Research works with Critical Remarks

S/N	Title	Author	Objective	Gap	Algorithm
1	A Hybrid CNN-LSTM Model for Virtual Machine Workload Forecasting in Cloud Data Center	(Leka et al., 2022)	To create an accurate prediction, the authors proposed a prediction model which integrated a convolutional neural network (CNN) architecture and a long-short-term memory (LSTM) neural network	The researcher used CNN and LSTM neural network. However, this research study has used optimization technique i.e., PSO	CNN and LSTM
2	Resource Utilization Prediction in Cloud Computing using Hybrid Model	(Anupama et al., 2021)	to accurately predict the CPU and memory utilization for different time intervals.	The researcher cannot develop task level resource usage prediction.	SARIMA and LSTM
3	Efficient resource utilization using multi-step-ahead workload prediction technique in cloud	Sounak Banerjee (2021)	timely provide the resources according to the actual demands of the application by accurately predicting the VM's future resource usage	Adopting a stochastic resource provisioning framework not fully addressed.	LSTM

4	An efficient forecasting approach for resource utilization in cloud data center using CNN-LSTM model	(Ouhame et al., 2021)	The paper proposes a convolutional neural network and long short-term memory model for predicting multivariate workload which are the central processing unit, memory, and network usage	The researcher used CNN and LSTM neural network. However, this research study has used optimization technique i.e., PSO	CNN and LSTM
5	Multiple Regression Particle Swarm Optimization for Host Overload and Under-Load Detection	(Saeed & Alhammadi, 2020)	The paper proposes an accurate prediction model called Multiple Regression particle swarm optimization (MR-PSO) to detect resource utilization	The paper uses PSO for multiple regression mode. However, the paper not used deep learning techniques.	Regression particle swarm optimization (MR-PSO)
6	Improving Resource Utilization in Data Centers using an LSTM-based Prediction Model	(Thonglek et al., 2019)	The paper aims to improve resource utilization in data centers by predicting the required resource for each application	The research study designed and implemented a neural network model based on Long Short-Term Memory (LSTM) to predict more efficient resource allocation for a job based on historical data	LSTM
7	Virtual Machines Placement using Predicted Utilization of Physical Machine in Cloud Datacenter	Varun Barthwal (2019)	to minimize consumption of power and enhance the service level agreement	The researcher used CPU usage as a prediction. However, it only depends on a single VM	Local Regression

8	Developing Deep RNN-Based Client Resource Utilization Prediction Model for an Improved Cloud Resource Provisioning	(Fulfillment & Engineering, 2019)	To predict the cloud resource utilization for better provisioning of cloud resources using a Deep RNN-Based architecture to improve the accuracy in the prediction of the workload.	This paper doesn't use any optimization technique.	LSTM

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Overview

The methodology, tools, and procedures used to analyze the research challenges are outlined in this chapter. It gives a quick overview of technique that can be applied to the proposed architecture's design, implementation, and evaluation. It went into sample selection, data collecting, and analytic methodologies in greater depth. Most crucially, this section includes the academic explanation for the method selected in order to achieve the research's main goal.

3.2 Data Collection Method

The dataset used in this study includes long-term and large-scale workload traces from a distributed cloud datacenter (Bitbrains). A total of 1,750 virtual machines' performance characteristics are included in the collection. For both requested and consumed resources, the traces include information about CPU, memory, disk I/O, and network I/O. Business-critical applications are hosted on the virtual machines. The Grid Workloads Archive has made the dataset publicly available. The dataset is in a row-based format and is sampled every 5 minutes. Each row represents a performance metric observation(Anupama et al., 2021).

The dataset is very huge in terms of size, with workload data for 1750 virtual machines (VMs), the majority of which include thousands of observations. We need to make a smaller sample to work with in order to evaluate in a reasonable amount of time. In reality, the dataset should be small enough that we can create and test deep learning models in an acceptable amount of time and calculations. Models that perform well can be scaled up and trained on any dataset afterwards. Randomly 6 samples of the dataset representing three different workload patterns were chosen for the examination.

In cloud computing environment there are three different workload patterns (St-Onge et al., 2020):

- Periodic workload which represents workloads with cyclical (seasonal) change.
- Growing workload which represents workload with growing trend.
- Unpredicted workload which represents fluctuating workload.

Samples from the dataset were chosen to illustrate these three workload patterns in this study.

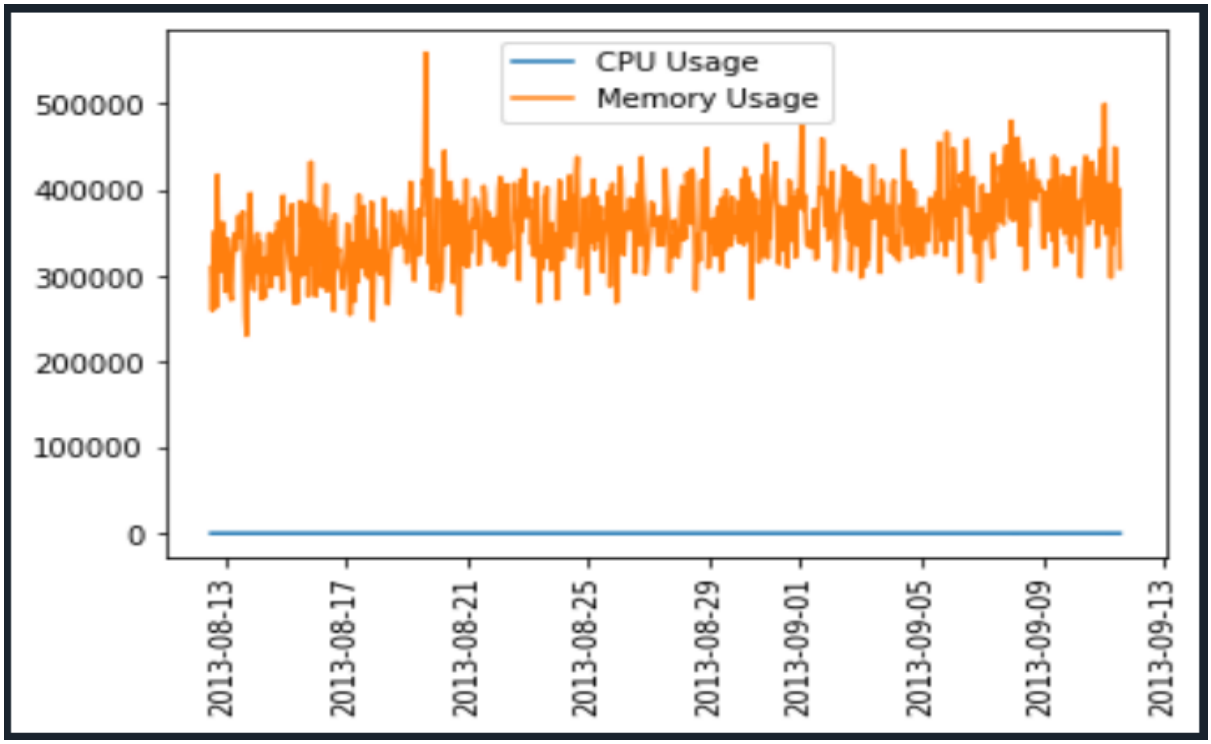

Figure 3.3 Sample of Dataset

3.3 Data Processing and Filtering

3.3.1 Data Preprocessing

Preparing the data in such a way that the structure of the problem is exposed to the deep learning algorithm that we will employ is a smart practice. We discovered that the dataset we have is clean and free of missing values after further investigation. We'll go over the preprocessing steps we took to get the data ready for the prediction models in this part.

It goes through several stages of pre-processing before being fed into the proposed model. First, the minimum-maximum scaler (MinMaxScaler) class scales all datasets' normal values to a range of 0.1 to 0.9.

$$x_{1} = LWL + \frac{X_{i-} \; X_{min}}{X_{max} - X_{min}}$$

Equation 3.1 (Banerjee et al., 2021)

3.3.1.2 Features Selection

The central processing unit (CPU) is a highly essential cloud resource required by customers; the CPU consumption of virtual machines is anticipated in this research study. It is possible that superfluous features will be removed during this phase, and new features will be developed. Unrelated features in the data can reduce the performance of many deep learning

models in general. Feature selection aids in the reduction of overfitting, increases model performance, and reduces training time. The features that have the greatest impact on the output variable (CPU utilization) should be chosen. The CPU consumption history will be utilized to forecast future values for the univariate deep learning model. It is critical to identify the desired features for a multivariate deep learning model. According to the research studies (Anupama et al., 2021) (Merangin et al., 2018) (Leka et al., 2022) the proposed model used the common features from the randomly selected samples of the three categories of workload i.e. periodic, growing and unpredictable dataset. The features that contribute most to the output attribute i.e. CPU are used as an input to the proposed model. So memory consumption and CPU usage history are used as an input to the proposed model.

3.4. Parameter Setting

Some hyperparameters must be set up and modified properly while developing the LSTM model in order to generate an accurate forecast when back testing the proposed model. The proposed ideal hyperparameters will improve accuracy while reducing the risk of data overfitting. The best value for a hyperparameter is determined by back testing the LSTM model with test data. The MSE between the model's predicted adjusted cloud resource consumption and the actual cloud resource utilization for the timestamp will be determined.

3.5 Evaluation Method

To explain the output of the model that we will be creating, the results will be evaluated on a test dataset. We plan to use metrics like RMSE, MAE, and MSE to assess the efficiency of our model.

Mean Squared Error: The sum of the squared forecast error values is used to measure the mean squared error, or MSE. In practice, the score penalizes models that make a significant number of incorrect predictions(J. Kumar et al., 2020).

$$MSE = \frac{1}{n}\sum_{n=1}^{n}(y_i - y_i)\,2$$

Root Mean Squared Error: The above-mentioned mean squared error is in squared units of the predictions. By taking the square root of the mean squared error score, it can be converted back to the original units of the predictions. The root mean squared error, or RMSE, is a measure of this(Niu et al., 2020).

$$RMSE = \sqrt{\frac{1}{n}\sum_{n=1}^{n}(y_i - y_i)\,2}$$

3.6 Model Training

3.6.1 Define and Compile the Model

Predicting VMs future resource i.e., Central Processing Unit (CPU) usage can be formally stated as a time series data.

First, the time series data of the Vm's CPU consumption is organized according to the proposed LSTM model's input format. Then, using the input shape of the input layer, the LSTM layer, the activation layer, and the dense layer, an LSTM model was generated by specifying different parameters and values to support the input format of the proposed framework Keras. The model was compiled after the model and network were created. The loss function, metrics, and optimizers are defined after the model has been compiled. The loss function is defined and bound in this stage by the model compilation. The evaluation metrics are a critical and mandatory phase in the model training process.

3.6.2 Train the Model

The network will fit the training and validation data after it is formed. The model measures how well the dataset fits and how well the suggested model fits the data in order to create a solid future prediction in this step. By continuously monitoring the changes in the loss and iteration over a specific number of epochs over the whole dataset, the model will be able to make dynamic modifications to the learning rate and make an early stop. The returned history item keeps track of the loss and metric data during training. In this research study we used the commonly used particle swarm optimization technique to optimize the proposed model's weight.

3.7 Deep Learning Models and Optimization Technique

In this research study we have reviewed latest literatures and used the most widely used deep learning techniques for virtual machine cloud workload prediction such as Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) and CNN-LSTM neural networks and optimization technique i.e., Particle Swarm Optimization (PSO) which is discussed in chapter three. After repeated experiments and review of literatures we have developed the PSO

based LSTM model for forecasting Virtual Machine (VM) workload prediction over cloud data center.

3.7.1 Recurrent Neural Network Models

A recurrent neural network is a feedforward neural network with built-in memory. For each data input, RNN performs the same function, but the current input's output is reliant on the preceding output. After then, the output will be replicated and sent back to the recurrent network. It evaluates the current input as well as the output it has learned from prior inputs whenever it makes a decision(*Understanding RNN and LSTM*, n.d.).

RNNs are neural networks that act on sequential data such as text, audio, video, and so on. It learns through sequence data and generates the current output using the previous step's output of network information as an input(Amiri & Mohammad-Khanli, 2017).

The Gated Recurrent Unit (GRU) and the Long Short-Term Memory (LSTM) cell are two RNN variants that are commonly used for sequence modeling tasks. With supervised learning problems(Hewamalage et al., 2021)(Ouhame et al., 2021).

- **Long short-term memory (LSTM)**

LSTM outperforms other RNN variations on tasks with substantial time lags. The vanishing gradient problem of RNN has been solved. Many sequential activities rely on the temporal distance between occurrences to communicate information, such as motor control and rhythm detection(Anupama et al., 2021).

Depending on the challenge, the LSTM neural network reads an input sequence one step at a time or more. The LSTM has an internal memory that allows it to accumulate internal state as it scans through a succession of inputs. At the end of the series, each node in a layer of secret LSTM units will emit a single value. The LSTM learnt or extracted from the input series is summarized in this vector of values. A completely linked layer will examine this before making a final forecast. LSTM can effectively identify, process, and forecast time series with unknown time lags. Back- propagation is used to train the model(P. Kumar, 2018).

- **ConvLSTM**

ConvLSTM is a spatiotemporal prediction variation of RNN that uses convolutional structures in both the input-to-state and state-to-state transitions. The ConvLSTM predicts the future state

of a cell in the grid using the inputs and past states of its nearby neighbors. A simple approach to achieve this is to use a convolution operator in the state-to-state and input-to-state transitions.

- **Convolutional Neural Networks**

CNNs, or Convolutional Neural Networks, are a type of neural network designed to efficiently process visual input. They've proven to be effective on difficult computer vision issues, reaching state-of-the-art results on tasks such as picture classification while also functioning as a component in hybrid models for wholly new problems such as object localization, image captioning, and more. Although CNN is most commonly associated with image datasets, it can also be applied to time series data (and may be more practical than RNNs)(Lee et al., 2021)(Masdari & Khoshnevis, 2019).

3.8 Development Tools

This study employs a variety of development techniques to build and implement the proposed system. The development tools are summarized and justified in this section.

Design Tools: Edraw-Max is used to create the proposed device. It's a quick and easy way to make professional-looking flowcharts, network diagrams, and UML diagrams, among other things(*Introduction to EdrawMax*, n.d.).

Hardware Tools: Hard disks is used as storage for huge datasets, GPUs will be used to expedite computation and training, and RAM used cooperatively with GPU to accelerate the training process.

Software Tools Software: Software tools, programming tools, and the libraries that are used:

- **TensorFlow:** It's a collection of open-source software for numerical computation and high-performance computing(*Deep Learning Framework Power Scores 2018 | by Jeff Hale | Towards Data Science*, n.d.).
- **Keras:** is a user-friendly, modular, and extendable Python-based framework(*Why Choose Keras?*, n.d.).
- **Anaconda:** Anaconda is a software that installs the Python programming language, as well as all of its modules, based on the Python version.
- **Jupyter notebook:** An interactive web-based tool for configuring Python APIs, loading Python code, and developing Python code.

- **Python:** It is used to conduct and demonstrate our testing, and it will include some of the drivers needed to configure and package any system that will be installed on the computer.
- **Microsoft office packages:** It's a process for creating diagrams and flowcharts for the solution we'll propose, and Visio will be utilized to do so.
- **Mendeley Desktop:** It's a handy reference manager that also functions as a scholarly social network for finding related works.

CHAPTER FOUR

PROPOSED PARTICLE SWARM-BASED LONG-SHORT TERM MEMORY MODEL

4.1 Overview

In this research study widely used cloud workload traces i.e., Bitbrain dataset, data pre-processing, LSTM models and optimization technique i.e., particle swarm optimization were used. Then the proposed model and its configuration parameters for the experiment are described and discussed in this chapter. The block diagram of the proposed PSO based LSTM model is shown in the figure below.

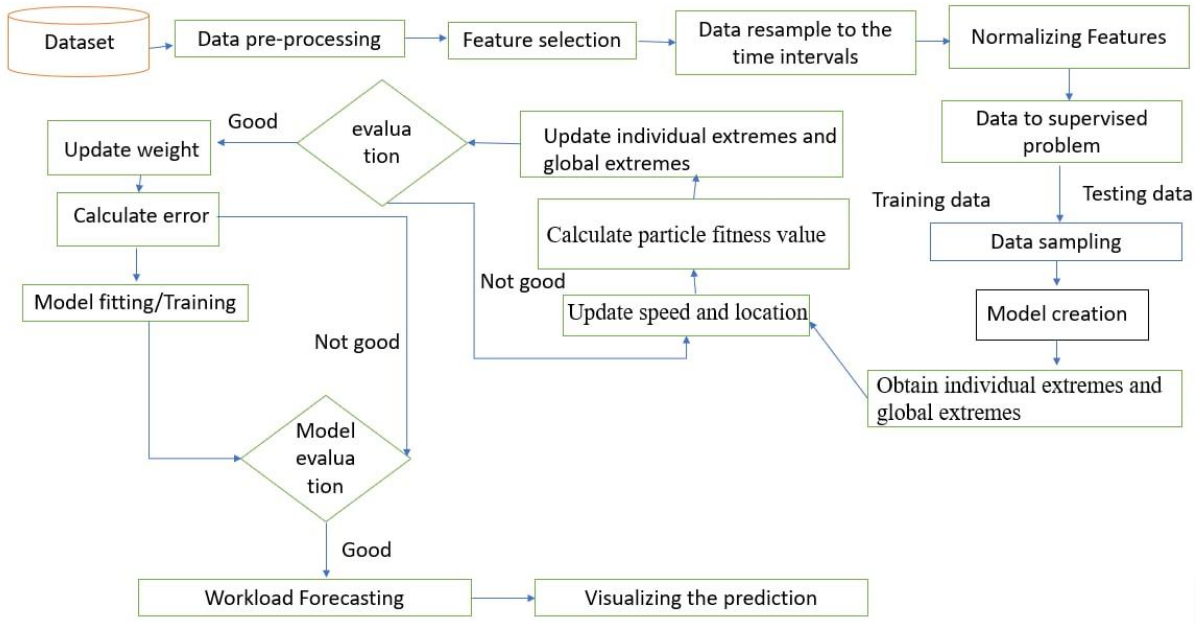

Figure 4.4 Block diagram of the proposed PSO based LSTM model

4.2 Dataset Collection

This research looks at the forecast of CPU consumption for six randomly chosen virtual machines from each of the three categories: periodic, increasing, and unexpected. The suggested model is evaluated in the tests using a CPU and memory consumption dataset from Bitbrain, a real-world large-scale and long-term workload trace from a distributed cloud datacenter. Virtual machines in the datacenter host mission-critical applications. There are two sets of data in the Bitbrains dataset. The first sub-dataset contains data for 1250 virtual machines in a fast storage area network (SAN), which is used in this research, and the second

group has data for 500 virtual machines from faster and slower Network Attached Storage (NAS) devices.

A total of 1750 virtual machines have been monitored and the CPU, memory, disk I/O, and network I/O utilization have been collected every 5 minutes. In a 70:30 ratio, the data is divided into training and testing data. The deep learning model's input vector is the central processing and memory utilization data history window for the entire VMs. Two virtual machines per category are chosen as the ones that need to be forecasted. The CPU consumption of 2 virtual machines per category is forecasted at four periods in the future, namely 5-minutes, 15-minutes, 45-minutes, and 1-hour.

Figure 4.5 The screenshot of Bitbrain trace for growing workload

4.3 Preprocessing and Data Formatting

4.3.1 Normalization

The term "normalization" refers to the practice of rescaling observations to the same scale between 0 and 1. It's most beneficial for sparse datasets with properties of different scales. All CPU and memory consumption data was scaled/standardized into [0, 1] before to training and validation. It comes in handy because the dataset contains a lot of zeros. Normalization is critical for algorithms that weight inputs, such as neural networks. Because this study will use

recurrent neural networks with Long Short-Term Memory, all features have been normalized into the range [0, 1] to make them appropriate for multivariate models. The MinMaxScaler class from the scikit-learn library was used to normalize the data (Masdari & Khoshnevis, 2020).

Figure 4.6 Screenshot of the normalized Bitbrain data for growing workload

4.3.2 Data Preprocessing

The row dataset from the Bitbrains must be transformed into the format that can be accepted by the proposed LSTM model. This preprocessing has been done by starting by normalizing the time series data into 0 and 1 using MinMaxScaler. In data preprocessing, scaling is the important step to achieve the fast learning and convergence of the network (Mashhadi Moghaddam et al., 2020). In the second step as shown in Figure 4.1, the dataset was divided into supervised learning problem which divides the data into input and output components using the sliding windows approach. In the last step, the supervised form data set is split into training and testing. The previous n (lag point/window size) time steps are utilized as an input variable, while the next k (forecast horizon) time steps are used as an output variable in the sliding window method. For the LSTM network, the sliding window approach is used to enable supervised learning. The input shape in [samples, window size, and features] format has been obtained after converting the time series to the supervised learning problem, which is the needed input format for the LSTM network. Other time series values will be considered while

constructing the multivariate time series, and the size of features will be more than one. The features in our situation are cpu usage data and memory usage data. For each time series, all of these data pretreatment processes were used.

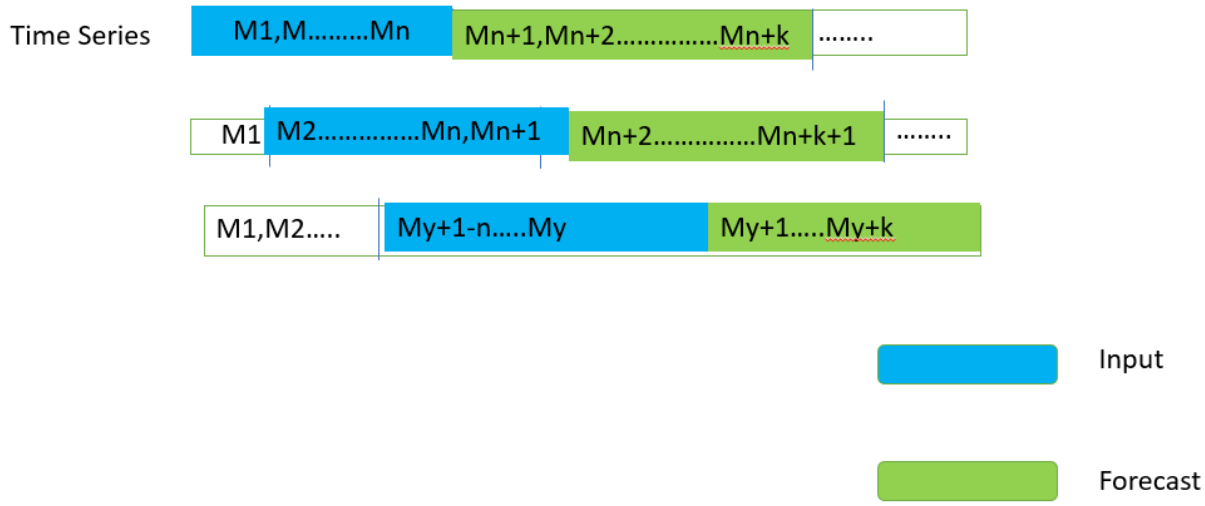

Figure 4.7 time series for LSTM model

4.4 LSTM Framework

The LSTM algorithm can learn dependencies over arbitrarily long time intervals. The LSTM unit or block replaces an ordinary neuron with a complicated design to avoid the problem of vanishing gradients. It's a recurrent neural network made up of three separate gates: forget, input, and output. Using those three gates, the neural network is capable of autonomously storing and erasing time state information(Mandal et al., 2020). As a result, LSTM excels at processing time series data and can mine the complex features of both long and short-term time series data. The most significant aspect of the LSTM is the memory unit, which stores current and prior time state information. The forget gate, input gate, and output gate of the LSTM remove, store, and output data, respectively. Figure 4.8 depicts an LSTM structural diagram.

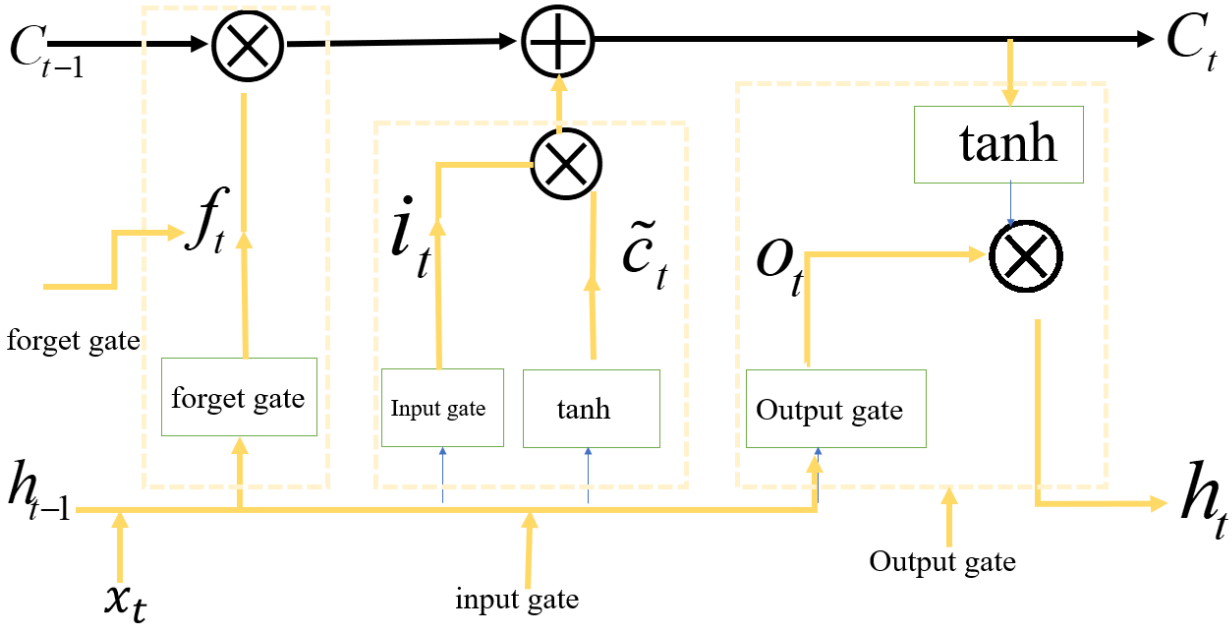

Figure 4.8 the basic cell of LSTM Network (Agga et al., 2021)

- The current input vector, represented by x_t, and the output from the previous time step (through the recurrent edges), denoted by h_{t-1}, are fed into the LSTM unit. The activation of tanh sums and moves the weighted inputs, resulting in z_t.
- The input gate reads x_t and h_{t-1}, calculates the weighted total, and then applies sigmoid activation. To give the input flowing into the memory cell, the result is multiplied by z_t.
- The forget gate is a mechanism that allows an LSTM to learn to reset memory contents when they become outdated and no longer useful. This can happen when the network is processing a new sequence, for example. The forget gate reads x_t and h_{t-1} and activates weighted inputs with sigmoid activation. The result, f_t, is multiplied by the cell state at the previous time step, i.e., s_{t-1}, allowing memory contents that are no longer required to be forgotten.
- The CEC, which has a repeated edge with unit weight, makes up the memory cell. The current cell state s_t is calculated by ignoring irrelevant information from previous time steps (if any) and accepting relevant information from the current input (if any).
- The output gate takes the weighted sum of x_t and h_{t-1} and applies sigmoid activation to control what information would flow out of the LSTM unit.

- The output of the LSTM unit, h_t, is computed bypassing the cell state st through a tanh and multiplying it with the output gate, ot. The functioning of the LSTM unit can be represented by the following set of equations.

The calculation process of individual's gate, input data, cell state, hidden state and the updating process of the LSTM memory unit is described in equation 1. x_t and h_t represents the input and hidden states of time t, respectively. The gates that control information flow, namely forgetting gate, input gate, and output gate are represented by $f_t, i_t, and\ o_t$, respectively. $\tilde{c}_t$ is the candidate information state for the input to be stored, and the amount of storage is then controlled by the input gate.

$$f_t = \sigma(W_{fx}.x_t + W_{fh}.h_{t-1} + b_f),$$

$$i_t = \sigma(w_{ix}.x_t + w_{ih}.h_{t-1} + b_i),$$

$$\tilde{\boldsymbol{c}}_{\boldsymbol{t}} = \boldsymbol{tanh\ tanh}\,(\boldsymbol{w}_{\tilde{c}\boldsymbol{h}}.\boldsymbol{x}_{\boldsymbol{t}} + \boldsymbol{w}_{\tilde{c}\boldsymbol{h}}.\boldsymbol{h}_{\boldsymbol{t-1}} + \boldsymbol{b}_{\tilde{\boldsymbol{c}}}) \qquad \textbf{Equation 4}.1 \quad \text{(Leka et al., 2022)}$$

$$c_t = f_t.c_{t-1} + i_t.\tilde{c}_t,$$

$$o_t = \sigma(w_{o_h}.h_{t-1} + w_{ox}.x_t + b_o),$$

$$h_t = o_t.tanh\ tanh\,(c_t)$$

$$\boldsymbol{sigmoid}\ (\boldsymbol{x}) = \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{e}^{-\boldsymbol{x}}} \qquad \text{Equation 4.2 (Leka et al., 2022)}$$

$$(\boldsymbol{x}) = \frac{\boldsymbol{e}^{\boldsymbol{x}}-\boldsymbol{e}^{-\boldsymbol{x}}}{\boldsymbol{e}^{\boldsymbol{x}}+\boldsymbol{e}^{-\boldsymbol{x}}} \qquad \text{Equation 4.3 (Leka et al., 2022)}$$

This study has been proposed several variants of LSTM architecture however, the prediction performance is similar. So, this research study proposed Particle Swarm Based LSTM model for better performance and accuracy as compared to backpropagation-based LSTM model. The proposed model is discussed in more detail next section.

4.5 Proposed PSO Optimization-Based LSTM Model

In this study, we took advantage of the benefits of deep learning architecture and created a particle swarm optimization (PSO) based LSTM model for cloud resource prediction, specifically CPU usage, for forecasting future workloads. The PSO technique is used in the model to optimize the model parameters, such as weight, in order to improve prediction accuracy. The proposed model is discussed in this chapter together with the optimization approach, PSO.

4.5.1 Particle swarm optimization

Particle swarm optimization (PSO) was suggested by Eberhart and Kennedy in 1995 as a population-based stochastic optimization technique based on the foraging behavior of a flock of birds (Yuan et al., 2021). The PSO algorithm first initializes the particle states to generate a set of stochastic solutions, and then the particles update their states in real time by tracking the individual local optimal solution (Pbest) and global optimal solution (Gbest) as they move around space.

In this paper, we use PSO algorithm that automatically search hyperparameters to improve the efficiency of cloud resource demand prediction. The PSO is an algorithmic representation of the movement of particles. It searches for a space using different particles and updates the result when an optimal position is found by a specific particle. We set the LSTM weight, and output weight of fully connected layer as hyper parameters. The fitness function is evaluated using Equation 4.4.

$$\boldsymbol{Fitness\ Function} = \mathbf{1} - \mathbf{1}/\boldsymbol{n} \sum_{i=1}^{n} (\boldsymbol{y_{i-}} \hat{\boldsymbol{y_l}})^{\mathbf{2}}$$ Equation 4.4 (Leka et al., 2022)

The PSO uses particles that swarm like birds to search for hyperparameter space. In general, each particle goes in the direction of hyperparameter optimization. The heuristic PSO method tracks all particle movement and attempts to identify the ideal position of all particles in hyperparameter space. We randomly insert particles in the hyperparameter space and give them a low beginning velocity. The speed and direction of the other particles influence the speed and direction of each particle. The PSO method is used to update the particle motion throughout generations. To find the best position, each particle adjusts its velocity, inertia, and position. We also set the objective function to optimize the use of cloud resources, such as CPU. Each particle is brought together across generations and time to optimize the LSTM model. The PSO algorithm keeps note of each particle's best previous location in order to influence surrounding particles to move to a good location. Equation (5) is used to update each particle.

$$v_{i(t+1)} = (W * v_t + c_{1} \times rand() * (p_i^{best} - p_i(t))) + (c_2 * rand() * (p_g^{best} - p_i(t)))$$ Equation 4.5

(Saeed & Alhammadi, 2020)

$$\boldsymbol{p_i(t+1 = p_i(t) + v_i(t+1)}$$ Equation 4.6(Saeed & Alhammadi, 2020)

Where *p* represents particle position, *V* represents particle velocity, gbest represents global best, best represents local best, *c1* and *c2* represent acceleration coefficients, and *W* represents inertial weight.

The PSO algorithm starts with random position p_0 and velocity v_0. Then for each particle it evaluates fitness and if the fitness of the particle greater than fitness of global best the global best will be updated. The other way if the fitness is of the particle is greater the fitness of personal best, the personal best will be updated. The velocity and the position of a particle is updated using equation 5 and 6 respectively.

The first part of Equation (5) is the sum of the previous generation particles' velocity and the inertia factor; the second part characterizes the individual search behavior of the particles, which motivates them to continuously search for the individual optimal solution; and the third part characterizes the information interaction between the particles, which jointly search for the global optimal value, where c1 and c2 are the individual optimal values.

4.5.2 PSO-Based LSTM Model

In this study, LSTM prediction model is developed and the key parameters i.e., weights in the LSTM are optimized using the PSO algorithm, and the model is finally applied to the three categories of VM workload prediction i.e., periodic, growing and unpredictable. The workload prediction and the optimization process are shown in Figure 4.6.

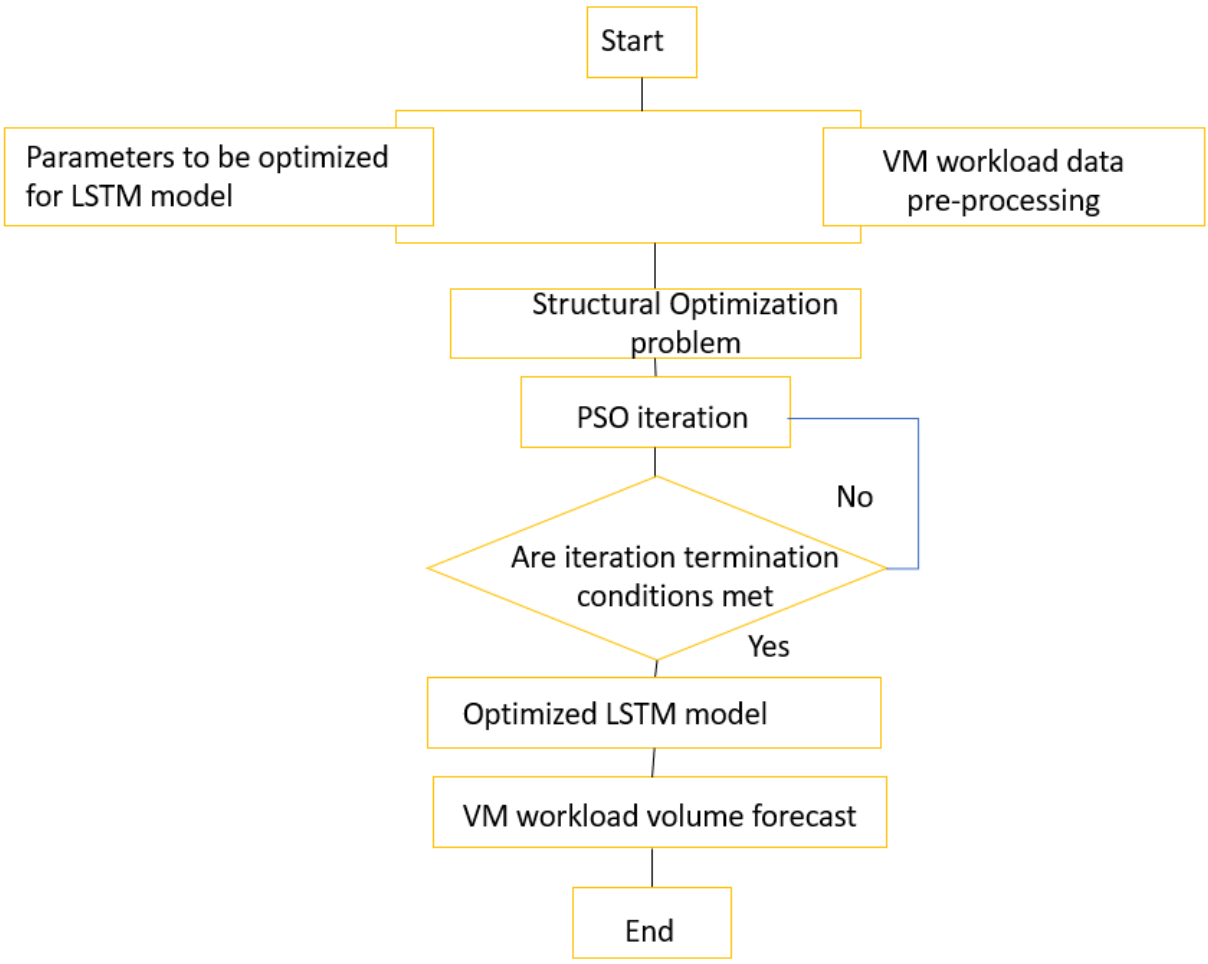

Figure 4.9 PSO based LSTM work load prediction process

The number of neurons, window size, time step, and other parameters such as learning rates are all manually adjusted in the LSTM model, and the weight parameters in each layer are improved via particle swarm optimization. The VM workload prediction error of the LSTM model is defined as the goal function, and the prediction error is minimized by iterating the method to discover a set of optimal model parameters. Once the model's weights have been optimized using PSO, the new PSO result is utilized as an initial weight for the LSTM model, and the outcome is compared to the LSTM model without PSO as a weight initialization.

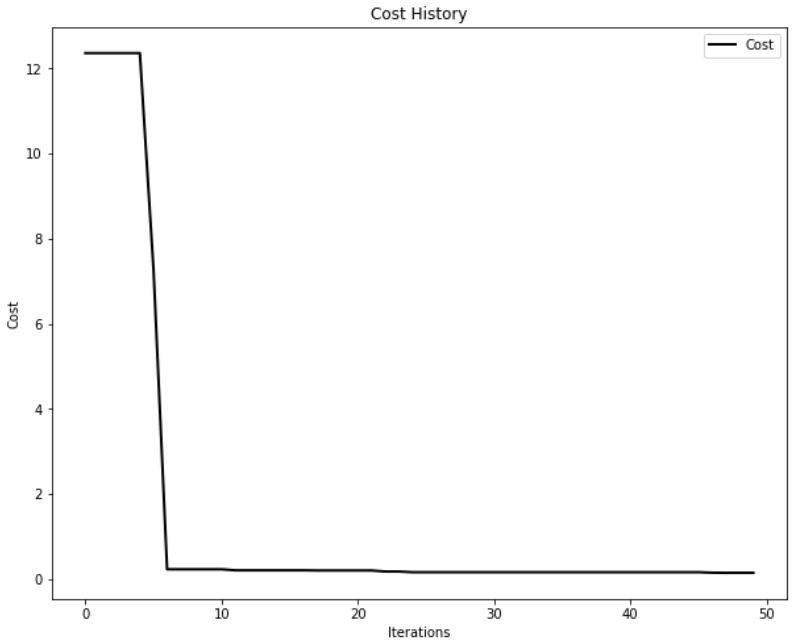

Figure 4.10 PSO based LSTM work load prediction process

The network structure of the proposed PSO-LSTM model is shown in Figure 4.8. After preprocessing of the dataset, the network takes 1x48 (48 lag points with two VMs CPU and memory usage) as input in order to process the data in the LSTM layer. The model has one LSTM and one dense layer. Then, the time series data which is inputted into the LSTM network learns the temporal features in a long-term sequence. Finally, the output time series data is passed via a fully connected layer for the future prediction. The last fully connected layer reduces the vector of length into the number of features, which is CPU utilization of two VMs under each workload category i.e., periodic, growing and unpredictable workload.

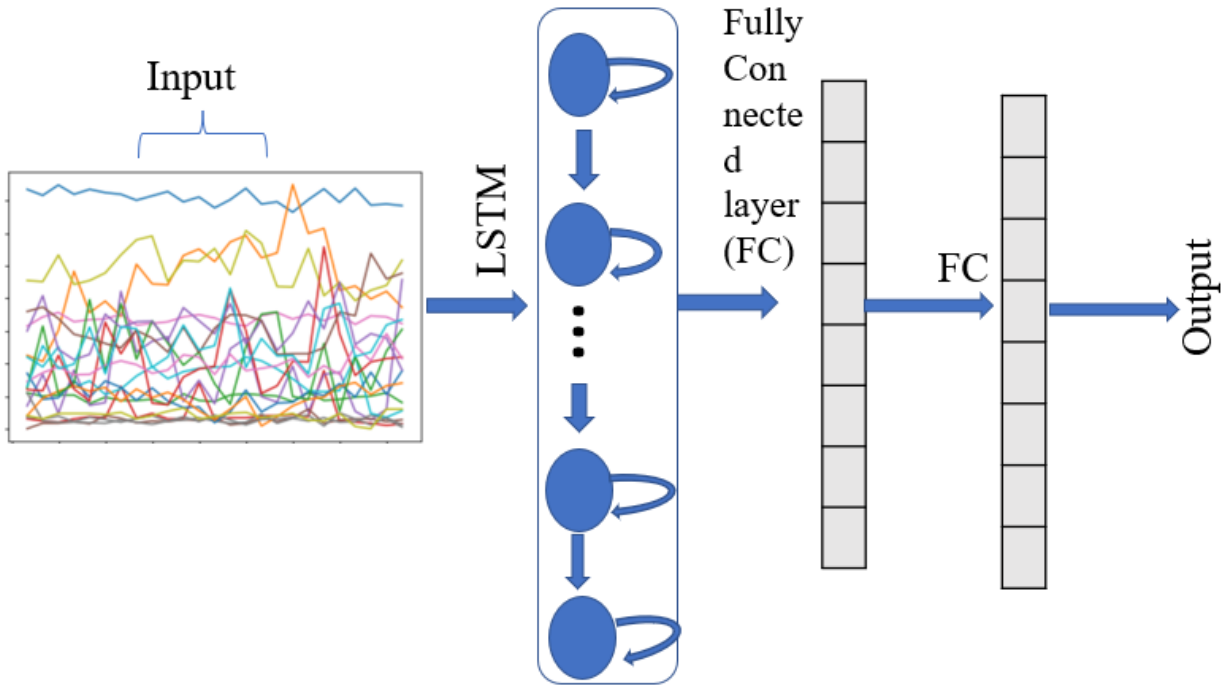

Figure 4.11 Proposed LSTM Model

The effectiveness of the suggested model is determined on the hyperparameter setup. The proposed model's configuration is provided in the following parts.

4.5.1 Data Input

The LSTM model learns a function that transfers a set of previous observations as input to a new observation as output. Each time-series timestamp is described in the dataset as a 1x2 vector, with 2 equaling the number of features (i.e., CPU and memory) that would be employed in the suggested forecast. The matrix has built and split all timestamp data into fixed-size windows, along with all timestamps and their attributes that are used in the forecast. Then reshape all of the vectors in the matrix such that the NumPy array is now a 3D vector with the form [sample, timesteps, features], where sample is the number of observations, timesteps is the duration of a window, and features is the number of features.

4.5.2 Network Architecture Layers

4.5.2.1 Hidden Layer

When creating an LSTM model, you must think about how many hidden layers the model will have, how many LSTM cells will be employed in each layer, and what the dropout will be. There is no correct or incorrect method for determining the number of hidden layers or cells

within each layer. The learning convergence rises as the number of layers increases. As a result, this study attempted to reduce the number of concealed layers.

The number of hidden layers depends on how the LSTM model is implemented since the number of cells and layers varies. However, the layers are typically one to eight (Fulfillment & Engineering, 2019), and the cells in each layer can have the same number of cells to discover the best structure.

In the proposed LSTM model, two hidden layers were used. The model is divided into two layers: LSTM and Dense layer. The PSO algorithm is used to optimize the layer weights.

4.5.2.2 Dense Layer

A dense layer is a densely connected NN layer (Ouhame et al., 2021), in which each cell in the next layer is connected by a dense layer. Building a model of hidden layers followed by several dense layers is how effective models are seen when using dense layers. A dense layer was employed in the proposed model.

4.5.2.3 Number of Layer

The LSTM model would have three layers, one hidden layer, and one input and output layer, according to this thesis. The output of the hidden layer is connected to the output layer i.e., hidden layer → dense layer, where → represents the connection between layers.

4.6 Optimal Hyperparameter

Some hyperparameters must be properly set up and updated when creating the LSTM model so that a credible estimate may be made when the proposed model is tested again. Inspired by this work (Brownlee, n.d.) on how empiric tests could be used to discover the appropriate hyperparameters, which would lead to improved accuracy and reduce the risk of over-fitting the results when compared to the suggested model and data without empiric tests. The LSTM model was constructed using default hyperparameters that better suit this scenario when undertaking an empiric evaluation, according to numerous studies that were regarded to be highly useful. Then, taking each hyperparameter one at a time, try to determine the best value for that hyperparameter. By analyzing the LSTM model by back-testing it with the test results, the best value for a hyperparameter is calculated.

4.7 Number of Epoch

The epoch is the time when all of the training data has been transferred over the network; consequently, one epoch is an iteration of all of the training data being sent across the network.

When the training data is spread over the network, it is divided into batches of 2, with 2 being the default batch size. This indicates that the first two samples from the training data (0-1) are collected and trained on the network, followed by the next two samples (2-3) and the creation of the network. The epoch will continue until the network has propagated all samples and one epoch has passed across the network.

4.8 Number of LSTM Cell in Each Layer

Every hidden and dense layer has a number of LSTM Cells; finding the most appropriate number of cells for each layer is required. The network was trained using different values for each cell in the hidden and dense layers, with the cells in each layer being set to 32 for the LSTM cell and 1 for the dense layer because that value has the minimum MSE in this situation.

4.9 Window

Empirical tests are used to determine the optimal length of a window; however, the proposed model does not want to reduce the size of the window because it would lose longer dependencies, causing it to overlook important features. When training the LSTM model, the simplest technique to determine the most appropriate window size is to run an empiric test on multiple window sizes and find the value of the minimum MSE on the training data. The proposed model made use of a 48 window-size.

4.10 LSTM Loss Function

During fast learning preparation, the loss function determines the distance between an LSTM model's output and the target output. The validation data set by the user produces the best results; 5% of the training data was utilized to establish the validation data. Because the data performance of the training is compared to the validation data at each epoch, this avoids over-fitting by stopping the model during training. It could be over-fitting if the training loss decreases while the validation grows at the same time(Hewamalage et al., 2020). Because it is often utilized for time series prediction when picking a loss function, the MSE was chosen as the proposed loss function.

4.11 Optimizer and PSO parameter setting

The Adam optimizer was suggested to be used as a default in the LSTM model because of its high efficiency and speedy convergence compared to the other potential optimizers(Brownlee, n.d.). The proposed model set the learning rate to 0.0003 while utilizing the Adam optimizer.

Furthermore, as it is discussed in section 4.5.1, the particle swarm optimization factors are manually set to be $c_1$is set to 0.5, c_2 is set as 0.3 and w is set to be 0.9.

4.12 Model Training

Model development, model compilation, and model training are the three essential components of the proposed learning module. Create a model that suits the proposed aim first, and then compile it by changing the hyperparameters it requires. The model will fit with the dataset to begin learning after detailing the first two phases.

4.13 Error Calculation

The RMSE and MAE are two evaluation metrics used to measure the efficiency of the models in the proposed method. It can be translated back to the original units of the predictions by taking the square root of the mean squared error score. The root mean squared error, sometimes known as the RMSE, the equation for RMSE and MAE are shown below. Details of the measurements are discussed in chapter three.

$$\boldsymbol{RMSE} = \sqrt{\frac{\mathbf{1}}{\boldsymbol{n}}\sum_{\boldsymbol{n=1}}^{\boldsymbol{n}} \quad (\boldsymbol{y_i} - \boldsymbol{y_i})\mathbf{2}}$$ Equation 4.7 (Anupama et al., 2021)

$$\boldsymbol{MAE} = \frac{\mathbf{1}}{\boldsymbol{n}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} |\boldsymbol{x_i} - \hat{\boldsymbol{x}}_{\boldsymbol{i}}|$$ Equation 4.8 Anupama et al., 2021)

4.14 Prediction

The proposed test data will be fed into the trained model in this stage. The trained model will loop through the dataset with the batch size specified and return a value for the predicted machine resource utilization. The anticipated value will reverse convert to compare and visualize it with the actual test data set after getting the expected values of machine resource utilization. The model prediction result analysis of the model is plotted and shown in chapter six.

CHAPTER FIVE

IMPLEMENTATION DETAILS

5.1 Overview

The implementation of the proposed approach is discussed in this chapter. Cloud resource consumption prediction, data pretreatment, hyperparameter tuning, and future prediction, as well as code segmentation, are all implemented. Significant code segments were presented in this section, while important detail codes were depicted in the appendix section.

5.2 Working Environments

5.2.1 Desktop Computers with Hardware Utilities

- Hard Disk: configured terabyte added two-tetra bytes of storage.
- SSD: configured SSD is 128GB.
- Processor: Inter(R) Core(TM) i7-7700HQ CPU @ 2.80GHZ (8 CPUs), ~2.8GHz
- Graphical purpose processing Unit: for high-speed time computation during training. NVIDIA GeForce GTX 1050 super installed.
- Installed Memory: Physical Memory (RAM) 16.00 GB. 5.2.2

5.2.2 Desktop Computers with Software Configuration

- OS: Microsoft Windows 10 Pro, x64-based PC, Version 10.0.19041 Build 1904.
- System Model: - HP Pavilion Power Laptop 15-cb0xx.
- GPU optimizer: the latest version of the NVIDIA GPU optimizer and scheduler.
- CUDA toolkit.
- CuDNN.

5.3 Setup Environments

5.3.1 Application Software

Anaconda: The proposed approach is implemented using an anaconda application version 4.8 with 64-bit support, which is used to install the latest version of Python with its many modules and IDEs.

5.3.2 Integrated Development Environment and Editors

Jupyter Notebook: - is a popular and highly recommended IDE for academics who need to update Python code, display diagrams, training and testing charts, and track processing progress. The proposed solution is implemented using the Jupyter notebook version 6.1.5, which is installed via anaconda installation.

Google Colab:- Using the Tensor Processing Unit (TPU), General Processing Unit (GPU), Random Access Memory, and Large Storage Space, a cloud-based Jupiter notebook can improve computation time by uploading data sets for training the results for up to 9 hours.

Visual-Studio-Code: - is an IDE that can be easily configured to work with various Python environments and can be used to edit Python, C++, and other programming language code. In addition to the Jupiter notebook, a version 1.51.1 with the community and 64-bit support has been installed.

5.3.3 Programming Language and Module Libraries

Python: - Due to version incompatibility, the recommended solution was implemented using the Python programming language version 3.8. On the most recent version of Python, several DL modules do not install successfully.

Kera's: - with version 2.3.0, which used to load the pre-trained model, used to apply convolution, pooling, time series generator, and other deep learning processes.

TensorFlow: - with version 2.3.0 and above is used to connect the retrained model and feature fusion model graphs Tensor board with version 2.3.0 and above to visualize training progress in deep learning.

5.4 Training Procedure

- The training procedure in this proposed research
- Collect massive dataset from real-world cloud service providers
- Pre-processing dataset
- Scaling and transforming the data by train and test split
- Building the model
- Compile the model
- Train the model

5.5 Implementation Techniques for Dataset Processing

5.5.1 Dataset Description

In this thesis, a bitbrain dataset obtained from real cloud environments is used to test the performance and verify the method's effectiveness. The following Figure illustrates the code snip of reading the dataset.

```
dataset = read_csv('mp792.csv', header=0,parse_dates=[0], index_col=0,squeeze=True)
#del dataset['Unnamed: 4']
data = dataset.resample('h').mean()
print(data.shape)
plt.plot(data)
plt.show()
values1 = data.values
```

Figure 5.12 Dataset Description

5.5.2 Preparing the Dataset

There were too many features in the original dataset that were unnecessary for cloud resource usage prediction. Since then, non-predictive features have been dropped, and outlier, null value, and string data that can be altered have been fixed, while those that cannot have been adjusted have been dropped, resulting in the loaded dataset being saved and opened again by the new feature optimization. In all datasets utilized to train the proposed model, the suggested model utilised a single machine trace.

5.5.3 Scaling and Transforming the Data

Before putting the data into the suggested model, it goes through many rounds of pre-processing. To begin, all datasets' normal values are scaled to a range of 0 to 1 using the minimum-maximum scalar. A code snippet for scaling and modifying the dataset is shown in the accompanying Figure. These scaled traces are employed in the proposed model's training and testing.

```
dataset = read_csv('mp792.csv', header=0,parse_dates=[0], index_col=0,squeeze=True)
#del dataset['Unnamed: 4']
data = dataset.resample('h').mean()
print(data.shape)
plt.plot(data)
plt.show()
values1 = data.values

# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values1)
```

Figure 5.13 Scaling and transforming the data

5.5.4 Create the Model

Several features, including as an optimizer, a loss, and metrics, are used to form the sequential model. Using the Keras Sequential API, the network is built up one layer at a time. The LSTM cells' layer is the network's beating heart. The network gains additional representational strength from the fully linked dense layer with relu activation.

```
model = Sequential()
model.add(LSTM(32, return_sequences=False, input_shape=(x_train.shape[1], x_train.shape[2])), name='layer_1')
#modelGA.add(LSTM(32, return_sequences=False, input_shape=(x_train.shape[1], x_train.shape[2]), name='layer_3'))
model.add(Dense(1, activation='linear', name='layer_2'))
model.compile(loss="mean_squared_error", optimizer='sgd', metrics=["mean_squared_error", r_square_loss])
model.summary()
historyNN = model.fit(x_train, y_train, validation_data=(x_val,y_val), epochs=10)


plt.plot(historyNN.history['loss'])
plt.plot(historyNN.history['val_loss'])
plt.title('Model Loss Plot')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

Figure 5.14 Model Create

5.5.5 Compile the Model

The model is constructed and trained using the MSE loss utilizing the Adam optimizer (a variation on stochastic gradient descent). By altering the training parameters, the network will try to minimise log loss during training (weights). Back-propagation is used to calculate the parameter gradients, which are then adjusted with the optimizer.

```
model.compile(loss="mean_squared_error", optimizer='sgd', metrics=["mean_squared_error", r_square_loss])
model.summary()
```

Figure 5.15 Compiling the model

5.6 Model Training

Model training is the phase where the best combination of weights and biases are fitted to the machine learning model to minimize a loss function over a prediction range. Once the network is built, the model will be trained. The proposed model training iteration epochs are 10. The model then trained using the code below.

```
historyNN = model.fit(x_train, y_train, validation_data=(x_val,y_val), epochs=10)
```

Figure 5.16 Model training

5.6.1 Test Prediction

The following code shows the test prediction with back tested traces.

```
### Checking the performance of the LSTM NN us
model.summary()

predictY = model.predict(x_test)

time = np.arange(predictY.shape[0])
plt.plot(time, predictY, y_test)
plt.xlabel('time')
plt.ylabel('amplitude')
plt.show()

lossNN_GA = model.evaluate(x_test, y_test)
```

Figure 5.17 Test prediction

5.6.2 Visualization (Data-frame and Plotting)

The following code snippet shows the visualization of the actual and training data.

```
predictY = model.predict(x_test)

time = np.arange(predictY.shape[0])
plt.plot(time, predictY, y_test)
plt.xlabel('time')
plt.ylabel('amplitude')
plt.show()

lossNN_GA = model.evaluate(x_test, y_test)

print('The loss output for test dataset is: %f' % lossNN_GA[0])

print('We see that the loss output for test dataset using Back Propagation is: %f, while the loss output using GA is: %f' % (lossNN[0], lossNN_GA[0]))

plt.plot(time, predictY, y_test)
plt.xlabel('time')
plt.ylabel('amplitude')
plt.show()
```

Figure 5.18 Plotting of the real-time prediction

5.7 Implementation of Model Evaluation

5.7.1 RMSE

The prediction method's performance is measured using RMSE. The RMSE is calculated to measure the error or accuracy in the prediction of one's models. Also, it calculates the error based on standard deviations.

```
def rmse(y_true, y_pred):
    return np.sqrt(np.mean(np.square(y_pred - y_true), axis=-1))
```

Figure 5.19 RMSE

CHAPTER SIX

RESULT, EVALUATION AND DISCUSSION

This chapter describes the experiments carried out to evaluate four different prediction models, namely CNN, LSTM, CNN-LSTM, and PSO based Long Short-Term Memory (LSTM) and Long Short-Term Memory (LSTM), using different workload data from a real distributed cloud datacenter with different workload patterns, such as periodic, growing, and unpredicted workloads, using different workload data from a real distributed cloud datacenter with different workload patterns, such as periodic, growing, and unpredicted workloads. The forecast error measurements used to evaluate the developed resource prediction models were Root Mean Squared Error (RMSE). The experimental setup is discussed in Section 6.1. The results are shown in Section 6.2, and they are discussed in Section 6.3.

6.1. Experimental setup

Google Colab, a free Jupyter notebook environment that runs fully in the cloud, was used to conduct the experiment. Python and the Keras package were utilized as the implementation and simulation tools. Experiments were conducted on 6 cloud VMs workload data from Bitbrain for varied workload patterns, as stated in Chapter 3. Various workload patterns are represented by these virtual machines. The authors of [17,18] categorize workload patterns in the cloud computing environment into three types: periodic, growing, and unpredictable workload. Each one shows a common scenario or application. This study will concentrate primarily on CPU prediction for specific workload patterns.

To forecast resource requirements, the experiment used workload data from a real data center. We forecast CPU utilization for the next 5, 15, 45, and 60 minutes based on recent data. We must evaluate how effectively the built prediction models work on new, unseen data, as the evaluation of a model is an estimate of how well the model might function in practice. As a result, the data was divided into two groups in a 75:25 ratio: training data and testing data. The remaining 25% of the data was used to evaluate the model, which included forecasting and computing performance indicators based on expected values.

The Root Mean Square Error is used to assess the produced resource prediction models (RMSE). Which is the most extensively used real-value prediction assessment metric for time series prediction (Anupama et al., 2021). The multivariate prediction models (LSTM, CNN, CNN-LSTM, and PSO-LSTM) that were built were LSTM, CNN, CNN-LSTM, and PSO-LSTM. Each is put to the test on six virtual machines, each with a different workload pattern (2 VMs for each workload type). The RMSE accuracy of prediction models is presented in the following section. The findings are then compared and analyzed to see whether model is better at predicting cloud resource requirements.

6.2 Baseline Methods Compared with the proposed Model

For the experiment of our model, we have selected three state of the art models, adopted from(Leka et al., 2022) where this research study author is also participated as a co-author. The adopted models are created and implemented according to the configurations stated in the paper. Therefore, the proposed model's prediction performance is compared with the baseline models such as CNN, CNN-LSTM adopted from (Leka et al., 2022) and LSTM model developed in this research study. The comparative result of the proposed PSO-LSTM model with the mentioned baseline models for RMSE are summarized in Table 6.2.

6.3. Evaluation of results

To compare the results of the constructed resource prediction models, the average RMSE is employed as a performance metric. For each workload category, Table 6.2 shows the average RMSE of the four prediction models for the CPU utilization value over several future time intervals (5-minutes, 15 minutes and 45 and 60 minutes). The tables depict the three types of workloads: predictable workload, unexpected workload, and growing workload. The black values represent the best results for a specific model for the projected interval.

Table 6.2 RMSE and MAE of the models for different time intervals

RMSE

Workload Type	CNN(Leka et al., 2022)				LSTM				CNN-LSTM(Leka et al., 2022)				PSO-LSTM			
	5 min	15 min	45min	60 min	5 min	15 min	45min	60 min	5 min	15 min	45min	60 min	5 min	15 min	45min	60 min
Periodic	18430.274	25958.149	18868.017	18857.051	**0.001**	0.01	0.026	0.036	22575.993	28983.706	24524.408	20732.85	0.002	**0.005**	0.009	**0.028**
Growing	131919.35	153219.023	123263.39	122235.63	0.006	0.015	0.052	0.044	126976.641	158901.47	195177.994	179432.4	**0.001**	**0.004**	**0.012**	**0.03**
Unpredictable	3207.371	3874.936	5283.867	5981.152	0.011	0.014	0.029	**0.022**	4230.645	5237.51	6305.597	5992.319	**0.002**	**0.006**	**0.017**	0.038

MAE

Workload Type	CNN(Leka et al., 2022)				CNN-LSTM(Leka et al., 2022)			
	5 min	15 min	45min	60 min	5 min	15 min	45min	60 min
Periodic	6933.103	13619.234	9125.861	9511.551	9316	11339	11264	10301
Growing	76498.365	78052.693	65143.328	103371.788	70699	77602	127748	115857
Unpredictable	916.714	1116.451	1716.546	2119.976	1096	1575	1840	1836.9

Figures 6.20-6.31 compare the accuracy of the proposed prediction model of anticipated CPU consumption to actual (expected) CPU usage for the three workload patterns 5 minutes, 15 minutes, 45 minutes, and 60 minutes in the future. It can be seen that the PSO-LSTM model's anticipated CPU usage is close to the actual CPU utilization. The Root Mean Squared Error is used to evaluate the suggested prediction model (RMSE).

As it is shown in Table 6.30 the proposed model out performs the other models for the 5 minutes interval of periodic workload patterns. It can be seen from Figure 6.20 that the PSO-LSTM model's anticipated CPU usage is close to the actual CPU utilization.

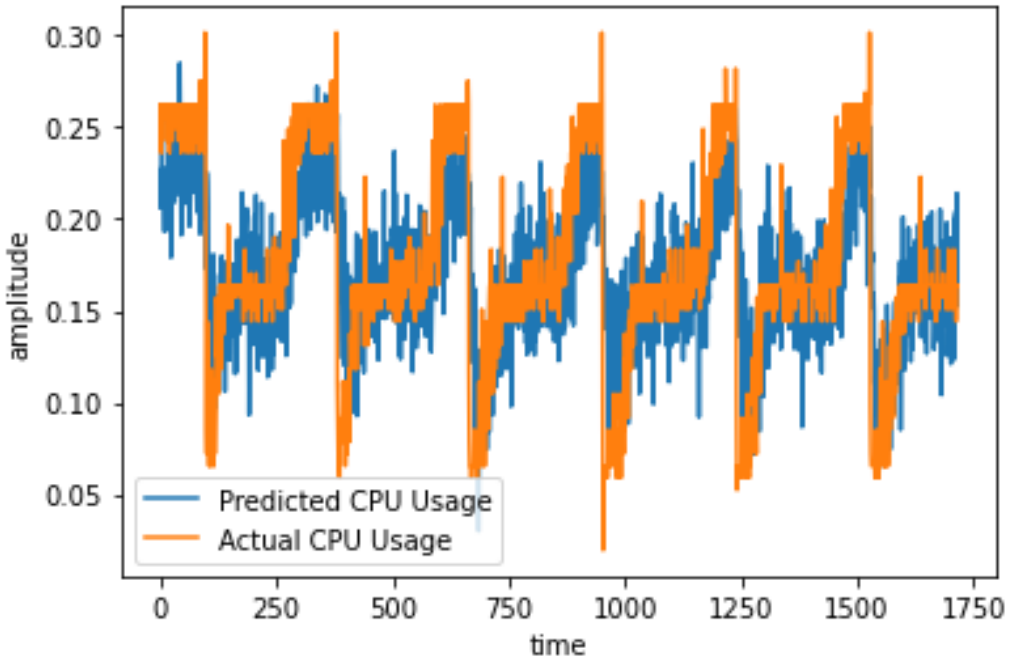

Figure 6.20 PSO LSTM for periodic 5minutes-actual vs predicted CPU usage

As it is shown in Table 6.21 the proposed model out performs the other models for the 15 minutes interval of periodic workload patterns. It can be seen from Figure 6.21 that the PSO-LSTM model's anticipated CPU usage is close to the actual CPU utilization.

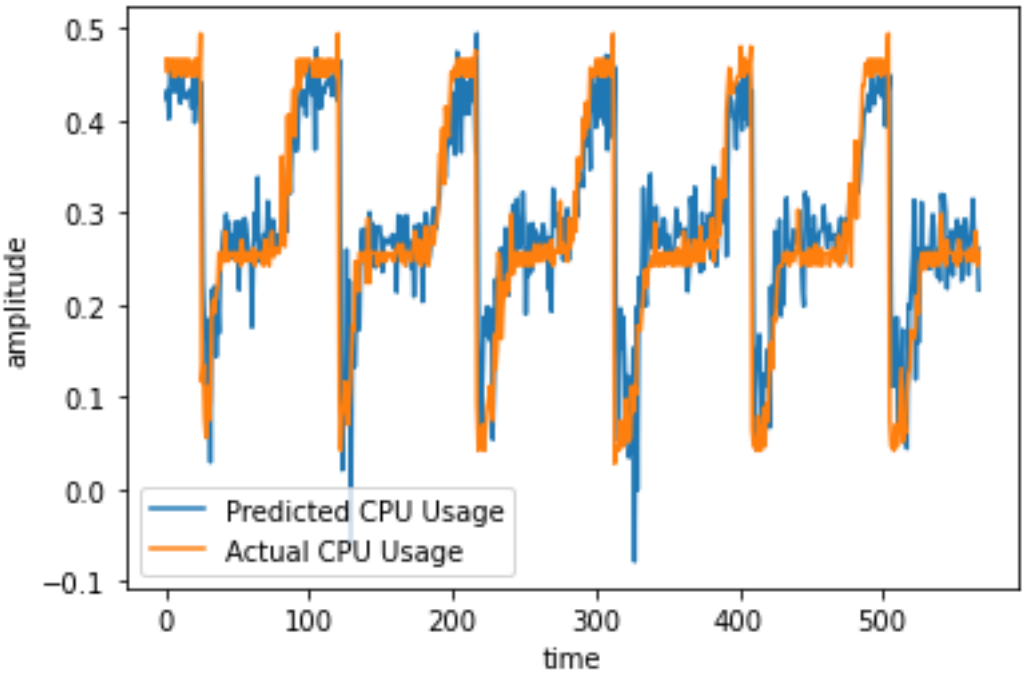

Figure 6.21 PSO LSTM periodic 15min-actual vs predicted CPU usage

As it is shown in Table 6.22 the proposed model out performs the other models for the 45 minutes interval of periodic workload patterns. It can be seen from Figure 6.22 that the PSO-LSTM model's anticipated CPU usage is close to the actual CPU utilization.

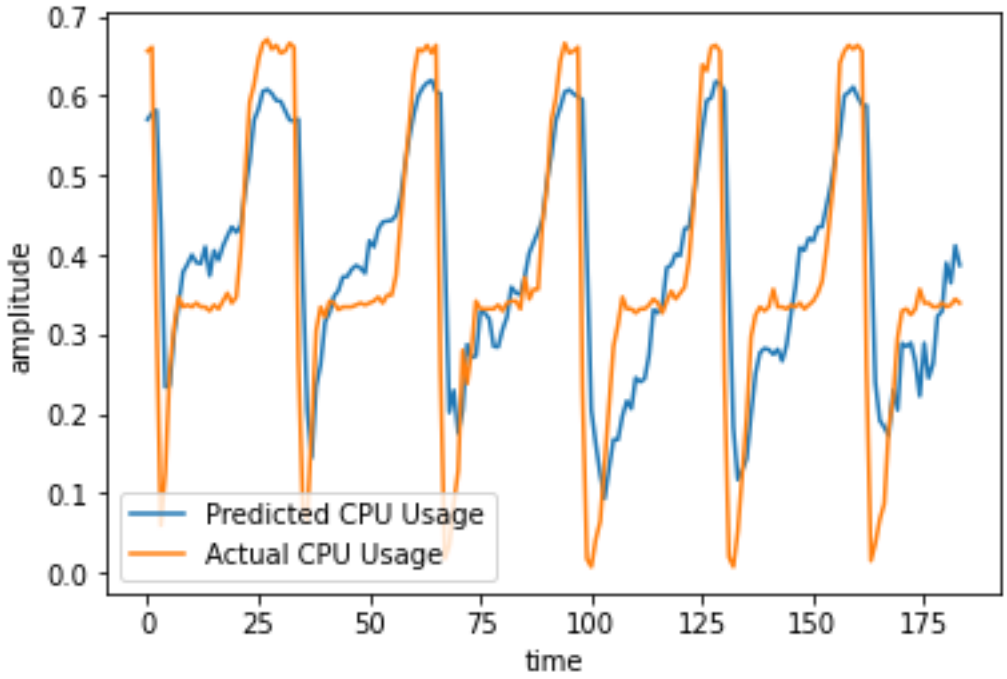

Figure 6.22 PSO LSTM periodic 45min -actual vs predicted CPU usage

As it is shown in Table 6.23 the proposed model out performs the other models for the 60 minutes interval of periodic workload patterns. It can be seen from Figure 6.23 that the PSO-LSTM model's anticipated CPU usage is close to the actual CPU utilization.

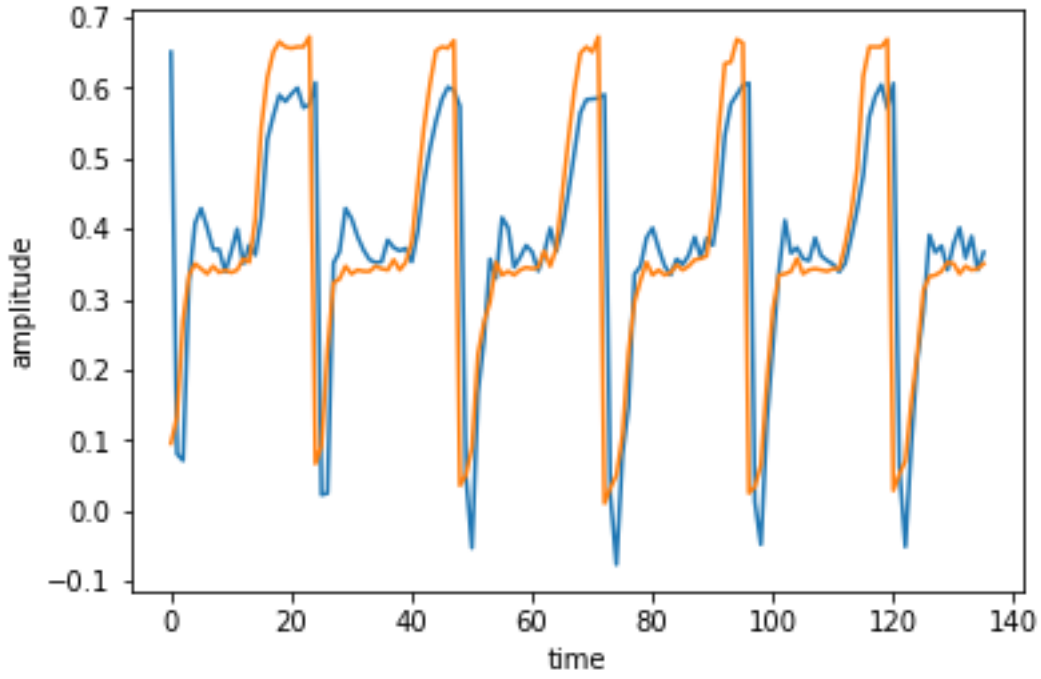

Figure 6.23 PSO LSTM periodic 60min-actual vs predicted CPU usage

As it is shown in Table 6.24 the proposed model out performs the other models for the 5 minutes interval of growing workload patterns. It can be seen from Figure 6.24 that the PSO-LSTM model's anticipated CPU usage is close to the actual CPU utilization.

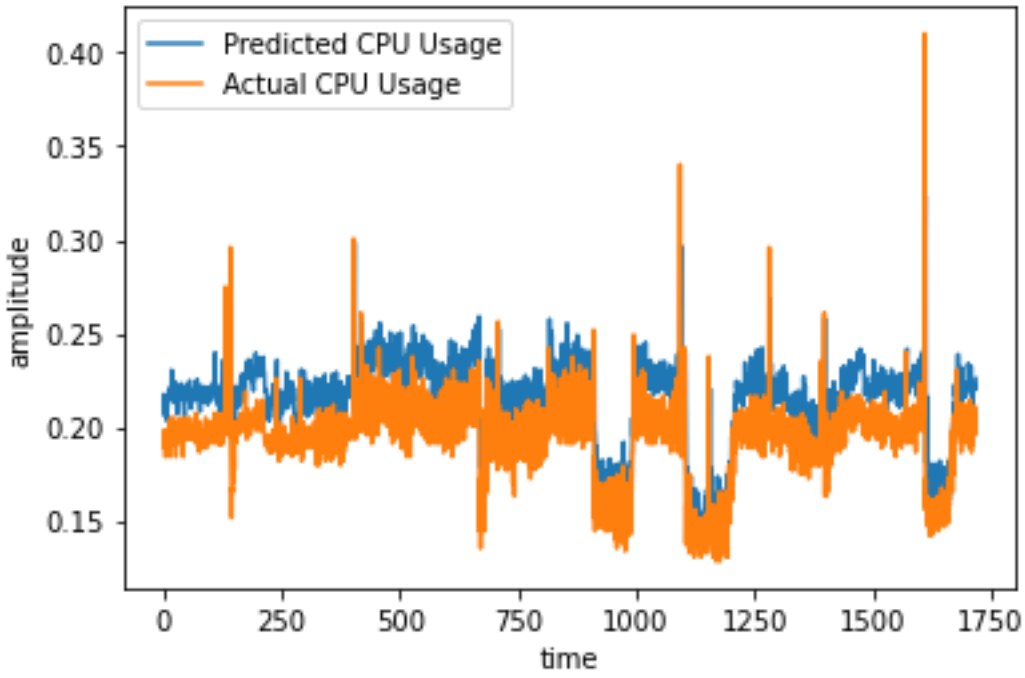

Figure 6.24 PSO LSTM Growing 5min-actual vs predicted CPU usage

As it is shown in Table 6.25 the proposed model out performs the other models for the 15 minutes interval of growing workload patterns. It can be seen from Figure 6.25 that the PSO-LSTM model's anticipated CPU usage is close to the actual CPU utilization.

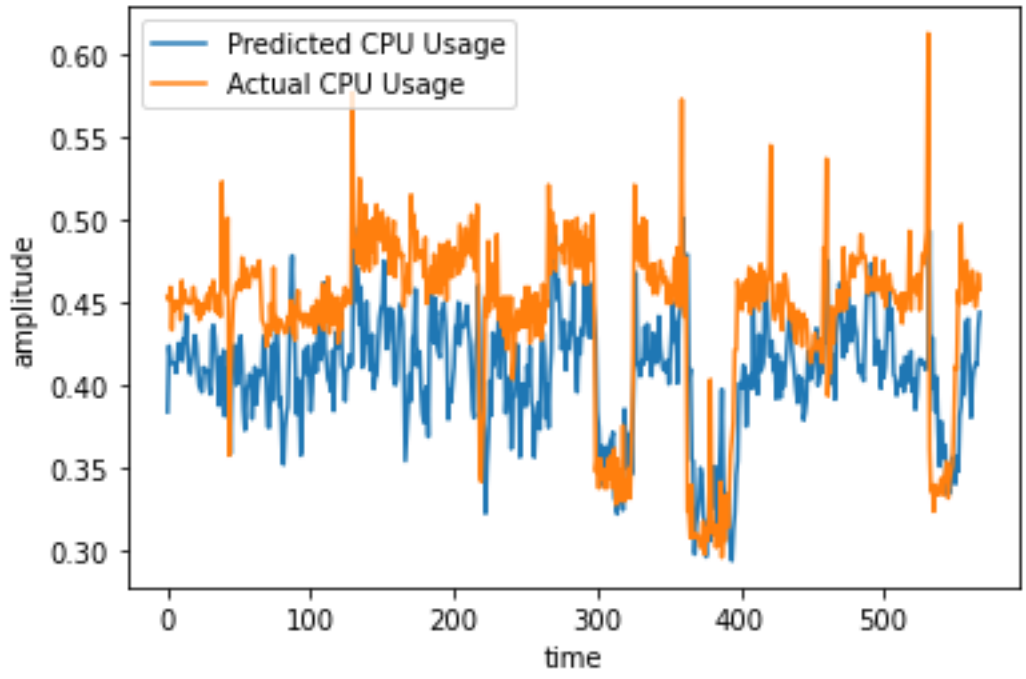

Figure 6.25 PSO LSTM Growing 15min-actual vs predicted CPU usage

As it is shown in Table 6.26 the proposed model out performs the other models for the 45 minutes interval of growing workload patterns. It can be seen from Figure 6.26 that the PSO-LSTM model's anticipated CPU usage is close to the actual CPU utilization.

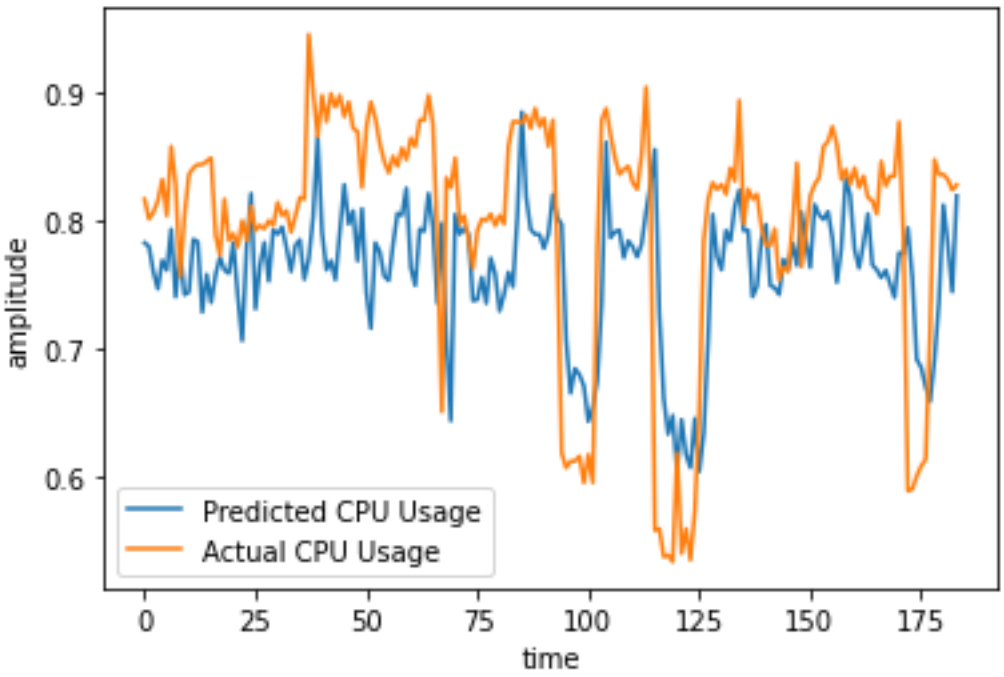

Figure 6.26 PSO LSTM Growing 45min-actual vs predicted CPU usage

As it is shown in Table 6.27 the proposed model out performs the other models for the 60 minutes interval of growing workload patterns. It can be seen from Figure 6.27 that the PSO-LSTM model's anticipated CPU usage is close to the actual CPU utilization.

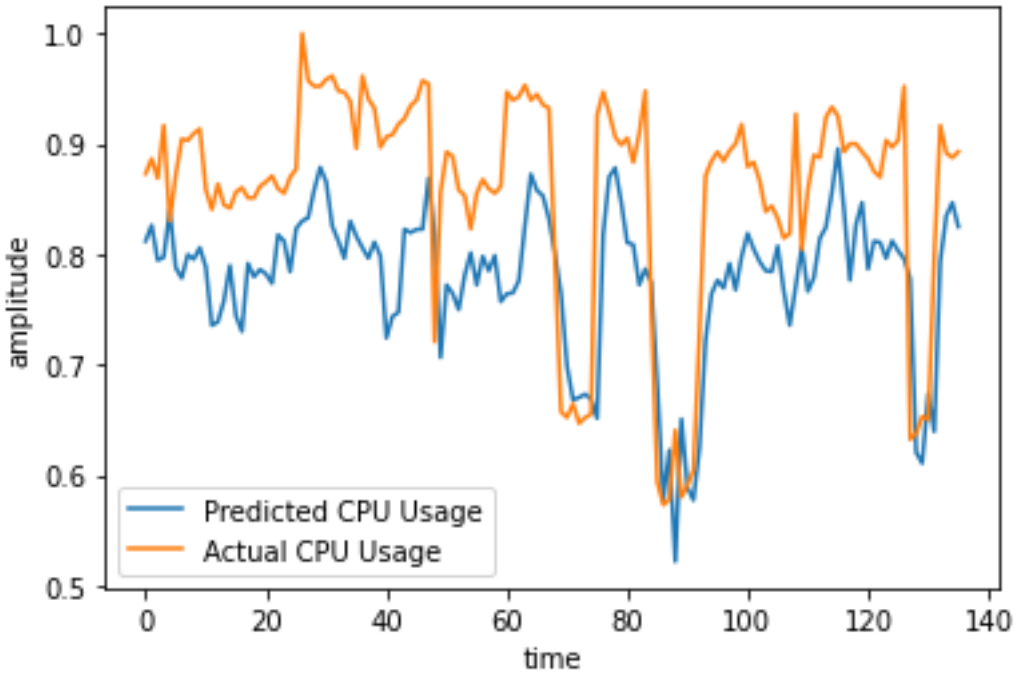

Figure 6.27 PSO LSTM Growing 60min-actual vs predicted CPU usage

As it is shown in Table 6.28 the proposed model out performs the other models for the 5 minutes interval of unpredictable workload patterns. It can be seen from Figure 6.28 that the PSO-LSTM model's anticipated CPU usage is close to the actual CPU utilization.

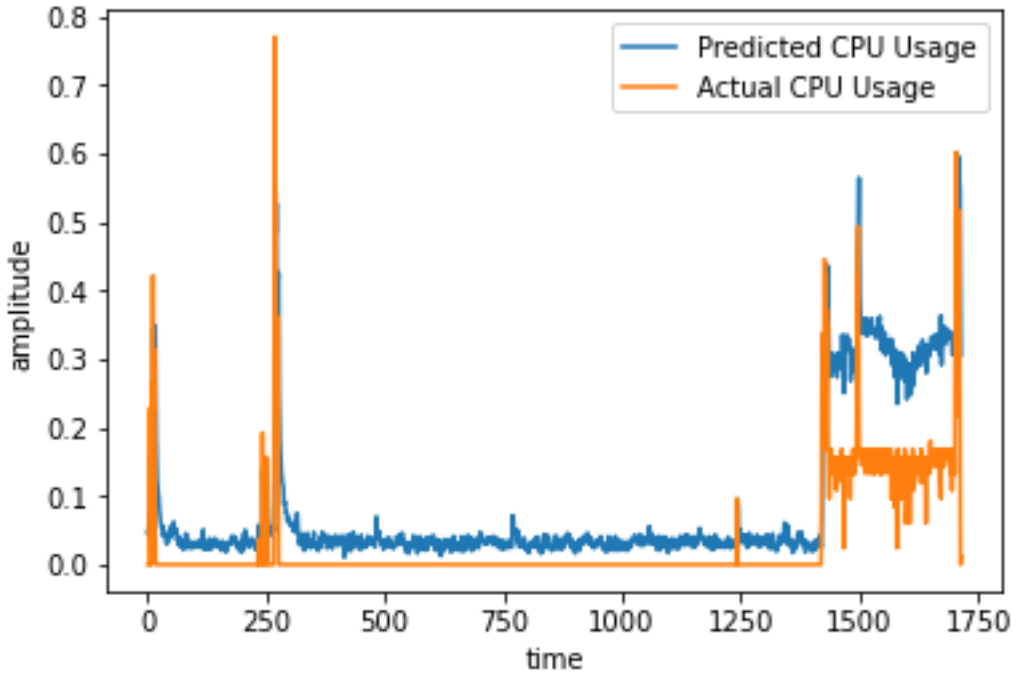

Figure 6.28 PSO LSTM Growing 5 min-actual vs predicted CPU usage

As it is shown in Table 6.29 the proposed model out performs the other models for the 15 minutes interval of unpredictable workload patterns. It can be seen from Figure 6.29 that the PSO-LSTM model's anticipated CPU usage is close to the actual CPU utilization.

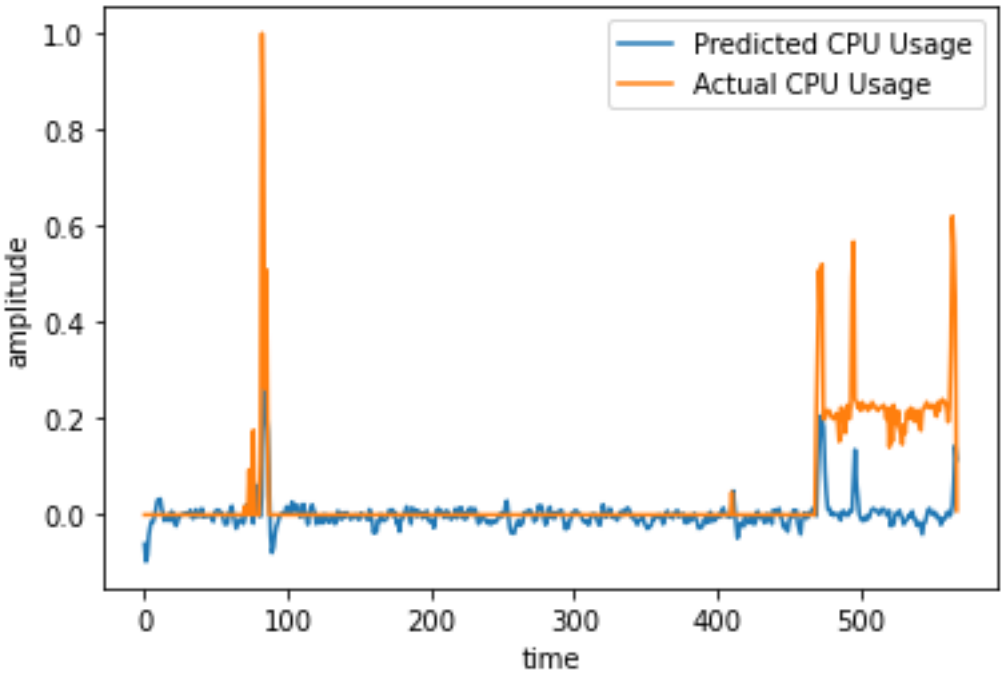

Figure 6.29 PSO LSTM Unpredictable 15 min-actual vs predicted CPU usage

As it is shown in Table 6.29 the proposed model out performs the other models for the 45 minutes interval of unpredictable workload patterns. It can be seen from Figure 6.29 that the PSO-LSTM model's anticipated CPU usage is close to the actual CPU utilization.

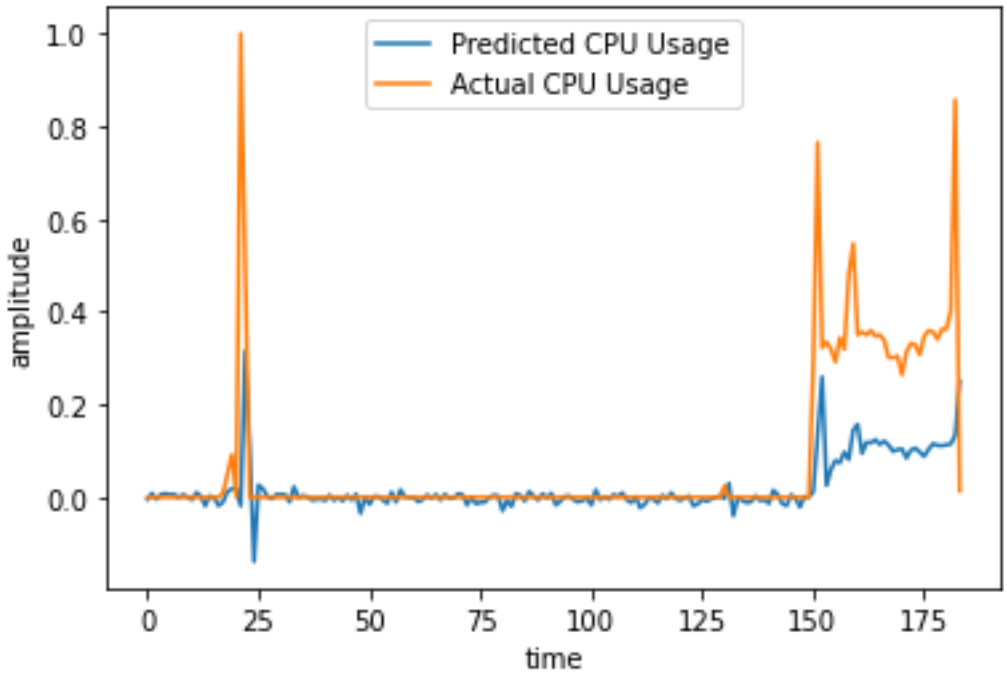

Figure 6.30 PSO LSTM Unpredictable 45 min-actual vs predicted CPU usage

As it is shown in Table 6.30 the proposed model out performs the other models for the 60 minutes interval of unpredictable workload patterns. It can be seen from Figure 6.30 that the PSO-LSTM model's anticipated CPU usage is close to the actual CPU utilization.

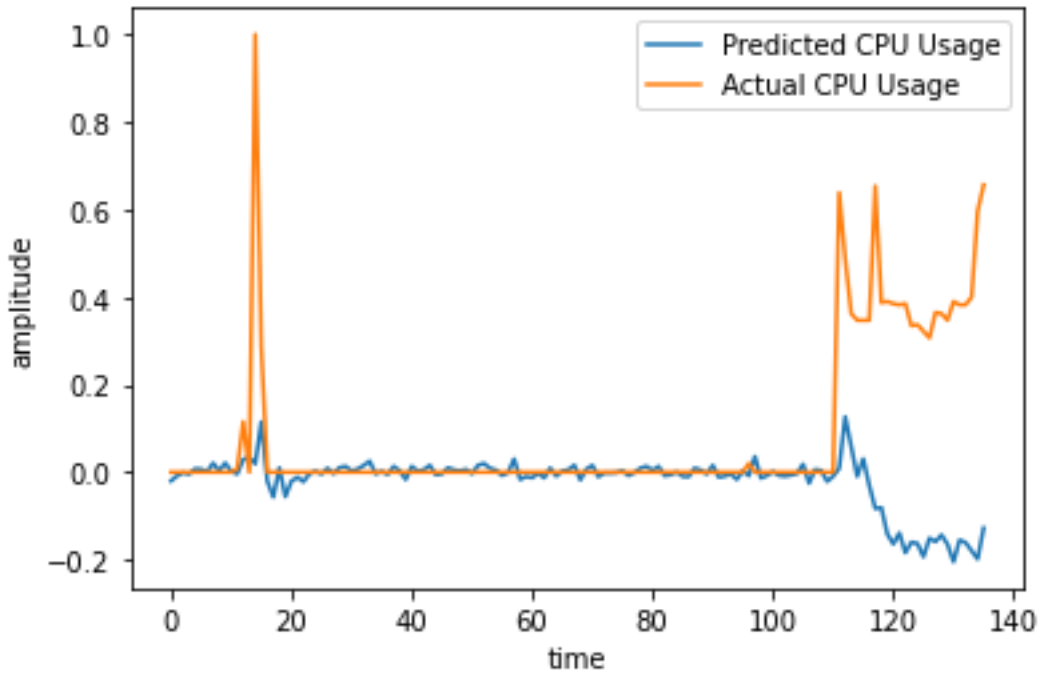

Figure 6.31 PSO LSTM Unpredictable 60 min-actual vs predicted CPU usage

6.3. Discussion

Experiments have proven that the created resource prediction models (LSTM, CNN, CNN-LSTM, and PSO-LSTM) are accurate in predicting incoming workload patterns. The proposed model wins for all workload patterns, as evidenced by the experiment results. For all workload patterns, the PSO-LSTM model clearly outperforms alternative prediction models (CNN, CNN-LSTM and LSTM). Table (7) shows that the PSO-LSTM model has improved accuracy results when RMSE is used for all VMs with periodic, growing, and unpredictable workloads. The performance comparison of the proposed prediction models for single VM CPU use prediction per category is shown in Figure 6.21.

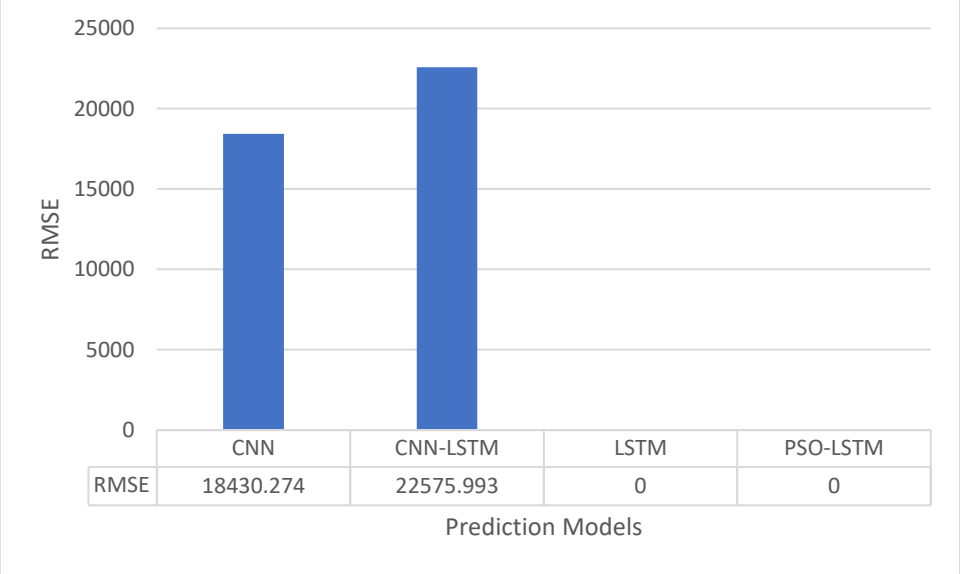

Figure 6.32 Comparison of periodic VM resource prediction of CPU usage

Finally, based on the data, it can be stated that using a particle swarm optimization-based LSTM to estimate future cloud resource needs for all workload patterns is the optimum model for doing so. Although starting with a univariate model reduces model complexity and reduces computation time, multivariate models should also be explored. Because the majority of modern cloud apps are built up of a collection of interconnected service components that execute various tasks. These service components are deployed over a variety of heterogeneous, non-independent virtual machines.

The goal of this study is to improve the accuracy of proactive auto-scaling system forecasts. According to the findings, the proposed PSO-based LSTM has a lot of potential for use in the analysis phase of proactive auto-scaling, as it produced the greatest outcomes for the three workload patterns with the least amount of modification. LSTM produces improved results when given a large amount of data and enough training epochs, such as cloud data.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

An effective particle swarm based deep learning model for predicting VM workload in a cloud data center is proposed in this research study. For estimating VM CPU utilization, four deep learning models were developed, deployed, and evaluated: CNN, LSTM, CNN-LSTM, and PSO-LSTM. The suggested model was tested against state-of-the-art deep learning models such as CNN, LSTM, and CNN-LSTM adopted from (Leka et al., 2022) to evaluate how accurate and effective it was at forecasting the future workload of virtual machines. The experimental results suggest that the proposed model can achieve greater prediction accuracy, demonstrating that the proposed technique surpasses all others in cloud workload forecasting for the three virtual machine workload patterns.

7.2 Future work

In the future, we can look into algorithms that automatically classify VM workloads, allowing us to choose the best resource forecast model based on workload trends. Furthermore, feature selection can be done automatically, allowing the most significant features to be included as input to the prediction model for more accurate outcomes. Finally implementing the proposed model on real cloud computing data center is a future plan by considering the factor of time constraints.

References

Agga, A., Abbou, A., Labbadi, M., & El Houm, Y. (2021). Short-term self consumption PV plant power production forecasts based on hybrid CNN-LSTM, ConvLSTM models. *Renewable Energy*, *177*, 101–112. https://doi.org/10.1016/j.renene.2021.05.095

Anupama, K. C., Shivakumar, B. R., & Nagaraja, R. (2021). Resource Utilization Prediction in Cloud Computing using Hybrid Model. *International Journal of Advanced Computer Science and Applications*, *12*(4), 373–381. https://doi.org/10.14569/IJACSA.2021.0120447

Banerjee, S., Roy, S., & Khatua, S. (2021). Efficient resource utilization using multi-step-ahead workload prediction technique in cloud. *Journal of Supercomputing*, *77*(9), 10636–10663. https://doi.org/10.1007/s11227-021-03701-y

Barthwal, V., Manmohan, P., Rauthan, S., & Verma, R. (2019). *Virtual Machines Placement using Predicted Utilization of Physical Machine in Cloud Datacenter*.

Bhatia, S. K., Tiwari, S., Ruidan, S., Chandra, M., & Editors, K. K. M. (2019). *Advances in Intelligent Systems and Computing 1158 Advances in Computer , Communication and Computational Sciences*.

Brownlee, J. (n.d.). *Deep Learning for Time Series Forecasting Predict the Future with MLPs , CNNs and LSTMs in Python*.

Chen, J., & Wang, Y. (2019). A Hybrid Method for Short-Term Host Utilization Prediction in Cloud Computing. *Journal of Electrical and Computer Engineering*, *2019*. https://doi.org/10.1155/2019/2782349

Computing, C. (n.d.). *A Proactive Mechanism To Improve Workload Prediction For Cloud Services Using Machine Learning Sumedh Gursale*.

Dang-Quang, N. M., & Yoo, M. (2021). Deep learning-based autoscaling using bidirectional long short-term memory for kubernetes. *Applied Sciences (Switzerland)*, *11*(9). https://doi.org/10.3390/app11093835

Fulfillment, P., & Engineering, I. S. (2019). *Office of Graduate Studies Adama Science and Technology University Adama, Ethiopia June 2019*. *June*.

Hasan Shuvo, M. N., Shahriar Maswood, M. M., & Alharbi, A. G. (2020). LSRU: A Novel Deep Learning based Hybrid Method to Predict the Workload of Virtual Machines in Cloud Data Center. *2020 IEEE Region 10 Symposium, TENSYMP 2020*, *June*, 1604–1607. https://doi.org/10.1109/TENSYMP50017.2020.9230799

Hewamalage, H., Bergmeir, C., & Bandara, K. (2020). *Recurrent Neural Networks for Time Series Forecasting: Current Status and Future Directions*.

Hsu, C. H., Slagter, K. D., Chen, S. C., & Chung, Y. C. (2014). Optimizing energy consumption with task consolidation in clouds. *Information Sciences*, *258*, 452–462. https://doi.org/10.1016/j.ins.2012.10.041

Kalyampudi, P. S. L., Krishna, P. V., Kuppani, S., & Saritha, V. (2021). A work load prediction strategy for power optimization on cloud based data centre using deep machine learning. *Evolutionary Intelligence*, *14*(2), 519–527. https://doi.org/10.1007/s12065-019-00289-4

Karim, M. E., Maswood, M. M. S., Das, S., & Alharbi, A. G. (2021). BHyPreC: A Novel Bi-LSTM Based Hybrid Recurrent Neural Network Model to Predict the CPU Workload of Cloud Virtual Machine. *IEEE Access*, *9*, 131476–131495. https://doi.org/10.1109/ACCESS.2021.3113714

Leka, H. L., Fengli, Z., Kenea, A. T., Tegene, A. T., Atandoh, P., & Hundera, N. W. (2022). *A Hybrid CNN-LSTM Model for Virtual Machine Workload Forecasting in Cloud Data Center*. 474–478. https://doi.org/10.1109/iccwamtip53232.2021.9674067

Mandal, R., Mondal, M. K., Banerjee, S., & Biswas, U. (2020). An approach toward design and development of an energy-aware VM selection policy with improved SLA violation in the domain of green cloud computing. *Journal of Supercomputing*, *0123456789*. https://doi.org/10.1007/s11227-020-03165-6

Masdari, M., & Khezri, H. (2020). Efficient VM migrations using forecasting techniques in cloud computing: a comprehensive review. *Cluster Computing*, *0123456789*. https://doi.org/10.1007/s10586-019-03032-x

Masdari, M., & Khoshnevis, A. (2020). A survey and classification of the workload forecasting methods in cloud computing. *Cluster Computing*, *23*(4), 2399–2424. https://doi.org/10.1007/s10586-019-03010-3

Mashhadi Moghaddam, S., O'Sullivan, M., Walker, C., Fotuhi Piraghaj, S., & Unsworth, C. P. (2020). Embedding individualized machine learning prediction models for energy efficient VM consolidation within Cloud data centers. *Future Generation Computer Systems*, *106*, 221–233. https://doi.org/10.1016/j.future.2020.01.008

Merangin, D. I. D., Pattiselanno, F., Mentansan, G., Nijman, V., Nekaris, K. A. I., Pratiwi, A. I. N., Studi, P., Nutrisi, I., Makanan, D. A. N., Peternakan, F., Penulisan, P., Ilmiah, K., Berbagai, P., Cahaya, I., Lapangan, D. I., Eropa, A., Geometry, R., Analysis, G.,

Nasution, R. D., … Bismark, M. (2018). No 主観的健康感を中心とした在宅高齢者における 健康関連指標に関する共分散構造分析Title. □□□□□ □□□□ 2016), 2(*2*, □□□□□□ □□□□□□□□□. https://doi.org/10.1016/j.gecco.2019.e00539%0Ahttps://doi.org/10.1016/j.foreco.2018.06.029%0Ahttp://www.cpsg.org/sites/cbsg.org/files/documents/Sunda Pangolin National Conservation Strategy and Action Plan %28LoRes%29.pdf%0Ahttps://doi.org/10.1016/j.forec

Nashold, L., & Krishnan, R. (2020). *Using LSTM and SARIMA Models to Forecast Cluster CPU Usage*. http://arxiv.org/abs/2007.08092

Ouhame, S., Hadi, Y., & Ullah, A. (2021). An efficient forecasting approach for resource utilization in cloud data center using CNN-LSTM model. *Neural Computing and Applications*, *9*. https://doi.org/10.1007/s00521-021-05770-9

Qiu, F., Zhang, B., & Guo, J. (2016). *A Deep Learning Approach for VM Workload Prediction in the Cloud*.

Ruan, L., Bai, Y., Li, S., He, S., & Xiao, L. (2021). Workload time series prediction in storage systems: a deep learning based approach. *Cluster Computing*, *0123456789*. https://doi.org/10.1007/s10586-020-03214-y

Saeed, A., & Alhammadi, A. (2020). *Multiple Regression Particle Swarm Optimization for Host Overload and Under-Load Detection*. *10253*, 10253–10261.

Saxena, D., & Singh, A. K. (2021). A proactive autoscaling and energy-efficient VM allocation framework using online multi-resource neural network for cloud data center. *Neurocomputing*, *426*(xxxx), 248–264. https://doi.org/10.1016/j.neucom.2020.08.076

Shishira, S. R., & Kandasamy, A. (2021). BeeM-NN: An efficient workload optimization

using Bee Mutation Neural Network in federated cloud environment. *Journal of Ambient Intelligence and Humanized Computing*, *12*(2), 3151–3167. https://doi.org/10.1007/s12652-020-02474-1

Shuja, J., Ahmad, R. W., Gani, A., Abdalla Ahmed, A. I., Siddiqa, A., Nisar, K., Khan, S. U., & Zomaya, A. Y. (2017). Greening emerging IT technologies: techniques and practices. *Journal of Internet Services and Applications*, *8*(1). https://doi.org/10.1186/s13174-017-0060-5

St-Onge, C., Kara, N., Wahab, O. A., Edstrom, C., & Lemieux, Y. (2020). Detection of time series patterns and periodicity of cloud computing workloads. *Future Generation Computer Systems*, *109*, 249–261. https://doi.org/10.1016/j.future.2020.03.059

Thonglek, K., Ichikawa, K., Takahashi, K., Iida, H., & Nakasan, C. (2019). Improving Resource Utilization in Data Centers using an LSTM-based Prediction Model. *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, *2019-Septe*, 1–8. https://doi.org/10.1109/CLUSTER.2019.8891022

Yadav, M. P., Pal, N., & Yadav, D. K. (2021). Workload prediction over cloud server using time series data. *Proceedings of the Confluence 2021: 11th International Conference on Cloud Computing, Data Science and Engineering*, 267–272. https://doi.org/10.1109/Confluence51648.2021.9377032

Yazdanian, P., & Sharifian, S. (2021). E2LG: a multiscale ensemble of LSTM/GAN deep learning architecture for multistep-ahead cloud workload prediction. In *Journal of Supercomputing* (Vol. 77, Issue 10). Springer US. https://doi.org/10.1007/s11227-021-03723-6

Yuan, C., Tang, Y., Mei, R., Mo, F., & Wang, H. (2021). A PSO-LSTM Model of Offshore Wind Power Forecast considering the Variation of Wind Speed in Second-Level Time

Scale. *Mathematical Problems in Engineering*, *2021*. https://doi.org/10.1155/2021/2009062

Zhang, Q., Yang, L. T., Yan, Z., Chen, Z., & Li, P. (2018). An Efficient Deep Learning Model to Predict Cloud Workload for Industry Informatics. *IEEE Transactions on Industrial Informatics*, *14*(7), 3170–3178. https://doi.org/10.1109/TII.2018.2808910

Zhu, Y., Zhang, W., Chen, Y., & Gao, H. (2019). A novel approach to workload prediction using attention-based LSTM encoder-decoder network in cloud environment. *Eurasip Journal on Wireless Communications and Networking*, *2019*(1). https://doi.org/10.1186/s13638-019-1605-z

APPENDIX

Appendix A: Sample Code for Building, Creating and Compiling the Model

```
import pyswarms as ps
from pyswarms.utils.plotters import (plot_cost_history)
#%matplotlib inline
from pandas import read_csv
import numpy as np
import matplotlib.pyplot as plt
from keras import Sequential
from keras.layers import LSTM, Dropout, Dense, TimeDistributed, Activation, Bidirectional
from sklearn.preprocessing import MinMaxScaler

def rmse_nn(y_true, y_pred):
    from keras import backend
    return backend.sqrt(backend.mean(backend.square(y_pred - y_true), axis=-1))

def r_square_loss(y_true, y_pred):
    from keras import backend as K
    SS_res =  K.sum(K.square(y_true - y_pred))
    SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
    return 1 - ( 1 - SS_res/(SS_tot + K.epsilon()))

dataset = read_csv('C:/Users/ayu/Documents/Experiment/SpiderCodeFromPreviousFile/mp792.csv', header=0,parse_dates=[0], index_c
#dataset = read_csv('C:/Users/Habte/Documents/Ayantu/Experiment/SpiderCodeFromPreviousFile/The Six Dataset/mg794.csv', header=

#del dataset['Unnamed: 4']
data = dataset.resample('15min').mean()
#data = dataset
print(data.shape)
plt.plot(data)
plt.show()
values1 = data.values
```

```
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values1)

# Converting the dataset into Supervised problem
def to_supervised(dataset, window_size=48):
    #dataset_as_np = dataset.to_numpy()
    dataset_as_np = dataset
    X = []
    y = []

    for i in range(len(dataset_as_np)-window_size):
        row = [r for r in dataset_as_np[i: i+window_size]]
        X.append(row)
        label = dataset_as_np[i + window_size][0]
        y.append(label)

    return np.array(X), np.array(y)
x, y = to_supervised(scaled)
# Dataset sampling in the ratio of 25/75
size = int(len(x)*0.75)
val_size = int(len(x)*0.05)
x_train, y_train = x[:size], y[:size]
x_val, y_val = x[size:size + val_size], y[size:size+val_size]
x_test, y_test = x[size+val_size:], y[size+val_size:]

model = Sequential()
model.add(LSTM(32, return_sequences=False, input_shape=(x_train.shape[1], x_train.shape[2]), name='layer_1'))
#modelGA.add(LSTM(32, return_sequences=False, input_shape=(x_train.shape[1], x_train.shape[2]), name='layer_3'))
model.add(Dense(32,activation='relu'))
model.add(Dense(1, activation='linear', name='layer_2'))
model.compile(loss="mean_squared_error", optimizer='Adam', metrics=["mean_squared_error", r_square_loss])
model.summary()
```

Appendix B**:** Sample Code for Training the Model

```
historyNN = model.fit(x_train, y_train, validation_data=(x_val,y_val), epochs=10)
plt.plot(historyNN.history['loss'])
plt.plot(historyNN.history['val_loss'])
plt.title('Model Loss Plot')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
lossNN = model.evaluate(x_test, y_test)
print('The loss output for test dataset is: %f' % lossNN[0])
predictY = model.predict(x_test)
plt.plot(predictY, label='Predicted CPU Usage')
plt.plot(y_test, label='Actual CPU Usage')
plt.xlabel('time')
plt.ylabel('amplitude')
plt.legend()
plt.show()
def rmse(y_true, y_pred):
    return np.sqrt(np.mean(np.square(y_pred - y_true), axis=-1))
# Get the loss value for each particle
def getFitness(params):
    layer_1 = model.get_layer('layer_1').get_weights()
    layer_2 = model.get_layer('layer_2').get_weights()

    layer1_W1_shape = (layer_1[0].shape[0] * layer_1[0].shape[1])
    layer1_W2_shape = layer1_W1_shape + (layer_1[1].shape[0] * layer_1[1].shape[1])
    layer1_B_shape = layer1_W2_shape + (layer_1[2].shape[0])
    layer2_W_shape = layer1_B_shape + (layer_2[0].shape[0] * layer_2[0].shape[1])
    layer2_B_shape = layer2_W_shape + (layer_2[1].shape[0])
```

```

    layer1_W1 = params[0:layer1_W1_shape].reshape(layer_1[0].shape)
    layer1_W2 = params[layer1_W1_shape:layer1_W2_shape].reshape(layer_1[1].shape)
    layer1_B = params[layer1_W2_shape:layer1_B_shape].reshape(layer_1[2].shape)
    layer2_W = params[layer1_B_shape:layer2_W_shape].reshape(layer_2[0].shape)
    layer2_B = params[layer2_W_shape:layer2_B_shape].reshape(layer_2[1].shape)

    model.get_layer('layer_1').set_weights([layer1_W1, layer1_W2, layer1_B])
    model.get_layer('layer_2').set_weights([layer2_W, layer2_B])

    predY = model.predict(x_train)
    loss = rmse(y_train.reshape(y_train.shape[0]), predY.reshape(y_train.shape[0]))
    return loss
# Computed loss for all particles
def f(params):
    print('Number of particles: %d' % params.shape[0])
    losses = np.array([getFitness(params[i]) for i in range(params.shape[0])])
    print('List of losses for all particles')
    print(losses)
    return losses


### Pyswarm algorithm
options = {'c1': 0.5, 'c2': 0.3, 'w':0.9}

layer_1 = model.get_layer('layer_1').get_weights()
layer_2 = model.get_layer('layer_2').get_weights()
```

```
127 dimensions = (layer_1[0].shape[0] * layer_1[0].shape[1]) + (layer_1[0].shape[0] * layer_1[0].shape[1]) + (layer_1[1].shape[0]
128
129 print("Number of params in Neural Network: %d" % dimensions)
130
131 optimizer = ps.single.GlobalBestPSO(n_particles=20, dimensions=dimensions, options=options)
132
133 cost, pos = optimizer.optimize(f, iters=40)
134 #print(pos)
135 plot_cost_history(cost_history=optimizer.cost_history)
136 plt.show()
```

Appendix C: Creating the Neural Network with the final optimized params using PSO

```
139 ### Creating the Neural Network with the final optimized params using PSO
140
141 layer_1 = model.get_layer('layer_1').get_weights()
142 layer_2 = model.get_layer('layer_2').get_weights()
143
144 layer1_W1_shape = (layer_1[0].shape[0] * layer_1[0].shape[1])
145 layer1_W2_shape = layer1_W1_shape + (layer_1[1].shape[0] * layer_1[1].shape[1])
146 layer1_B_shape = layer1_W2_shape + (layer_1[2].shape[0])
147 layer2_W_shape = layer1_B_shape + (layer_2[0].shape[0] * layer_2[0].shape[1])
148 layer2_B_shape = layer2_W_shape + (layer_2[1].shape[0])
149
150 layer1_W1 = pos[0:layer1_W1_shape].reshape(layer_1[0].shape)
151 layer1_W2 = pos[layer1_W1_shape:layer1_W2_shape].reshape(layer_1[1].shape)
152 layer1_B = pos[layer1_W2_shape:layer1_B_shape].reshape(layer_1[2].shape)
153 layer2_W = pos[layer1_B_shape:layer2_W_shape].reshape(layer_2[0].shape)
154 layer2_B = pos[layer2_W_shape:layer2_B_shape].reshape(layer_2[1].shape)
155
156 model.get_layer('layer_1').set_weights([layer1_W1, layer1_W2, layer1_B])
157 model.get_layer('layer_2').set_weights([layer2_W, layer2_B])
158
159 ### Checking the performance of the LSTM NN using Particle Swarm
160 model.summary()
161
162 predictY = model.predict(x_test)
163
164 time = np.arange(predictY.shape[0])
165 plt.plot(time, predictY, y_test)
166 plt.xlabel('time')
167 plt.ylabel('amplitude')
168 plt.show()
169
170 lossNN_GA = model.evaluate(x_test, y_test)
```

```
171
172 print('The loss output for test dataset is: %f' % lossNN_GA[0])
173
174 print('We see that the loss output for test dataset using Back Propagation is: %f, while the loss output using GA is: %f' % (lossNN[0], lossNN_GA[0]))
175
176 #plt.plot(time, predictY, y_test)
177 plt.plot(time, predictY, label='Predicted CPU Usage')
178 plt.plot(time, y_test, label='Actual CPU Usage')
179 plt.xlabel('time')
180 plt.ylabel('amplitude')
181 plt.legend()
182 plt.show()
```