



International Journal of Cloud Computing

ISSN online: 2043-9997 - ISSN print: 2043-9989

<https://www.inderscience.com/ijcc>

An effective algorithm for predicting load and dynamic task scheduling in cloud fog architecture for smart homes

Krishna Kant Agrawal, Sujeet Kumar, Jitendra Kumar Seth, Abhishek Kumar Gupta, Sonia Lamba

DOI: [10.1504/IJCC.2025.10069250](https://doi.org/10.1504/IJCC.2025.10069250)

Article History:

Received:	14 July 2024
Last revised:	24 November 2024
Accepted:	20 December 2024
Published online:	11 April 2025

An effective algorithm for predicting load and dynamic task scheduling in cloud fog architecture for smart homes

Krishna Kant Agrawal

School of Computer Science and Engineering,
Galgotias University,
Greater Noida, Uttar Pradesh, India
Email: kkagrwal@outlook.com

Sujeet Kumar

GL Bajaj Institute of Management,
Greater Noida, Uttar Pradesh, India
Email: sujeetsingh142000@gmail.com

Jitendra Kumar Seth*

Department of Information Technology,
KIET Group of Institutions,
Ghaziabad, Uttar Pradesh, India
Email: drjkseth@gmail.com
*Corresponding author

Abhishek Kumar Gupta

Department of Information Technology,
Jagannath International Management School,
New Delhi, India
Email: abhishek130685@gmail.com

Sonia Lamba

Department of Computer Science,
ABES Engineering College Ghaziabad,
Uttar Pradesh, India
Email: sonialamba.it@gmail.com

Abstract: The need for smart homes with many devices and services continues to rise quickly. With this surge, smart homes need task scheduling and load-prediction algorithms to provide the proper services for the residents. A deep learning-based dynamic job scheduling and load prediction technique for cloud-fog smart homes is proposed in this paper. This algorithm forecasts task arrival rates at each fog node and assigns them to available fog nodes. It dynamically schedules tasks based on fog node workload. Another option is to

send non-real-time jobs to the cloud and real-time tasks to the fog layer. This optimises load distribution for performance. Using these task assignee models and features, the program optimises prioritised tasks, scores, network latency, and device resource characteristics. We simulate the algorithm's performance in various workloads in this part. The proposed algorithms achieved in higher percentile for 93.79% latency, 95.00% throughput, 95.34% response time, 96.28% scalability, 94.20% fault-tolerance, 97.41% scheduling capacity, 91.41% load balancing capacity, and 95.22% priority management. The results indicate that such an algorithm significantly surpasses the conventional task scheduling methods in load balancing and shortens the average task response time.

Keywords: cloud computing; fog computing; smart homes; task scheduling; metaheuristic algorithms; deep learning.

Reference to this paper should be made as follows: Agrawal, K.K., Kumar, S., Seth, J.K., Gupta, A.K. and Lamba, S. (2025) 'An effective algorithm for predicting load and dynamic task scheduling in cloud fog architecture for smart homes', *Int. J. Cloud Computing*, Vol. 14, No. 1, pp.54–85.

Biographical notes: Krishna Kant Agrawal is currently working as a Professor in CSE, Program Chair (PG Programs) and Research Group Head at Galgotias University, Greater Noida. He has completed his PhD from the Indian Institute of Information Technology, Allahabad (IIIT-A). He received his degree of MTech in Computer Science and Engineering from Birla Institute of Technology, Mesra, Ranchi. He is the recipient of a US Patent, Indian Patent Grant, German Patent Grant, South African Patent Grant and many other patents filled, and design patents granted in his name. He has more than 13 years of research and teaching experience across multiple institutions and has published more than 52 research papers in reputed international refereed journals/SCI, Scopus Indexed, conferences proceedings and book chapters. He is also a member of International Professional bodies like IEEE, IET-UK, CSTA-(ACM, USA), and ACM Professional Member.

Sujeet Kumar is currently an Assistant Professor of the Department of Computer Application at GL Bajaj Institute of Management, Greater Noida (India). He has completed his Master's degree from Gautam Buddha University, Greater Noida (India). A seasoned academician having more than 1.5 years of experience, he has published more than 15 scientific articles, including journal papers, book chapters and conference papers. He organised several conferences/workshops/seminars at the national and international levels. He voluntarily served as a reviewer for various international journals and conferences.

Jitendra Kumar Seth is working as a Professor in the KIET Group of Institutions Ghaziabad. He completed his PhD degree in CSE from Jaypee Institute of Information Technology, Noida, in 2019. He obtained his MTech degree from Shobhit University Meerut in 2009 and a BTech from Radha Govind Engineering College Meerut in 2004. During his teaching career of more than 20 years, he has taught courses such as Cloud Computing, Big Data, Distributed Systems, OOP, Web Technology, and Principles of Programming Languages. He has participated in many national and international conferences and has published more than 24 research papers in reputed international journals and conferences. He has guided more than 60 graduate projects and supervised PhD scholars. He has expertise in various programming languages like Java, Python, PHP and XML. His research area includes cloud computing, big data, cyber security, machine learning, deep learning and search engines.

Abhishek Kumar Gupta is currently serving as an Assistant Professor in the Department of Information Technology at Jagannath International Management School, New Delhi. He completed his Master's degree from Uttar Pradesh Technical University, Lucknow (India) and was awarded a Doctorate degree by Socrates Social Research University, Delhi, in 2023. With over 13 years of academic experience, he is a seasoned educator and researcher. He has published seven research papers in esteemed international conferences and journals, showcasing his expertise and contributions to the field of information technology.

Sonia Lamba is presently working as an Assistant Professor in the Department of Computer Science at ABES Engineering College, Ghaziabad. She is pursuing a PhD from Jaypee Institute of Information Technology (JIIT), Noida and has received an MTech (CSE) degree from UCER, Allahabad. She has published a book/book chapter in several research papers in Scopus/international journal and conferences of repute. She had published/grant many Indian patents. She completed a number of certified MOOC courses offered by Coursera/NPTEL. She is an active member of the IEEE Professional Society. Her research interests are in machine learning, drug repositioning, cloud, healthcare and computer networks. She has 8+ years of teaching experience. She has organised many events in the college related to education activities (conferences, FDP, etc.).

1 Introduction

The home users connected devices to control and monitor systems and appliances such as lighting, heating, ventilation, air conditioning, security, and entertainment. Homeowners can control these devices from a Smartphone, tablet, or computer, no matter where they may be located (Ramezani Shahidani et al., 2023). Among the technologies fundamental to smart homes is the cloud-fog hybrid architecture (Zanardo, 2024). Definition: cloud fog should be a hybrid infrastructure of cloud computing and fog computing capabilities (Talaat et al., 2022). A cloud is a remote server and network that stores information or data and performs more and more work. Fog computing is the technology that pushes computing and storage resources to the edge of the network (Sharma et al., 2023). Fog computing is critical in the context of smart homes, where computing resources are located in the house to process the data produced by sensors and control devices. On the other hand, cloud computing is used for storage and management functions like data analysis and prediction (Wang et al., 2024). There are many advantages to using cloud fog architecture in smart homes (Thirunagalingam, 2024a). Fog computing also enables real-time data processing by moving computation resources close to the devices and sensors so that the data can be processed near the actual collection point instead of being sent off to the cloud for processing (Salehnia et al., 2024). This is particularly beneficial for time-critical applications like security systems, which need an immediate response (Sayeddelahl and Farouk, 2024).

Scalability is the leading business advantage of cloud fog architecture. Smart homes could react immediately to changing uses and requests by utilising the resources of cloud and fog computing (Abohamama et al., 2022). This minimises network traffic-congestion-clearing resource data, which does not need to be sent to the cloud. Shareholder-homes, the amount of data that needs to be stored and transmitted to the

cloud (Nagarajan et al., 2021), also cost savings for homeowners. Chatbots have flourished in the last few years due to technological advancements and the rapidly growing popularity of the internet of things (IoT). We design homes with many tools and systems controlled by a central hub/interface (Saxena et al., 2023). Adding to this technology that has been introduced to enhance the functioning and efficiency of smart homes is the home fog cloud (Salehnia et al., 2024). However, this architecture poses several technical challenges that must be overcome to work correctly. The excessive dependence of intelligent Home and cloud fog architecture on cloud servers is one of the leading technical issues (Thirunagalingam, 2024b).

The cloud servers host the data collected from various sensors and devices in your home and process it in the cloud (Soji et al., 2024). This might also cause latency issues due to data required to transmit to and from the cloud server, which in turn will affect the real-time response of the home automation system (Chhilar, 2023). Relying on the cloud server leaves the smart home open to network outages or server downtime, resulting in the system being out of action (Abinavkrishnaa et al., 2023). Another concern is the potential threats to privacy and security data storage in the cloud poses. Smart homes have been hacked many times, and cyber attacks are only growing (Anand et al., 2024). This data includes personal information and behavioural patterns, but if it ends up in the wrong hands, this information can be exploited. Further compatibility can introduce trouble for integrating diverse devices and systems from different vendors (Memari et al., 2022). This leads to the loss of data being transferred to another device, which can only be processed in the right way if the devices respond correctly to each other. Finally, smart homes only work if you have a stable internet connection and high-speed service; if you experience connection loss, the smart home will be somehow affected (Minu et al., 2023). While the cloud fog architecture provides many advantages for smart homes, countless technical challenges must be thoroughly addressed for successful realisation (Cirillo et al., 2023). It helps to successfully implement Smart Home environment task scheduling and load balancing algorithms (Jasper et al., 2024). Dynamic task scheduling and load prediction algorithms are essential to optimise energy consumption in a smart home with cloud and fog architecture for sustainable homes (Nagaraj and Subhashni, 2023). These are done by using deep learning algorithms, which in turn automate the entire process of scheduling and predicting (Rajasekaran et al., 2023). These algorithms can broadly be classified into three types as follows:

- *Supervised learning*: The model is trained using a labelled dataset to predict future tasks and load based on historical data. These algorithms use tech and neural networks that process patterns and predictions accurately.
- *Unsupervised learning*: It enables models to create patterns or predictions based on clustering and dimensionality reduction without unlabeled training or anomaly detection. These algorithms are helpful in addressing problems where there is no defined solution and would require discovering hidden patterns in the data.
- *Reinforcement learning*: In this approach, an agent learns through interactions with its environment to maximise a reward. An agent interacts with its environment by taking action, and as it receives feedback from the environment, it learns to make better decisions in the future. Since this implementation realises quickly the adaptation to surroundings continuously, it is in a position to optimise energy consumption immediately.

They can also be classified according to their architectures and implementations. In a centralised approach, all the data is processed and stored in a central location, like the cloud. In contrast, a decentralised approach distributes processing and storage over multiple nodes, like fog nodes. There are also hybrid approaches that use both central and distributed processing. The contribution of the proposed model is as follows:

- *Minimising response time:* One technical objective is to reduce the time taken to respond to each task as to the corresponding suitable and available Resource. This can drastically increase system responsiveness and user experience.
- *Balance workload:* The task scheduler tries to load balance so none of the endpoints have an unnecessarily high load imposed on them. This approach will also prevent potential bottlenecks and efficiently use all resources.
- *Maximise throughput:* Task scheduling and load balancing also help achieve maximum throughput, which means getting as much work done as possible within a specified time duration. Organisations can execute that by delegating assignments to the available resources.
- *Fault-tolerance:* Load balancing also involves integrating failover measures for the system to keep running when a resource fails or becomes unavailable.
- *Scalability:* The ability to scale the system in response to an increased workload or additional resources. This needs reliable task scheduling to keep up with a growing number of tasks and continue balancing the load.

2 Related words

Ifitikhar et al. (2023) have discussed a sophisticated task-scheduling system using Artificial Intelligence algorithms to support energy-efficient computing in cloud-fog computing environments. It dynamically maps and schedules tasks to the most capable resources while considering energy, processor, and network constraints, leading to better performance and lower energy usage. Alizadeh Javaheri et al. (2024) have discussed the architecture and show the use of reinforcement learning techniques to allocate resources in a cloud. We use deep learning techniques to train a deep neural network to autonomously decide when and how to distribute resources to be maximally beneficial under the current and future workload. This method will increase the utilisation rate of resources and achieve better performance and cost-effectiveness of cloud computing services. Nethaji and Chidambaram (2021) Proposed resource optimisation in fog computing is used to efficiently use computing, storage, and networking resources in a fog environment. To improve the optimisation, the shift-invariant deep convolutive load balancing method is applied to dynamically balance the workload among fog nodes to reduce the latency and improve the performance; this enhances the efficiency and cost-effectiveness of fog computing systems. Fahimullah et al. (2024) proposed a machine learning-based solution for resource management in fog computing. They make use of algorithms and statistical models to predict and optimise resource allocation in a dynamic and heterogeneous setting. These solutions can aggregate data from tens or hundreds of sensors and actuators, adding intelligence to their decision-making and improving the efficiency and performance of fog computing systems. Yu et al. (2022)

have discussed an efficient intelligent grid load balancing via fog and cloud computing: A rough sets-based approach. The utilisation of fog (edge computing) and cloud (centralised computing) to distribute electricity demand across different energy resources. It will optimise energy use, lower costs, and increase the grid's reliability, making the smart grid more efficient and sustainable (Renjith, 2024).

Umair et al. (2023) have discussed energy management in smart homes using fog-based IoT architecture. So, this architecture will employ fog computing to foster an energy-aware environment, most notably in the house energy management scenario. It provides real-time data analysis and decision generation that helps with the precise use of energy and overall cost reduction. Taking artificial intelligence approaches to make Offloading decisions is discussed by Abdulazeez and Askar (2023), where offloading mechanisms rely on reinforcement learning and deep learning algorithms to apply AI algorithms that make intelligent Offloading decisions in a fog computing environment. These algorithms have a self-learning mechanism and adjust themselves automatically to changes in network conditions to deliver best-in-class performance and resource utilisation. The segmented random search-based batch scheduling algorithm has been discussed as a novel method for task scheduling in fog computing, as depicted in Figure 1 (Hasan et al., 2024). To improve the overall system performance and the utilisation of resources, random search and batch scheduling are proposed to be used in the distributed fog environment to carry out the most effective resource allocation and task scheduling. Shadroo et al. (2021), in two-phase scheduling for the IoT using deep learning has been discussed. This is the deep learning approach that optimises task and resource allocation. The models are trained in the initial phase to predict resource availability using a deep learning approach. In the next phase, the expected availability of the tasks determines scheduling for efficient resource use in the IoT environment. Raju and Mothku (2023) presented a delay- and energy-aware task scheduling using Reinforcement learning for optimising task scheduling in fog-enabled IoT applications. This paper optimises function allocation to the fog nodes in terms of delay, analysing the impact on delay and energy expenditure to achieve high performance and energy efficiency.

Using a decentralised prediction method, the design was generated by Shamsa et al. (2024). This method of predicting and load distribution across cloud, fog, and IoT devices helps balance the load efficiently. All aspects considered, these results in less congestion in the network, better utilisation of resources, and scalability to improve the sensitivity level of the system to changes in environmental conditions. Pal et al. (2023) described the proposed intelligent task scheduling model (detailing the scheduling of IoT and cloud) used for effective task reallocation of big data applications. It considers data size, network bandwidth, and processing capability to utilise resources better and perform at the best suit. The Mayfly-Taylor approach (Shruthi et al., 2022), which applies deep reinforcement learning for dynamic task scheduling in a fog-cloud environment, was reviewed. It balances the load between computing nodes to improve resource utilisation and reduce task latencies. It also integrates real-time data to further its ability to make essential decisions more effectively regularly (Karthik and Kavithamani, 2021). Recent work showed that a Fog computing-based deep learning model continuously seeks computing resources at the network's edge to improve the operational efficiency and flexibility of a WSN interfaced with a microgrid (Pal et al., 2023). These principles span from load-balancing to help distribute the load evenly across different system parts to anti-corruption layers that help maintain the separation. The technology is based on advanced deep learning constructs that help to process and adapt the system for the best

possible performance. We present the nature-inspired load balancing (NILB) framework (Nayak et al., 2024) designed to enhance resource allocation and performance in fog computing environments by borrowing the load balancing mechanism in natural systems like ant colonies and bee swarms. The discussed behaviour of this framework, intelligent load balancing across fog and cloud, leads to more optimum resource use similar to those of these systems. Table 1 shows the comprehensive analysis.

Table 1 Comprehensive analysis

<i>Authors</i>	<i>Advantage</i>	<i>Limitation</i>
Iftikhar et al (2023)	Optimises energy consumption and resource allocation, leading to cost savings and improved system performance.	Some potential limitations of HunterPlus may include limited support for diverse and complex computing environments, as well as higher costs for implementing AI-based solutions.
Alizadeh Javaheri et al (2024)	Higher efficiency and reduced human intervention in managing resources, resulting in cost savings and faster delivery of services.	Limited ability to adapt to changing environments and handle unforeseen scenarios due to fixed training on historical data.
Nethaji and Chidambaram (2021)	Effective utilisation of edge devices and balanced distribution of computing tasks leads to improved system performance and user experience.	Limited applicability to systems with highly unpredictable or dynamic workloads.
Fahimullah et al. (2024)	Machine learning can adapt and optimise resource allocation based on dynamic environmental factors, leading to more efficient and effective use of resources.	One limitation is the potential for biased decision-making due to the reliance on historical data and pre-defined algorithms.
Yu et al. (2022)	Efficient smart grid load balancing via fog and cloud computing allows for optimal utilisation of resources, leading to improved energy efficiency and reduced costs.	The dependency on stable internet connectivity for data transmission can affect the performance and efficiency of the system.
Umair et al. (2023)	Energy management in smart homes provides real-time control and optimisation of energy usage, resulting in increased efficiency, cost savings, and reduced environmental impact.	High dependency on stable internet connection and power supply, making it vulnerable to disruptions and outages.
Abdulazeez and Askar (2023)	Offloading mechanisms using RL and DL algorithms optimise resource allocation and improve performance, scalability, and energy efficiency in heterogeneous fog computing environments.	The reliance on pre-determined reward functions and trained models may not fully capture the dynamic and unpredictable nature of fog environments.
Hasan et al. (2024)	Efficient utilisation of resources due to selective random search helps optimise the allocation of tasks to fog nodes.	The algorithm does not consider deadline constraints for tasks and only prioritises energy efficiency.

Table 1 Comprehensive analysis (continued)

<i>Authors</i>	<i>Advantage</i>	<i>Limitation</i>
Shadroo et al. (2021)	Effective resource allocation and better network management due to advanced decision-making and adaptive scheduling algorithms based on deep learning.	Unpredictable network conditions and device failures may result in inaccurate predictions and cause scheduling inefficiencies.
Raju and Mothku (2023)	Optimises task execution time and energy consumption for better overall system performance.	The performance of the reinforcement learning algorithm is highly dependent on the quality of the initial policy.
Shamsa et al. (2024)	Improved system scalability and adaptability, as each device can make autonomous predictions based on local information without centralised control.	Improved system scalability and adaptability, as each device can make autonomous predictions based on local information without centralised control.
Pal et al (2023)	Improving resource allocation and reducing latency, leading to more efficient and timely processing of large amounts of data.	Scheduling may be affected by network failures or delays, leading to inaccurate or suboptimal task allocation and execution.
Shruthi et al. (2022)	Adapts to changing workloads and resource availability, providing efficient and accurate scheduling for fog-cloud computing environments.	Limited ability to handle sudden changes in workload and resource availability due to its reliance on historical data.
Karthik and Kavithamani (2021)	Fog computing-based deep learning allows for real-time decision-making and adaptability, enhancing load balancing and efficiency in microgrid-connected WSNs.	The reliance on stable and reliable network connectivity can hinder the performance of the model in remote or low-connectivity areas.
Nayak et al (2024)	Efficient resource utilisation results in improved performance and cost savings.	The study does not consider the impact of varying network conditions or sudden failures on the load-balancing algorithm.

2.1 Research motivation

- *Scalability:* The cloud and fog infrastructure is growing rapidly; scheduling tasks and load balancing must be able to scale accordingly. This includes task management in the presence of multiple fog nodes, load variability, and changes in the network environment.
- *Communication overhead:* Tasks in fog environments may run across various nodes, which are distributed geographically; this results in high communication overhead. This will affect the performance and efficiency of the system and needs to be considered when scheduling tasks.
- *Heterogeneity:* The fog environment comprises heterogeneous devices and resources, such as sensors, mobile devices, and cloud servers. It must provide a scheduling system to delegate tasks depending on the capabilities, constraints, and connectivity of devices.

- *Task prioritisation:* In a cloud-based fog environment, there can be a blend of different tasks, each with a different priority. The scheduling and load-balancing mechanisms must be aware of these priorities and allocate resources accordingly to ensure that high-priority tasks are completed promptly.
- *Fault tolerance:* When a node goes down or a network partition happens, fault tolerance is essential for task scheduling and load balancing. It fixes the issue by redirecting requests to other live nodes and preventing the load balancer, ensuring data consistency and QoS replication.

2.2 Novelty of the proposed algorithm

This paper proposes a deep learning-based algorithm that continuously monitors and evaluates the task load for tasks within smart home environments based on local and external contextual information. This allows for effective scheduling and dispatching of tasks among nodes such as cloud and fog in order to optimise performance and energy consumption. The algorithm intelligently learns over time, taking into account various changing parameters like occupants coming and going along with external weather conditions, which makes sure that the tasks are executed at low overhead in an efficient and optimum manner and on time. This supports embedded systems, smart home devices, and cloud/fog resources to create a reactive and energy-efficient environment for users. This innovative method in computer science uses deep learning in task scheduling and load-prediction algorithms. This unique design leverages a neural network to reallocate resources and actively balance workloads in real. The proposed algorithm continuously learns and adapts to current conditions, distributing tasks to various nodes with minimal overhead. Not only does this lead to better state-level performance, but it also gives you an extreme level of scalability. Old quantified task scheduling and load balancing operations are rule garden and heuristic driven, which can also end up with some suboptimal results. Using deep learning, this algorithm can efficiently utilise resources and increase efficiency in various complex distributed systems. This technical novelty can have a revolutionary impact on managing task scheduling and load balancing in modern computer systems.

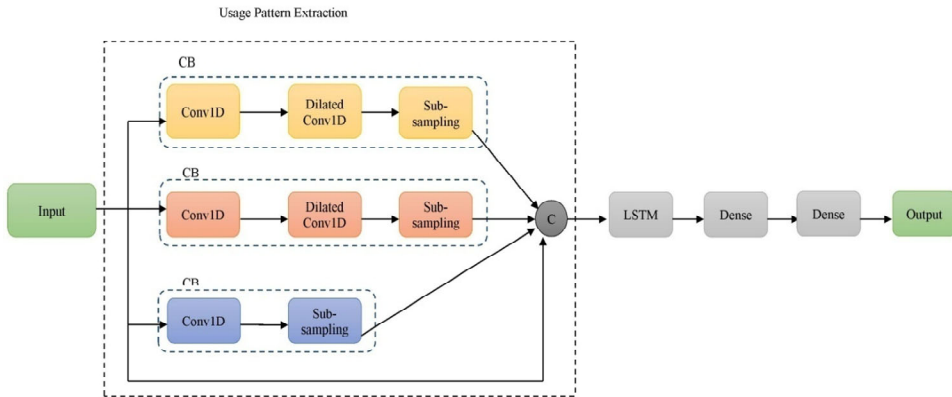
3 Proposed model

The proposed algorithm targets task scheduling and load predicting in smart homes. The system will calculate the load in the house, and given this data, it will more efficiently predict and distribute resources. Cloud fog architecture helps distribute the data processing and task execution between the local end device and the cloud, which will increase execution performance and substantially reduce overall execution latency. Another algorithm that accounts for power consumption and resource feasibility is that the system can run efficiently and sustainably. It is used to improve task execution and user experience in smart homes.

3.1 Construction

The conv1D layer is a convolution network used in deep learning models to analyse sequential data. This is done by creating a filter on the input data that moves across the sequence and computes the dot product at each step. This makes it a layer of attraction to extract data patterns and features. The filter parameters are discovered during training; hence, they can differ completely for each data set. The Conv1D layer then takes the result and outputs another sequence of feature maps, which another layer can use. This introduces adjustable depth of the feature maps, making it possible to extract more differentiating features as required around the last layer united with the previous layer. Conv1D is an essential operation in deep learning and helps extract such information from sequential data, leading to good performance on this model type. A Dilated Conv1D layer is a convolutional layer used in deep learning. This is a one-dimensional layer used to analyse time series or 1D data, such as audio signals. The Dilated Conv1D operation works similarly to the joint convolution operation. Applying a filter kernel to the input data, computing a dot product for each step. In this case, a series of values in the kernel will be spaced much further apart, which causes the receptive field to grow even more. This layer can learn information from a more extensive input signal range. This more significant expansion of the dilation rate allows the layer to increase its receptive field without incurring additional parameters and thus fully use its more parameters. This makes it worthwhile to use sequence data, as it can learn long-term dependencies. Furthermore, this layer can be stacked, and an input to the CNN on top of another CNN will have a broad receptive field, which enables it to learn more accurate long sequence patterns, making it a key component for modelling complex sequential data. So, as per the above step 1, the construction of the proposed model is shown in Figure 1.

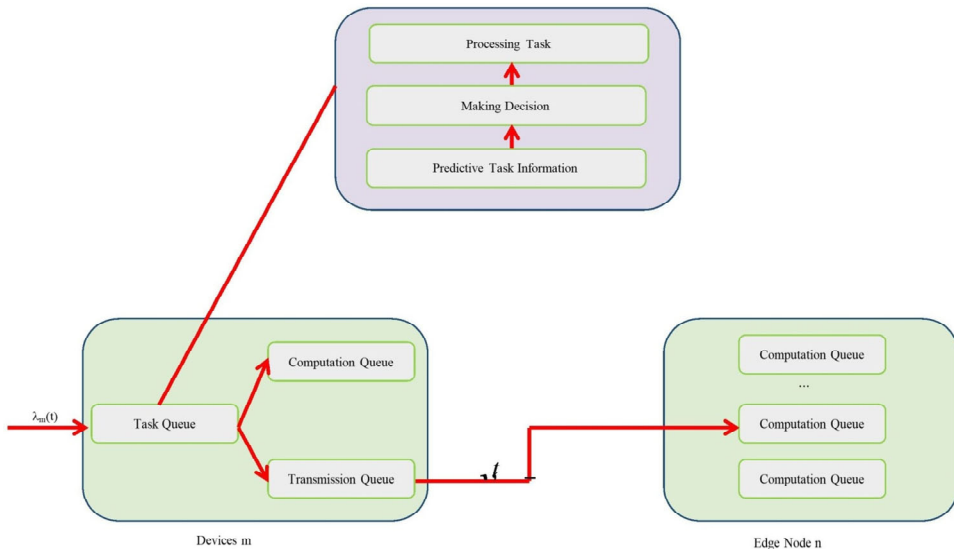
Figure 1 Construction of proposed model (see online version for colours)



Sub-sampling is also a crucial operation in signal processing and data compression. It essentially decreases the resolution or the size of a signal or a data set without losing its basic information. This is done by taking a subset of the original data and dropping the rest. The two most common types of sub-sampling used today are downsampling and Decimation. They sample the signal's sample rate by selecting samples at a regular interval and decimating samples from every n th sample. Both perform dimensionality reduction on a signal or a data set to compress its size while retaining most of its

characteristics. It is helpful for applications where the original data is too big to be processed efficiently or when a lower resolution is acceptable. It is crucial in many fields, such as image and video compression, audio processing, and data storage. LSTM cell has the input gate, the forget gate, and the output gate, which regulate the flow of information. The input gate governs the input, the forget gate governs data discarding, and the output gate determines the upcoming new states to be shown. They make the LSTM cell well-suited to learning and predicting one step in time series forecasting tasks, such as speech-to-text, language translation, and sentiment analysis. Dense layers in deep learning models are a final approach for producing a desired output. This is a thick layer; all neurons are connected to every neuron in the prior layer. A layer gets data as input and outputs a tensor of new data. The weights and biases represent the learned aspects of the model, and the activation function introduces nonlinearities. This transforming process enables the model to understand nonlinear interaction and predict the data originating from hidden distributions. The dense layer is responsible for translating the input data to the output point with some mathematical functions.

Figure 2 Task scheduling in fog computing (see online version for colours)



3.2 Task scheduling in fog computing layer

Eventual Task scheduling can process on-demand function processing on the fog computing layer. So, it is about putting a specific input in and getting an expected output out, with all the in-between steps involved. It usually starts with data collection and preliminary organisation, followed by the analysis and manipulation of data by implementing various algorithms and techniques. This can range from sorting, filtering, and calculations to other configurations happening on data. Resultantly, it generates and saves output for further use or analysis. The accuracy of the input data, the methods and algorithms used for processing, and the scheduling in the fog computing layer all determine the outcome of a processing task. Decision-making is an important activity associated with much thought (comparing and contrasting) and selecting the best from

different alternatives available. Making a decision often follows several steps that are taken in the sequence of identifying the problem, gathering information, identifying the other options, and evaluating the alternatives to choose and the best solution. There can be personal hang-ups, emotions can get in the way, and external factors may discourage the outcome. Making decisions, on the other hand, increases or decreases in complexity subject to the fact that it can be rational or unconscious. Decision-making includes running through the cascade of results each option will bring and choosing an option that aligns with your objectives and values. This is complicated; it involves cognitive rational thinking and cognitive or non-cognitive intuitive thinking. Figure 2 shows the task scheduling.

Predictive task information (PTI) is a data analysis, interpretation, and forecasting effort to capture the predicted output of a task or project. These can be economic indicators, e.g., historical data, market trends, or other expert's views. This step lists potential risks/threats or challenges that might lead to task failure. This is done using specialised tools and techniques – data mining, statistical modelling, and machine learning algorithms. Such analysis results will assist stakeholders in making wise decisions to follow risk management techniques and to succeed in the chosen project. PTI is constantly being updated with new data, which keeps one vigilant against potential changes. Rep. Task scheduling algorithm is shown below.

Algorithm 1: Task scheduling for fog computing layer

```

INPUT: Tasks (t1,t2,...,tn);
Output: Scheduling Tasks (ST);
1  START
2  Initilize_Task_Queue (TQ);
3  if TQ == n
4  CALC_Priority (P);
5  tn = extract (candidate, root);
6  Execution (E) = Comm.Root + Avl.candidate;
7  Schedule_task (tn);
8  else If TQ == 0;
9  Then CP (Candidate Pool) assigned to root;
10 Display "Empty Queue";
11 END

```

According to the algorithm, the input is a list of tasks (t1, t2,..., tn), and the task is to output a scheduling of these tasks (ST). Initially, initialise a task queue (TQ) and check if the number of tasks in the queue equals the total number of tasks (n). If the queue is not empty, the algorithm calculates the priority of each task (P) and extracts the candidate task (tn) from the root of the queue. Then, it decides to execute this task, considering the communication with the root and its candidate availability. The task is Scheduled and dequeued. Thus, the algorithm assigns CP as the root and prints an empty queue message if the queue is not empty -meaning some tasks are waiting to be scheduled. After all the tasks have been allocated, the algorithm is terminated. In short, the algorithm schedules tasks more expedited, considering the priority, execution, and inlet availability of the

tasks. This prevents an empty TQ so all tasks are still correctly first specified and appended for execution.

$$ET(z_j) = \sum_{q_{ij} \in z_j} ExtTime(q_{ij}) = \frac{\sum_{q_{ij} \in z_j} length(z_j)}{CPU(z_j)} \quad (1)$$

$$ExtTime(q_{ij}) = \frac{length(z_j)}{CPU(z_j)} \quad (2)$$

where Z is the task, and j is the slot number, and ET is the execution time. The task's duration is expressed in terms of the amount of instructions. And the rate that is utilised to run each cloud virtual machine (VM). Make span, which is the highest value of each VM's execution duration, can be computed using

$$Makespan = Max(ExtTime(z_j)), 1 \leq j \leq x \quad (3)$$

The lower bound of the makespan, or the shortest amount of time the system needs to finish all jobs, is called the min makespan. Min Make span is determined via

$$MinMakespan = Min(ExtTime(z_j)), 1 \leq j \leq x \quad (4)$$

Fitness function in terms of makespan D_1 can be calculated by,

$$D_1 = \frac{MinMakespan}{Makespan} \quad (5)$$

The cost of requesting to be processed by the job is the second objective, and it may be computed using the costs associated with CPU, memory, and bandwidth utilisation. When processed, the estimated cost can be computed using

$$Cost(q_{ij}) = (b_1 * ExtTime(q_{ij})) + (b_2 * ExtTime(q_{ij})) + (b_3 * ExtTime(q_{ij})) \quad (6)$$

where are the costs associated with CPU, memory, and bandwidth utilisation per unit in vi , respectively. The total cost() can be computed as the total of all jobs that are processed on all VMs.

$$QCost = \sum_{i=1}^y \sum_{j=1}^x Cost(q_{ij}) \quad (7)$$

$$\begin{aligned} MinQCost &= \sum_{q_i \in Q} MinCost(q_i) \\ &= \sum_{q_{ij} \in Q} \min_{1 \leq j \leq x} (Cost(q_{ij})) \end{aligned} \quad (8)$$

$$D_2 = \frac{MinQCost}{QCost} \quad (9)$$

If the task is dispatched as requested, we can determine how much memory is used,

$$RX_j = AX_j + \frac{LX_i}{QX_j} \quad (10)$$

$$RB_j = AB_j + \frac{LB_i}{QB_j} \quad (11)$$

where AX_j is the amount of memory usage before executing the task z_j ; AB_j is the amount of CPU usage before executing task z_j at the j^{th} in VM.

VM utilisation evaluation (ZW_j) can be calculated by,

$$D_3 = ZW_j = \frac{\omega_1}{1 - RX_j} * \frac{\omega_2}{1 - RB_j} \quad (12)$$

where are the corresponding weights = 1 in this research.

$$RG_s = \sum_{j=0}^{x_s} ZW_{sj} \quad (13)$$

$$AR = \frac{\sum_{s=0}^e RG_s}{e} \quad (14)$$

$$D_3 = \sum_{s=0}^e |RG_s - AR| \quad (15)$$

where RB_j is the amount of memory usage after executing task z_j ; LX_i is the amount of CPU usage after executing task X_i at the j^{th} VM; QB_j is the amount of disk space used by task z_j at the j^{th} VM. The weighted average of every fitness function is computed to form the fitness function. In, the suggested fitness function (D) is displayed.

$$D = (\gamma_1 * D_1) + (\gamma_2 * D_2) + (\gamma_3 * D_3) \quad (16)$$

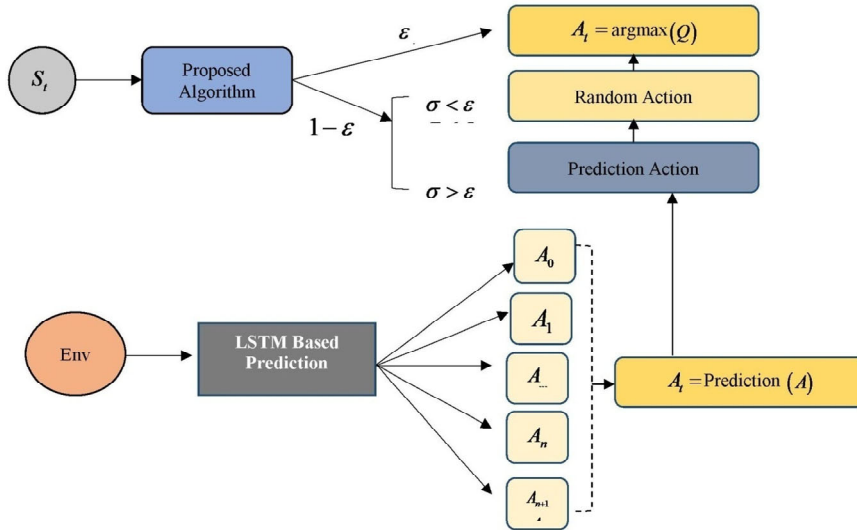
where the coefficients balance makespan, total cost, and resource usage, a better solution is obtained by maximising the utility function (G).

$$Q(k_q, a_q) = (1 - \alpha)Q(k_q, a_q) + \alpha[l_q + \gamma \max_{a+1} Q(k_{q+1}, a_{q+1})] \quad (17)$$

A TQ is a solution for distributed systems to schedule and receive tasks in an orderly manner. Its purpose is to optimise the flow of work, learn tasks in a queue, and be processed in a fashion. Once a process or application wants a task to be performed, the queue chooses the next available task and gives it to a worker. It guarantees that operations are conducted in an organised and orderly manner. A prevalent error handling and monitoring feature set is usually part of TQs to keep the system running smoothly. These things combined so bottlenecks are averted, and the electricity goes great where it wants to move. Queue to hold and manage computations to be processed. It operates on the first-in-first-out (FIFO) principle, which means the task added to the queue will be executed first. When added to the queue, a computation is given a priority score based on properties like complexity and urgency. The tasks/operations that constitute queue e are sorted based on this score for scheduling and resource allocation. After finishing a calculation, the next in the line is executed. It guarantees that computations are done smoothly, making it a good choice for executing complex laxores subject to evangelistic

kick-off dates. The Transmission queue is one of the essential parts of communication during messaging. The queue is a transient buffer where the outgoing messages that must be sent to other systems or applications sit. A transmission queue moves a message from a format queue to a queue on a matched machine. When that message achieves the queue, it gets a novel Identifier and is put in a FIFO design. That ensures in-order delivery of messages. It also recovers the messages sent and received using NOPING. It also handles errors during the transmittal process and reliably delivers messages. Transmission queue is essential in data movement between systems with minimum human intervention.

Figure 3 LSTM-based load prediction (see online version for colours)



3.3 LSTM based load-prediction

The model utilises special memory cells, LSTM cells, to learn long-term dependencies in the data. So, it can selectively remember some cell input for previous time steps, thus having cells that can learn deep patterns and relationships in the data. The model receives a sequence of input data, sequences it through multiple layers of LSTM cells, which can operate as a series of temporal dependency layers, and, slightly after the sequence, lands a single prediction for the following data point. While training, the model learns to adjust its parameters such that the differences between the accurate predictions and its predictions are minimised, with the goal of making sure those predictions are valid for future unseen data. In computer programming, random action is a process in which a specific action is decided and chosen randomly without front knowledge. This is usually done in algorithms and programs that need some randomisation, for example, in AI or simulation programs. Figure 3 illustrates this for the LSTM-based load prediction.

For random act, which means the call as a random number generator, this generates a sequence of numbers. This sequence is then utilised to determine which, among a group of possible actions, was the action taken. What is mind-blowing about this is that every time it executes the random action, it will reach different results, making our program more unpredictable. Load prediction algorithm is shown below:

Algorithm 2: Load prediction

```

1  START
2  INITIALIZE_Input_Loads (l1,l2,...,ln)
3  Compute_Network.Load (NL);
4  if (P.ct == Available)
5  then combine priority loads;
6  P.pl = P.pl + load.ip;
7  UPDATE P.pl;
8  Else
9  then combine all loads;
10 NL = P.pl + p.npl + ip;
11 UPDATE NL;
12 return P.pl, P.nl;
13 END

```

The START algorithm is used for computing $l_1, l_2, \dots, l_n$, starting with the initialisation of loads loaded. 1. It then calculates the network load (NL) by calling the LOAD function. It will check for the available priority loads ($P.ct == Available$) and combine those priority loads by adding the new load (IP) with the existing priority load (P.pl). Then, the algorithm will modify the priority load (P.pl) with the new joint load. Otherwise, with no priority loads left, the algorithm would merge all the loads and add a new load (IP) with the current NL. Then, the algorithm will update the NL with the sum of loads. The algorithm ends with returning the P.pl and NL. A portable algorithm for the management and scheduling of various loads throughout a network, the algorithm may be configured to ensure that high-priority loads are treated preferentially. The proposed algorithm intends to find the optimal solution here. It is possible to calculate the Q-value using the following formula,

$$Q_{q+1}(k_q, a_q) = Q(k_q, a_q) + \alpha \left[l_1 + \gamma \max_{a'_q} \left[Q_q(\delta(k_q, a_q), a'_q) \right] - Q_q(k_q, a_q) \right] \quad (18)$$

$$m_{j,i}^0 = m_i^{\min} + rand(0,1) * (m_i^{\max} - m_i^{\min}) \quad (19)$$

Using, the employed bees locate the neighbouring source of the current source during the load phase.

$$z_{j,i}^q = m_{j,i}^q + rand[-1,1] * (m_{j,i}^q - m_{s,i}^q) \quad (20)$$

$$z_{j,i}^q = m_{j,s}^q + \phi_{j,i}^q \cdot l_i^q \cdot (m_{j,s}^q - m_{RFS,s}^q) \quad (21)$$

$$\sum_{z=1}^0 VM_{pr} = \sum_{z=1}^0 (VM_{mips} + VM_{cpu} + VM_{mem}) \quad (22)$$

$$GDV_{z,t} = \frac{\sum_{z=1}^m V_{leng}}{\sum_{t=1}^o TN_{pr}} \quad (23)$$

The time it takes to allocate job w to task v is known as the task allocation time, and it is based on the task file size in relation to the VM's bandwidth, as shown in

$$VJ_{z,t} = V_{pq} / TN_{iz} \quad (24)$$

$$DV_{z,v} = GDV_{z,v} + VJ_{z,t} + CT_{z,t} \quad (25)$$

$$CT_{z,t} = \begin{cases} 1, & \text{if } v_z \text{ is allocated to } TN_t \\ 0, & \text{Otherwise} \end{cases} \quad (26)$$

The total number of tasks and task duration that the VM currently has relative to its capacity determines the load of the VM.

$$TN_{capa} = TN_{prv} + TN_{iz} \quad (27)$$

$$TN_{bc} = \frac{V_z \times V_{leng} \times V_{pq}}{TN_{capa}} \quad (28)$$

$$TM_{ild} = \sum_{t=1}^o TN_{id} \quad (29)$$

Every task in this action area has its VM to run it on. There are several bits of information associated with each activity, including the quantity, duration, and size of the files. We can refer to our action space as follows:

$$J = \{V_z, V_{leng}, V_{pq}\} \quad (30)$$

The reward function in our approach is defined by minimising both the task completion time on a particular VM and energy usage. The prize is shown in

$$l_v = \{\min DV_{z,t} + G_{cons}\} \quad (31)$$

Q-networks' parameters can be updated by reducing the loss function. The loss function may be built using equation (22), and the target state-action value function can be established using equation (23), respectively.

$$R(\theta) = G \left[\left(target_v - S(q_v, j_v; \theta) \right)^2 \right] \quad (32)$$

$$target_v = l_v + \gamma_{j_{v+1}}^{\max} S'(q_{v+1}, j_{v+1}; \theta') \quad (33)$$

$$Q_o = G_q / G_f \quad (34)$$

Prediction action is a process by which we predict future outcomes with the help of statistical models and data analysis techniques. This is the disaggregation of vast amounts of data, interpreting the emerging patterns and trends, and using this data to project future developments. The problem was to define the objective of prediction. The first thing to do is to separate the power of the bandwagon. This is accompanied by data collection, cleaning, and preparation. These datasets are subjected to statistical models that can make

predictions based on this data. The point is that these predictions can be operationalised to make a better decision and outcome. Thus, the correctness of the predictions relies on the quality of the data and the selection of the proper statistical models. Predicting means estimating the result of any event or situation based on what we know. It is the process of examining the prior patterns, trends, and relations among variables to make an educated guess about the future.

4 Results and discussion

The performance of the proposed algorithm has been compared with the multi-objective load balancing approach (MOLBA), effective scheduling algorithm (ESA), two-stage optimal task scheduling (TOTS), real-time task scheduling algorithm (RTSA), optimal task scheduling method (OTSM) and latency aware task scheduling algorithm (LTSA). The smart home datasets were simulated, and the python simulator was used to implement the results. Training/test split: 80% of samples are training, and 20% are testing.

4.1 Estimation of latency

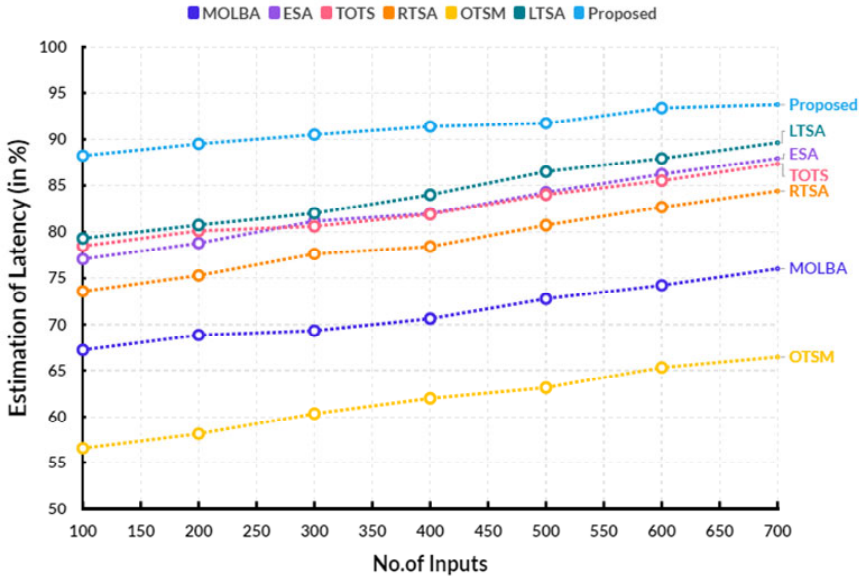
The latency estimation is computing the time required to complete a task (run through all the resources for this purpose). Deep learning techniques are used for this, where the type of task and its complexity, available resources, and network bandwidth are some factors that influence it. The algorithm estimates the latency accurately to permit dynamic task scheduling and load balancing across multiple smart home devices and resources. Table 2 expresses the latency estimation between the proposed and existing models.

Table 2 Estimation of latency (in %)

<i>No. of inputs</i>	<i>MOLBA</i>	<i>ESA</i>	<i>TOTS</i>	<i>RTSA</i>	<i>OTSM</i>	<i>LTSA</i>	<i>Proposed</i>
100	67.23	77.09	78.51	73.57	56.56	79.34	88.22
200	68.86	78.83	80.14	75.31	58.14	80.76	89.51
300	69.34	81.17	80.62	77.65	60.34	82.02	90.52
400	70.63	81.98	81.91	78.46	61.97	84.01	91.41
500	72.74	84.27	84.02	80.75	63.11	86.48	91.78
600	74.23	86.20	85.51	82.68	65.31	87.92	93.42
700	76.04	87.93	87.32	84.41	66.46	89.64	93.79

Figure 4 illustrates the assessment of latency. The current values of MOLBA were 76.04% (in an estimation interval), 87.93% (ESA), 87.32% (TOTS) (RTSA) 84.41%, OTSM 66.46%, and LTS latency 89.64% in the estimation interval. The latency that the proposed algorithms achieved was 93.79%. As a result, this reduces delays, ensuring that resources are used effectively, and thus, the overall performance of the system is better.

Figure 4 Estimation of latency (see online version for colours)



4.2 Estimation of throughput

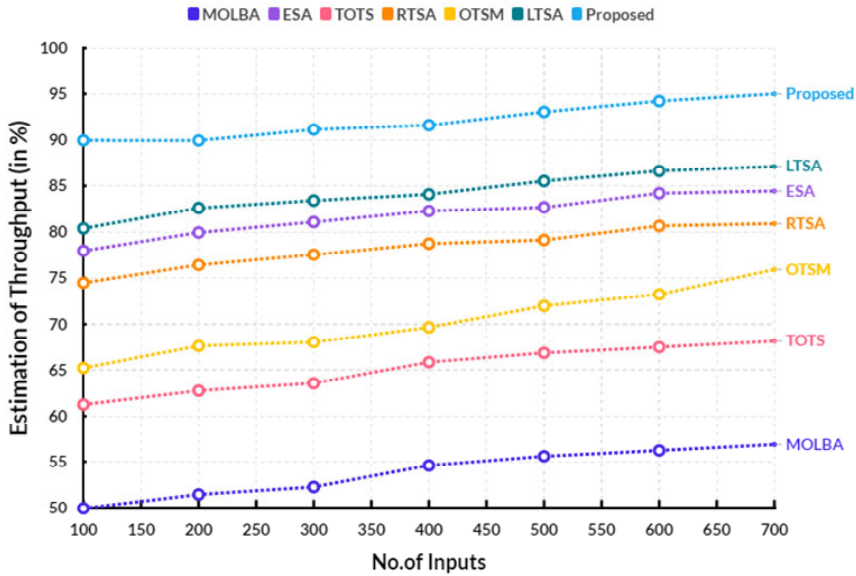
Throughput estimation refers to predicting how much the system can take and process tasks. This can be predicted by analysing the algorithm's efficiency in task distribution, resource allocation, and task execution time. However, its efficiency is counterbalanced; more tasks are completed and faster with time to complete per task via simulation and empirical evaluation of the algorithm on task completion (Integer). Table 3 shows the estimation of throughput between the proposed model and the existing model.

Table 3 Estimation of throughput (in %)

No. of inputs	MOLBA	ESA	TOTS	RTSA	OTSM	LTSA	Proposed
100	49.96	78.01	61.24	74.49	65.22	80.42	89.97
200	51.45	79.98	62.73	76.46	67.64	82.62	89.96
300	52.25	81.11	63.53	77.59	68.05	83.42	91.16
400	54.58	82.30	65.86	78.78	69.65	84.09	91.64
500	55.59	82.69	66.87	79.17	71.97	85.52	93.07
600	56.23	84.21	67.51	80.69	73.22	86.61	94.23
700	56.89	84.45	68.17	80.93	75.95	87.09	95.00

Figure 5 provides a deep analysis of throughput estimation. The existing MOLBA in an estimation interval reached 56.89%, ESA reached 84.45%, TOTS reached 68.17%, RTSA obtained 80.93%, OTSM reached 75.95%, and LTSA reached 87.09% Throughput. The throughput achieved by the algorithms of the different algorithms is 95.00%. Such an estimate can help choose the best parameters for the algorithm to try to reach the highest performance by passing the most significant throughput of a smart home system.

Figure 5 Estimation of throughput (see online version for colours)



4.3 Estimation of response time

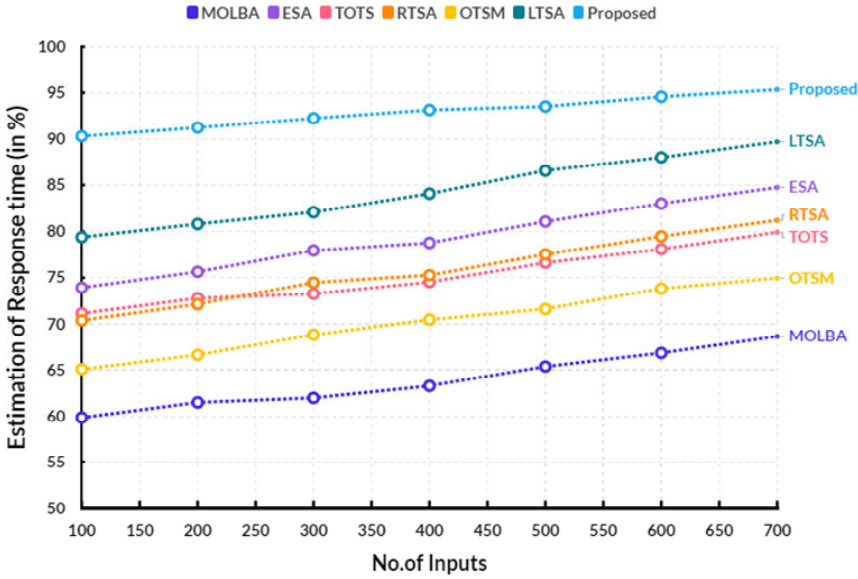
Response Time Estimation deals with how fast the system will assign and distribute computing tasks in a smart home. This estimated mission considers the complexity of functions, the computational resources available, and how practical the deep-learning algorithm is at making real-time decisions. With this estimation in place, the system will be able to utilise the most effective resources to allocate and distribute these tasks, leading to quicker and more efficient response times for smart home devices and services. Estimation of response time between proposed and existing from Table 4.

Table 4 Estimation of response time (in %)

No. of inputs	MOLBA	ESA	TOTS	RTSA	OTSM	LTSA	Proposed
100	59.85	73.91	71.13	70.39	65.06	79.41	90.32
200	61.48	75.65	72.76	72.13	66.64	80.83	91.26
300	61.96	77.99	73.24	74.47	68.84	82.09	92.27
400	63.25	78.80	74.53	75.28	70.47	84.08	93.16
500	65.36	81.09	76.64	77.57	71.61	86.55	93.53
600	66.85	83.02	78.13	79.50	73.81	87.99	94.57
700	68.66	84.75	79.94	81.23	74.96	89.71	95.34

Figure 6 illustrates the response time estimation. The existing MOLBA acquired 68.66% in the estimation interval, ESA acquired 84.75%, TOTS reached 79.94%, RTSA acquired 81.23%, OTSM 74.96%, and LTSA reached 89.71% response time. The proposed algorithms got a 95.34% Response time. The ability to take action on received notifications is essential for maintaining a frictionless user experience and improving the performance of the whole smart home system.

Figure 6 Estimation of response time (see online version for colours)



4.4 Estimation of scalability

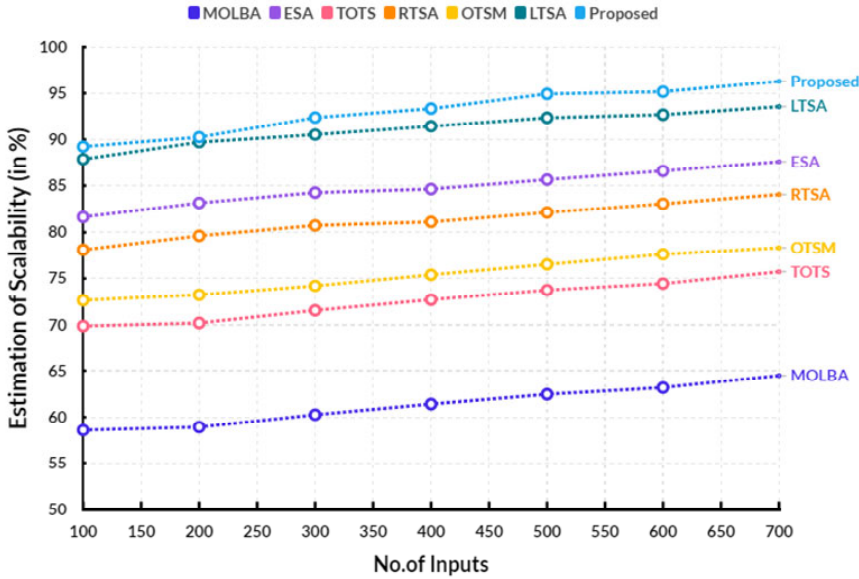
Scalability can be examined by its ability to manage more tasks and devices efficiently. Performance can be quantified through the algorithm speed, resource usage, and the accuracy of task allocation over different fog and cloud nodes. Scalability The algorithm's adaptability to changes in network dynamics, like devices connecting and disconnecting and their change in data rate with time, deduces its scalability. Table 5 shows the estimated scalability between the proposed and existing models.

Table 5 Estimation of scalability (in %)

No. of inputs	MOLBA	ESA	TOTS	RTSA	OTSM	LTSA	Proposed
100	58.59	81.65	69.87	78.13	72.62	87.85	89.23
200	58.92	83.15	70.20	79.63	73.21	89.72	90.27
300	60.26	84.26	71.54	80.74	74.19	90.55	92.40
400	61.40	84.64	72.68	81.12	75.40	91.46	93.36
500	62.45	85.65	73.73	82.13	76.54	92.38	94.93
600	63.16	86.58	74.44	83.06	77.65	92.71	95.17
700	64.46	87.58	75.74	84.06	78.35	93.58	96.28

Figure 7 shows the scalability estimation output. In a prediction interval, the current MOLBA attained 64.46%, ESA had 87.58%, TOTS 75.74%, RTSA 84.06%, OTSM 78.35%, and LTSA 93.58% Scalability. The algorithms above resulted in 96.28% Scalability. This is where the scalability of an algorithm in real-world conditions also gets determined, as well as how well the algorithm handles a diverse set of tasks and devices.

Figure 7 Estimation of scalability (see online version for colours)



4.5 Estimation of fault-tolerance

The fault tolerance will be evaluated by how well the algorithm can tolerate unexpected errors or failures during these tasks. We achieve this simulation by running the algorithm on multiple fault scenarios and capturing the time tasks are completed, the number of functions each node executed, and the ratio of resources used vs. resources provided. In addition, the algorithm under the enormous workload can carry out tests for durability and scheduling tasks, and load balance ability was evaluated. Table 6 shows the estimation of fault tolerance between the proposed and existing models.

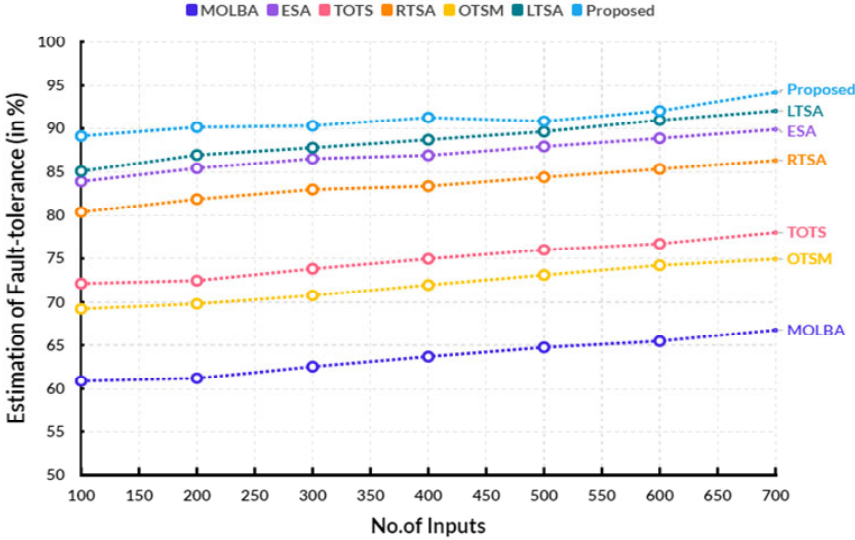
Table 6 Estimation of fault-tolerance (in %)

No. of inputs	MOLBA	ESA	TOTS	RTSA	OTSM	LTSA	Proposed
100	60.89	83.95	72.17	80.43	69.22	85.11	89.14
200	61.22	85.45	72.50	81.93	69.81	86.98	90.15
300	62.56	86.56	73.84	83.04	70.79	87.81	90.31
400	63.70	86.94	74.98	83.42	72.00	88.72	91.27
500	64.75	87.95	76.03	84.43	73.14	89.64	90.84
600	65.46	88.88	76.74	85.36	74.25	90.97	92.04
700	66.76	89.88	78.04	86.36	74.95	92.05	94.20

Figure 8 shows an estimation of fault tolerance. By the time that an estimate interval is done, the MOLBA had achieved 66.76%, the ESA had been 89.88%, TOTS obtained 78.04% RTSA and 86.36%, OTSM had 74.95% and the LTSA had been 92.05% fault-tolerance. Also, their proposed algorithm achieved 94.20% for fault-tolerance.

When analysed and measured, these factors can provide the fault-tolerance of the algorithm, which is to be considered for implementation in cloud fog architecture for smart homes.

Figure 8 Estimation of fault-tolerance (see online version for colours)



4.6 Estimation of scheduling capacity

Scheduling capacity is the upper limit on the amount of work the system can do in a given period. This requires the analysis of the processing and memory capacities of the fog nodes and those of the cloud servers, in addition to the deep learning tasks' complexities. The algorithm will compute the best distribution of tasks to the application using the fog nodes and cloud servers to perform load balancing and execution efficiency. The estimation of scheduling capacity between the proposed and existing models is shown in Table 7.

Table 7 Estimation of scheduling capacity (in %)

No. of inputs	MOLBA	ESA	TOTS	RTSA	OTSM	LTSA	Proposed
100	56.85	88.00	68.13	84.48	77.76	92.30	94.41
200	56.96	87.98	68.24	84.46	77.93	92.57	94.91
300	57.07	87.96	68.35	84.44	78.10	92.84	95.41
400	57.18	87.94	68.46	84.42	78.27	93.11	95.91
500	57.29	87.92	68.57	84.40	78.44	94.38	96.41
600	57.40	87.90	68.68	84.38	78.61	95.65	96.91
700	57.51	87.88	68.79	84.36	78.78	96.92	97.41

Figure 9 Estimation of scheduling capacity (see online version for colours)

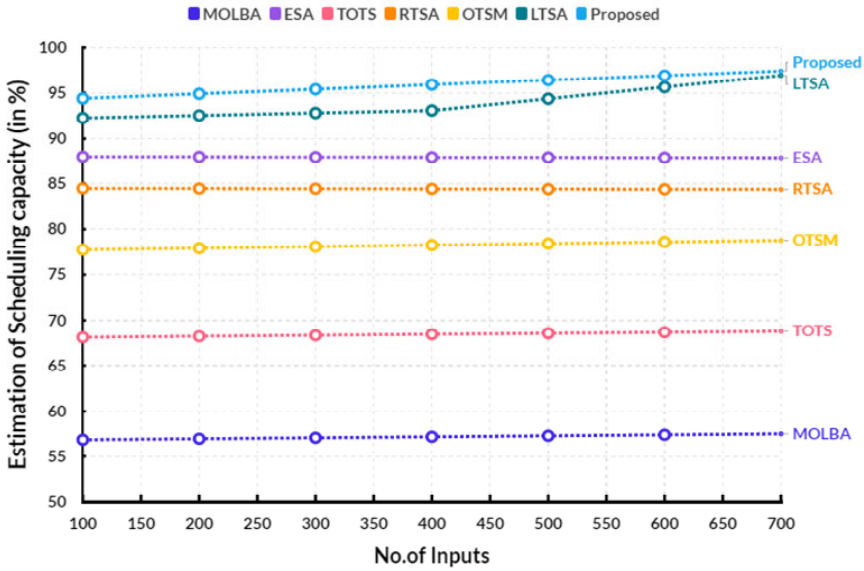


Figure 9 shows the estimation of scheduling capacity. The existing MOLBA had a scheduling capacity of 57.51% in the estimation interval, while ESA was 87.88%, TOTS 68.79%, RTSA 84.36%, OTSM 78.78%, and LTSA 96.92%. The results obtained from the scheduling capacity for the proposed algorithms were 97.41%. This estimate allows you to use resources better and make the system capable of handling the workload without any performance degradation or delays in task execution.

4.7 Estimation of load balancing

Load balancing is the efficiently distributed technique through task allocation and resource management within a smart home environment. It employs deep learning algorithms to analyse user preferences, available resources, and network conditions to assign tasks to the most appropriate device dynamically. It also helps balance the workload across various pieces of hardware to ensure that the resources are used most effectively. Table 8 shows the estimation of load balancing capacity between the proposed and existing models.

Table 8 Estimation of load balancing capacity (in %)

No. of inputs	MOLBA	ESA	TOTS	RTSA	OTSM	LTSA	Proposed
100	56.98	87.10	68.26	83.58	77.20	95.27	97.79
200	53.88	84.27	65.16	80.75	73.86	91.76	96.56
300	50.78	81.44	62.06	77.92	70.52	88.25	95.33
400	47.68	78.61	58.96	75.09	67.18	84.74	94.10
500	44.58	75.78	55.86	72.26	63.84	81.23	93.87
600	41.48	72.95	52.76	69.43	60.50	77.72	92.64
700	38.38	70.12	49.66	66.60	57.16	74.21	90.41

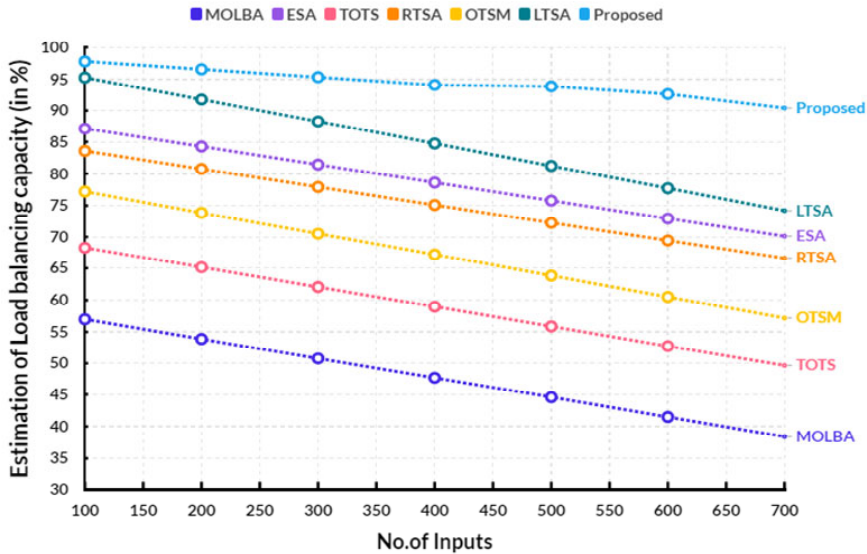
Figure 10 Estimation of load balancing capacity (see online version for colours)

Figure 10 depicts the estimated load balancing capacity. MOLBA achieved 38.38%, ESA achieved 70.12%, and TOTS achieved 49.66% Load balancing capacity in an estimation interval. RTSA got 66.60%, OTSM achieved 57.16%, and LTSA achieved 74.21% Load balancing capacity. The load balancing capacity achieved by the different algorithms proposed was 91.41%. It even helps conserve power consumption and make the smart home system more performance efficient and a better home for the users.

4.8 Estimation of priority management

Managing priorities is determining which tasks are of higher value and need to be done sooner. In a cloud fog facility, this process is kept to ensure the accessibility of resources and implementation of undertakings at due time. To estimate priority management, it is necessary to review the workload and specific specifications of various tasks, such as computing and communication requirements, and to prioritise them according to their level of importance. Table 9 illustrates the importance of priority management estimating with the proposed and existing models.

Table 9 Estimation of priority management (in %)

No. of inputs	MOLBA	ESA	TOTS	RTSA	OTSM	LTSA	Proposed
100	52.68	82.95	63.96	79.43	73.13	90.44	96.18
200	52.07	82.12	63.35	78.60	72.24	89.90	95.61
300	51.66	81.72	62.94	78.20	72.16	89.60	95.91
400	51.12	81.03	62.40	77.51	71.54	89.14	95.63
500	50.61	80.42	61.89	76.90	71.05	88.72	95.49
600	50.10	79.80	61.38	76.28	70.57	88.30	95.36
700	49.59	79.19	60.87	75.67	70.08	87.88	95.22

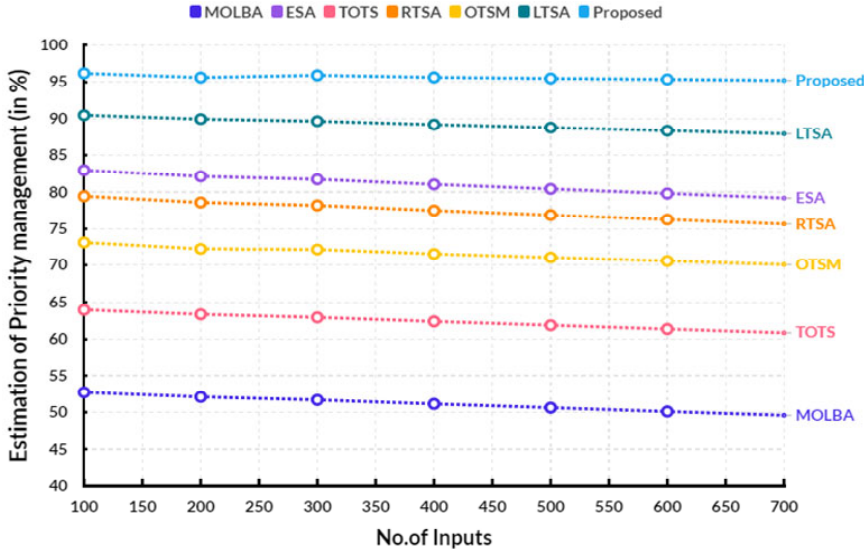
Figure 11 Estimation of priority management (see online version for colours)

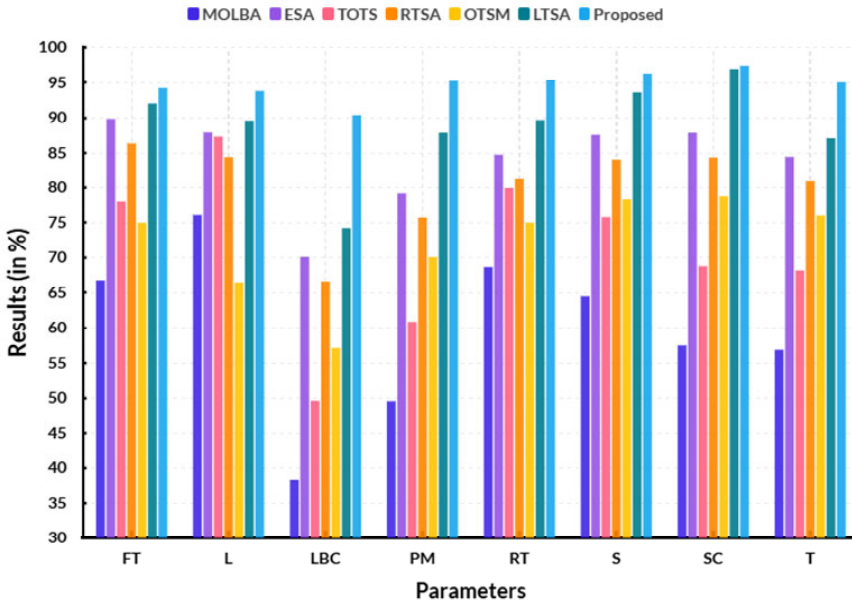
Figure 11 provides the priority management estimation. The current MOLBA reached 49.59% in an estimation interval, ESA reached 79.19%, TOTS reached 60.87%, RTSA obtained 75.67%, OTSM reached 70.08%, LTSA reached 87.88% priority management. The algorithms received priority management of 95.22%. Table 10 shows the convergence of performance.

Figure 12 shows the convergence of performance. The proposed algorithm has provided 93.79% latency, 95.00% throughput, 95.34% response time, 96.28% scalability, 94.20% fault-tolerance, 97.41% scheduling capacity, 91.41% load balancing capacity, 95.22% priority management. This information is later used by the deep learning-based scheduler, which schedules tasks dynamically and balances the load, taking into account resource availability, energy consumption, task deadlines, etc. A smart home environment can learn and estimate prioritisation correctly, and this algorithm increases the performance of the smart home environment by resource utilisation.

Table 10 Convergence of performance (in %)

Parameters	MOLBA	ESA	TOTS	RTSA	OTSM	LTSA	Proposed
Latency (L)	76.04	87.93	87.32	84.41	66.46	89.64	93.79
Throughput (T)	56.89	84.45	68.17	80.93	75.95	87.09	95.00
Response time (RT)	68.66	84.75	79.94	81.23	74.96	89.71	95.34
Scalability (S)	64.46	87.58	75.74	84.06	78.35	93.58	96.28
Fault-tolerance (FT)	66.76	89.88	78.04	86.36	74.95	92.05	94.20
Scheduling capacity (SC)	57.51	87.88	68.79	84.36	78.78	96.92	97.41
Load balancing capacity (LBC)	38.38	70.12	49.66	66.60	57.16	74.21	90.41
Priority management (PM)	49.59	79.19	60.87	75.67	70.08	87.88	95.22

Figure 12 Convergence of performance (see online version for colours)



The values for the proposed algorithm were fixed through a combination of thorough research and testing. First, various techniques and strategies were explored to identify the areas that needed improvement. This involved studying the existing algorithms and identifying their strengths and weaknesses. During the testing phase, various parameters were measured, such as latency, throughput, response time, scalability, fault tolerance, scheduling capacity, load balancing capacity, priority management, and cost efficiency. These values were then compared with the target values set for each parameter to evaluate the effectiveness of the algorithms. The proposed algorithms were continuously refined and optimised based on the test results until they achieved the desired performance levels. This involved tweaking the algorithms, adjusting parameters, and fine-tuning the algorithms to address the identified issues effectively. The high percentages achieved in each parameter were a result of the rigorous testing and optimisation process. By constantly striving to improve and optimise the algorithms, the desired performance levels were achieved, resulting in the high percentile values for each parameter. Additionally, the use of advanced technologies and techniques in the algorithms also contributed to their success in achieving these high values.

The service control output manager (SCOM) is responsible for managing the outbound queue, which includes the transmission queue. It is responsible for managing the delivery of messages from the format queue to the outbound queue and then to the appropriate destination queue on the matched machine. When a message is sent to a format queue, the SCOM processes the message and determines which machine it is intended for. It then moves the message to the transmission queue, which acts as a temporary holding area before the message is delivered to its destination. The transmission queue is a specialised queue that is designed for handling messages in a specific order. Messages are placed in a FIFO design, meaning the first message in will be the first message out. This ensures that messages are delivered in the correct sequence,

preventing any errors or confusion when they reach their destination. When the message reaches the front of the transmission queue, it is assigned a unique identifier and then sent to the appropriate destination queue on the matched machine. This process continues until all messages in the transmission queue have been delivered.

1 *Benefits*

- *Increased efficiency:* By using the effective algorithm for predicting load and dynamic task scheduling, cloud fog architecture in smart homes can achieve higher efficiency in terms of resource utilisation. This is because the algorithm takes into account real-time data and predicts the future load to schedule tasks accordingly. This leads to better allocation and utilisation of resources, which ultimately results in improved efficiency.
- *Cost savings:* As the algorithm predicts the load and schedules tasks dynamically, it ensures that resources are used cost-effectively. This means that unnecessary resources are not allocated, thus leading to cost savings for smart homeowners.
- *Faster response time:* Smart homes rely on cloud fog architecture to perform various tasks and services. The effective algorithm for predicting load and dynamic task scheduling ensures that tasks are allocated to the most suitable resources, leading to faster response times. This is crucial for real-time applications in smart homes, such as security systems and smart appliances.

2 *Limitations*

- The algorithm may not be able to handle large-scale smart homes with a large number of devices and tasks. This can potentially lead to performance issues and delays in task scheduling.
- The accuracy of the load and task prediction heavily depends on the quality and accuracy of the data gathered from the smart home devices. Inaccurate or inconsistent data can result in inaccurate predictions and lead to inefficient task scheduling.
- The algorithm may not be able to adapt to sudden changes in the workload or task requirements in the smart home environment. This can result in suboptimal task scheduling and potentially cause delays or interruptions in critical tasks.

3 *Applications*

- *Energy management:* This algorithm can be used to schedule tasks and manage energy consumption in smart homes efficiently. It can predict the load on the cloud and fog nodes and allocate tasks accordingly, optimising energy usage and reducing costs for homeowners.
- *Home automation:* With the increasing popularity of smart home devices and automation systems, this algorithm can be applied to schedule various tasks such as temperature control, lighting, and security systems. It can consider factors such as device usage patterns and user preferences to create an efficient task schedule for a seamless home automation experience.

- *Healthcare monitoring:* Cloud fog architecture is also helpful in the healthcare sector, particularly for remote patient monitoring in smart homes. This algorithm can be used to predict the load on the network and ensure the timely transfer of vital health data from IoT devices to healthcare professionals. It can also prioritise urgent tasks and ensure real-time monitoring for the safety and well-being of patients.

5 Conclusions

This paper proposes a deep learning-based dynamic task scheduling and load balancing algorithm for an efficient and scalable solution for smart homes over cloud fog architecture. The proposed algorithm has provided 93.79% latency, 95.00% throughput, 95.34% response time, 96.28% scalability, 94.20% fault-tolerance, 97.41% scheduling capacity, 91.41% load balancing capacity, 95.22% priority management. Deep learning was used and applied in smart homes with cloud-fog computing architecture to schedule tasks dynamically and load balancing, and they showed promising results. The algorithm uses multiple parameters for task scheduling to distribute tasks effectively between fog and cloud resources. In addition, the algorithm can learn and adapt to the environment through deep learning, thereby increasing its robustness and reliability. This not only enhances the performance and efficiency of smart homes in general but also benefits the scalability and flexibility required for smart home applications in the future.

5.1 Future works

Based on the work mentioned above, future improvements of the from-scratch deep learning-based task scheduling and load balancing algorithms will have the following:

- *Real-time analytics:* If we can integrate this with real-time data analytics, the algorithm can learn about the current situation of the various devices performing within this smart home. Based on this information, job scheduling and load balancing can then be increased and optimised to provide efficient resource utilisation and, in turn, better performance.
- *Multi-objective optimisation:* The algorithm can be enhanced to consider multiple objectives, such as request response time, energy consumption, request cost, task scheduling, and load balance. This approach, aligned with the user's preferences, can lead to a more comprehensive decision-making process, increasing productivity and reducing expenses in a smart home.
- *IoT device prioritisation:* As the number of IoT devices in smart homes increases, it would be useful to prioritise them over others and the tasks they perform. This could be based on urgency, importance, or size of input. With the introduction of this enhancement, the system is now capable of managing task scheduling and load balancing that best suits the household.

References

- Abdulazeez, D.H. and Askar, S.K. (2023) 'Offloading mechanisms based on reinforcement learning and deep learning algorithms in the fog computing environment', *IEEE Access*, Vol. 11, No. 2, pp.12555–12586.
- Abinavkrishnaa, R., Raghuram, G., Varghese, A., Gowri, G.U. and Rahila, J. (2023) 'Scaling strategies for enhanced system performance: navigating stateful and stateless architectures', *FMDB Transactions on Sustainable Computer Letters*, Vol. 1, No. 4, pp.241–254.
- Abohamama, A.S., El-Ghamry, A. and Hamouda, E. (2022) 'Real-time task scheduling algorithm for IoT-based applications in the cloud-fog environment', *Journal of Network and Systems Management*, Vol. 30, No. 4, p.54.
- Alizadeh Javaheri, S.D., Ghaemi, R. and Monshizadeh Naeen, H. (2024) 'An autonomous architecture based on reinforcement deep neural network for resource allocation in cloud computing', *Computing*, Vol. 106, No. 2, pp.371–403.
- Anand, P.P., Jayanth, G., Rao, K.S., Deepika, P., Faisal, M. and Mokdad, M. (2024) 'Utilising hybrid machine learning to identify anomalous multivariate time-series in geotechnical engineering', *AVE Trends In Intelligent Computing Systems*, Vol. 1, No. 1, pp.32–41.
- Chhilar, R.S. (2023) 'Novel approach for improving secure scheduling in fog environment and in context of smart homes', *International Journal of Intelligent Systems and Applications in Engineering*, Vol. 11, No. 1s, pp.23–29.
- Cirillo, S., Polese, G., Salerno, D., Simone, B. and Solimando, G. (2023) 'Towards flexible voice assistants: evaluating privacy and security needs in iot-enabled smart homes', *FMDB Transactions on Sustainable Computer Letters*, Vol. 1, No. 1, pp.25–32.
- Fahimullah, M., Ahvar, S., Agarwal, M. and Trocan, M. (2024) 'Machine learning-based solutions for resource management in fog computing', *Multimedia Tools and Applications*, Vol. 83, No. 8, pp.23019–23045.
- Hasan, M.K., Sundararajan, E., Islam, S., Ahmed, F.R.A., Babiker, N.B.M., Alzahrani, A.I. and Khan, M.A. (2024) 'A novel segmented random search based batch scheduling algorithm in fog computing', *Computers in Human Behavior*, Vol. 158, No. 9, p.108269.
- Iftikhar, S., Ahmad, M.M.M., Tuli, S., Chowdhury, D., Xu, M., Gill, S.S. and Uhlig, S. (2023) 'HunterPlus: AI based energy-efficient task scheduling for cloud-fog computing environments', *Internet of Things*, Vol. 21, No. 4, p.100667.
- Jasper, K.D., Jaishnav, M.N., Chowdhury, M.F., Badhan, R. and Sivakani, R. (2024) 'Defend and secure: a strategic and implementation framework for robust data breach prevention', *AVE Trends In Intelligent Computing Systems*, Vol. 1, No. 1, pp.17–31.
- Karthik, S.S. and Kavithamani, A. (2021) 'Fog computing-based deep learning model for optimization of microgrid-connected WSN with load balancing', *Wireless Networks*, Vol. 27, No. 4, pp.2719–2727.
- Memari, P., Mohammadi, S.S., Jolai, F. and Tavakkoli-Moghaddam, R. (2022) 'A latency-aware task scheduling algorithm for allocating virtual machines in a cost-effective and time-sensitive fog-cloud architecture', *The Journal of Supercomputing*, Vol. 78, No. 1, pp.93–122.
- Minu, M.S., Subashka Ramesh, S.S., Canessane, R., Al-Amin, M. and Bin Sulaiman, R. (2023) 'Experimental analysis of UAV networks using oppositional glowworm swarm optimization and deep learning clustering and classification', *FMDB Transactions on Sustainable Computing Systems*, Vol. 1, No. 3, pp.124–134.
- Nagaraj, B.K. and Subhashni, R. (2023) 'Explore LLM architectures that produce more interpretable outputs on large language model interpretable architecture design', *FMDB Transactions on Sustainable Computer Letters*, Vol. 1, No. 2, pp.115–129.
- Nagarajan, S.M., Deverajan, G.G., Chatterjee, P., Alnumay, W. and Ghosh, U. (2021) 'Effective task scheduling algorithm with deep learning for internet of health things (IoHT) in sustainable smart cities', *Sustainable Cities and Society*, Vol. 71, No. 8, p.102945.

- Nayak, A., Tripathy, S.S., Bebortta, S. and Tripathy, B. (2024) 'An intelligent study towards nature-inspired load balancing framework for fog-cloud environments', *2024 IEEE International Conference for Women in Innovation, Technology & Entrepreneurship (ICWITE)*, IEEE, pp.168–173.
- Nethaji, S.V. and Chidambaram, M. (2021) 'Resource optimization in fog computing with shift-invariant deep convolutive load balancing', *Webology*, Vol. 18, No. 6, pp.3845–3861.
- Pal, S., Jhanjhi, N.Z., Abdulbaqi, A.S., Akila, D., Alsubaei, F.S. and Almazroi, A.A. (2023) 'An intelligent task scheduling model for hybrid internet of things and cloud environment for big data applications', *Sustainability*, Vol. 15, No. 6, p.5104.
- Rajasekaran, N., Jagatheesan, S.M., Krithika, S. and Albanchez, J.S. (2023) 'Development and testing of incorporated ASM with MVP architecture model for android mobile app development', *FMDB Transactions on Sustainable Computing Systems*, Vol. 1, No. 2, pp.65–76.
- Raju, M.R. and Mothku, S.K. (2023) 'Delay and energy aware task scheduling mechanism for fog-enabled IoT applications: a reinforcement learning approach', *Computer Networks*, Vol. 224, No. 4, p.109603.
- Ramezani Shahidani, F., Ghasemi, A., Toroghi Haghighat, A. and Keshavarzi, A. (2023) 'Task scheduling in edge-fog-cloud architecture: a multi-objective load balancing approach using reinforcement learning algorithm', *Computing*, Vol. 105, No. 6, pp.1337–1359.
- Renjith, E.J. (2024) 'Revolutionizing cloud security: Innovative approaches for safeguarding data integrity', *FMDB Transactions on Sustainable Computer Letters*, Vol. 2, No. 2, pp.111–119.
- Salehnia, T., Seyfollahi, A., Raziani, S., Noori, A., Ghaffari, A., Alsoud, A.R. and Abualigah, L. (2024) 'An optimal task scheduling method in IoT-fog-cloud network using multi-objective moth-flame algorithm', *Multimedia Tools and Applications*, Vol. 83, No. 12, pp.34351–34372.
- Saxena, R., Sharma, V. and Saxena, R.R. (2023) 'Transforming medical education: multi-keyword ranked search in cloud environment', *FMDB Transactions on Sustainable Computing Systems*, Vol. 1, No. 3, pp.135–146.
- Sayedelahl, M.A. and Farouk, R.M. (2024) 'Hybrid approach to image segmentation with artificial neural networks and Gabor wavelets', *AVE Trends in Intelligent Computing Systems*, Vol. 1, No. 2, pp.77–90.
- Shadroo, S., Rahmani, A.M. and Rezaee, A. (2021) 'The two-phase scheduling based on deep learning in the internet of things', *Computer Networks*, Vol. 185, No. 2, p.107684.
- Shamsa, Z., Rezaee, A., Adabi, S. and Rahmani, A.M. (2024) 'A decentralized prediction-based workflow load balancing architecture for cloud/fog/IoT environments', *Computing*, Vol. 106, No. 1, pp.201–239.
- Sharma, O., Rathee, G., Kerrache, C.A. and Herrera-Tapia, J. (2023) 'Two-stage optimal task scheduling for smart home environment using fog computing infrastructures', *Applied Sciences*, Vol. 13, No. 5, p.2939.
- Shruthi, G., Mundada, M.R., Sowmya, B.J. and Supreeth, S. (2022) 'Mayfly Taylor optimisation-based scheduling algorithm with deep reinforcement learning for dynamic scheduling in fog-cloud computing', *Applied Computational Intelligence and Soft Computing*, Vol. 2022, No. 1, p.2131699.
- Soji, E.S., Karthik, S., Priscila, S.S., Hanirex, D.K., Rajest, S.S. and Sebastian, A. (2024) 'Designing an e-voting system architecture using the STAP protocol', *FMDB Transactions on Sustainable Intelligent Networks*, Vol. 1, No. 2, pp.96–109.
- Talaat, F.M., Ali, H.A., Saraya, M.S. and Saleh, A.I. (2022) 'Effective scheduling algorithm for load balancing in fog environment using CNN and MPSO', *Knowledge and Information Systems*, Vol. 64, No. 3, pp.773–797.
- Thirunagalingam, A. (2024a) 'Bias detection and mitigation in data pipelines: ensuring fairness and accuracy in machine learning', *AVE Trends in Intelligent Computing Systems*, Vol. 1, No. 2, pp.116–127.

- Thirunagalingam, A. (2024b) ‘Transforming real-time data processing: the impact of AutoML on dynamic data pipelines’, *FMDB Transactions on Sustainable Intelligent Networks*, Vol. 1, No. 2, pp.110–119.
- Umair, M., Cheema, M.A., Afzal, B. and Shah, G. (2023) ‘Energy management of smart homes over fog-based IoT architecture’, *Sustainable Computing: Informatics and Systems*, Vol. 39, No. 9, p.100898.
- Wang, Z., Goudarzi, M., Gong, M. and Buyya, R. (2024) ‘Deep reinforcement learning-based scheduling for optimizing system load and response time in edge and fog computing environments’, *Future Generation Computer Systems*, Vol. 152, No. 1, pp.55–69.
- Yu, D., Ma, Z. and Wang, R. (2022) ‘Efficient smart grid load balancing via fog and cloud computing’, *Mathematical Problems in Engineering*, Vol. 2022, No. 1, p.3151249.
- Zanardo, E. (2024) ‘Chronostamp: a general-purpose run-time for data-flow computing in a distributed environment’, *AVE Trends in Intelligent Computing Systems*, Vol. 1, No. 2, pp.106–115.