International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 6, 2025
https://theaspd.com/index.php

Linear Regression And Recurrent Neural Network Ensemble For Swift Tensile Load Balancer For Static And Dynamic Clouds

Ananya Saha[1], Dr. S.K.Manju Bargavi[2]
[1]Research Scholar, Jain (Deemed-to-be) University, Bengaluru, Karnataka
[2]Professor, Jain (Deemed-to-be University), Bengaluru, Karnataka

Abstract
Efficient load balancing is essential for optimizing resource utilization and performance in both static and dynamic cloud environments. Dynamic cloud environments are used to have frequent workload fluctuations, thus, demand adaptive load balancing strategies are essential to handle the criteria.In general, the load in a cloud computing refers the twin paired loads namely computational load and communication load. Real-time monitoring and machine learning based resource reallocation is the most modern innovative strategy to handle the balance between the computational and communication loads. This work introduced as "Linear Regression and Recurrent Neural Network Ensemble for Swift Tensile Load balancer for Static and Dynamic Clouds (LRESTL)" is the hormonic integration of three novel modules namely Tailored Elastic Net Load Prognosticator, Custom Echo State RNN Load Tracker, and Swift Tensile Consolidator". LRESTL is implemented and evaluated in a real-time cloud environment to assess benchmark metrics such as resource utilization, balance degree, average response time, migration cost, throughput based on the number of tasks, and throughput based on the number of virtual machines.
Keywords: Communication Load, Computational load, Dynamic Cloud, Load Balance, Recurrent Neural Network, Resource allocation, Server Optimization, Static Cloud, Linear Regression

1. INTRODUCTION

The rise of cloud computing has transformed how businesses handle and deploy IT resources. By utilizing cloud services, companies can tap into scalable and adaptable computing power via the internet, minimizing the reliance on costly on-site hardware [1]. This transition facilitates quick application deployment, economical resource scaling, and improved teamwork among distributed teams. The cloud offers a variety of services, including infrastructure, platforms, and software, customized to fit various business requirements [2]. As cloud technology advances, it fosters innovation, accelerates digital transformation, and gives organizations the flexibility to remain competitive in a fast-changing landscape [3].Load balancing plays a crucial role in cloud computing by ensuring that workloads are evenly distributed across multiple servers or resources. This process is vital for maintaining high performance, reliability, and availability of cloud-based applications and services. Effective load balancing helps prevent any single server from becoming overwhelmed, which can lead to slowdowns or outages [4]. By dynamically distributing incoming traffic and balancing resource utilization, load balancing enhances the overall efficiency of the cloud environment, reduces the risk of downtime, and supports seamless scaling as demand fluctuates [5]. This approach improves the user experience while also enhancing resource management and reducing operational costs, making load balancing a crucial part of a well-functioning and adaptable cloud infrastructure.

In cloud computing, managing computational and communication loads is key to optimizing system performance and resource use [6]. Computational load pertains to the processing power needed for tasks like running applications or conducting data analysis. Properly managing this load ensures resources are used efficiently [7], avoiding performance issues and keeping applications running smoothly. Communication load, on the other hand, involves the amount of data exchanged between cloud services, users, and different cloud components [8]. Effective management of communication load helps reduce delays and congestion, ensuring fast and reliable data transfer. Balancing both types of load is essential for a responsive and efficient cloud system, impacting scalability, user experience, and overall operational effectiveness [9].

Dynamic load monitoring in cloud computing is essential for maintaining optimal performance and resource efficiency in an ever-changing environment [10]. This approach involves continuously tracking and analyzing the workload on cloud resources in real time, allowing for immediate adjustments based

on current demands. By dynamically monitoring load conditions, cloud systems can automatically allocate or deallocate resources, balance traffic, and address potential bottlenecks before they impact performance [11]. This proactive strategy enhances scalability, ensures consistent service availability, and improves user experience by adapting to fluctuating demands [12]. Effective dynamic load monitoring also helps optimize resource utilization, reduce operational costs, and support seamless transitions during peak usage or unexpected spikes in activity [13].

The impact of machine learning on cloud load balancing is transformative, as it introduces advanced capabilities for optimizing resource distribution and enhancing system performance. Machine learning algorithms analyze vast amounts of historical and real-time data to predict traffic patterns, identify potential bottlenecks, and make informed decisions about resource allocation [14]. By leveraging these insights, cloud load balancing can dynamically adjust to fluctuating demands, ensuring that workloads are efficiently distributed and minimizing latency. Machine learning also enables predictive scaling, where resources are proactively scaled up or down based on anticipated needs, reducing the risk of surplus resource provisioning or inadequate resource allocation [15]. This results in improved system reliability, cost efficiency, and user experience, as machine learning-driven load balancing continuously adapts to changing conditions and delivers optimal performance across the cloud infrastructure.

2. Existing Methods

Examining existing methods is the primary task for new research work as it highlights gaps, shortcomings, and opportunities for enhancement, ensuring that the research offers new insights instead of repeating previous work. A Fast Converging and Globally Optimized Approach for Load Balancing in Cloud Computing [16], Energy efficient resource utilization and load balancing in virtual machines using prediction algorithms [17], A novel weight-assignment load balancing algorithm for cloud applications [18], Task Scheduling Mechanism Based on Reinforcement Learning in Cloud Computing [19], and CSO-ILB: chicken swarm optimized inter-cloud load balancer for elastic containerized multi-cloud environment [20] are handpicked from the vast collection due to their versatility and comparable features. The methodologies, advantages, and limitations of these methods are studied deeply in this section.

2.1. A Fast Converging and Globally Optimized Approach for Load Balancing in Cloud Computing(GWOPSO)

In 2023, M. S. Al Reshan et al., proposed A Fast Converging and Globally Optimized Approach for Load Balancing in Cloud Computing work which is the ensemble of Gray Wolf Optimization (GWO) and Particle Swarm Optimization (PSO). This research introduces Swarm Intelligence as a solution for load balancing in cloud computing, focusing on Particle Swarm Optimization and Grey Wolf Optimization. It highlights that existing methods like genetic algorithms and others do not address both load balancing convergence time and global optimization. The proposed combined GWO-PSO approach leverages the strengths of both techniques to achieve fast convergence and effective global optimization, thereby enhancing system efficiency and resource allocation. The results show that this approach reduces the overall response time by 12% compared to traditional methods and improves the convergence of PSO, achieving a best optimal value of 97.253% from the objective function.Experiments were conducted using the MATLAB platform to evaluate the optimization and convergence performance of PSO, GWO, and the combined GWO-PSO algorithms. MATLAB, a powerful and widely-used scientific computing language developed by MathWorks, supports various operating systems and offers ease of use, making it a staple in research and engineering. By incorporating toolkits, MATLAB can be customized and enhanced for specific tasks or fields, making it an increasingly versatile and dominant tool for complex computational experiments.

The hybrid GWO-PSO load-balancing technique achieves globally optimal rapid convergence, leading to reduced response time, increased throughput, and low execution time while efficiently balancing loads among virtual machines is the advantage of GWOPSO method. Whereas, the basic GWO algorithm does not perfectly balance exploration and exploitation, potentially affecting its ability to consistently achieve optimal solutions, and the proposed technique focuses on fast convergence and system efficiency but does not prioritize Degree of Balance, which is the identified limitations of GWO_PSO work.

2.2. Energy efficient resource utilization and load balancing in virtual machines using prediction algorithms (EPRUA)

In 2023, P. Udayasankaran et.al., introduced EPRUA work for optimizing the load balancing process in the cloud computing environments. EPRUAwork focuses on various load balancing techniques that distribute incoming traffic across multiple targets to optimize data center efficiency. As power supply capacity increases and demand continues to grow, there is a need to reduce the burden on active server infrastructure. This approach addresses multiple challenges in the data center environment, though the behavior of systems and consumers is not clearly defined. A novel resource optimization framework is proposed to identify jobs with minimal resource utilization, aiming to enhance energy efficiency and reduce overall consumption in data centers. EPRUA methodology intermittently assesses resources based on anticipated future VM requests to prevent server overload and save energy. EPRUA approach aims to match energy use with the fluctuating demands of applications effectively. In addition, a novel user behavior forecasting system, driven by descriptive analytics, predicts session lengths and workload arrivals to optimize resource usage. A unique optimization framework is also proposed to minimize energy consumption during job execution by forecasting the expected workload intensity. Experimental validation using Google Cloud trace logs demonstrates that this framework effectively improves reliability and reduces energy costs in data center operations.he experimental setup was carried out on a system equipped with an Intel Core i5 processor, 16GB of RAM, and 1000GB of storage. The implementation utilized the Java programming environment, specifically using JDK 1.8, NetBeans 8.0.2, and a MYSQL 5.5 database.
The EPRUA algorithm effectively reduces energy consumption and processing time during resource allocation, which improves overall system performance and availability in cloud servers, which is the observed advantage of EPRUA work. The comparison of various parameters, such as request time, processing time, CPU utilization, and energy usage, presents complex problems, making it challenging to reduce time variations and optimize energy utilization levels efficiently, that causes higher migration costs, which is identified as the limitation.

2.3. A novel weight-assignment load balancing algorithm for cloud applications (NWALBA)
Adewojo. A.A. et al., proposed NWALBA in 2023 to address flash crowds and resource failure problems in cloud computing environments.An innovative weight assignment load balancing algorithm is introduced that integrates five carefully chosen server metrics to effectively allocate the workload of three-tier web applications across virtual machines. Using a private cloud environment running OpenStack, the load distribution capabilities of the new algorithm, along with a baseline algorithm and a round-robin approach, were assessed. Performance comparisons of the three algorithms were conducted by simulating resource failures and flash crowds, with response times meticulously recorded. Results indicate that the approach enhances average response times by 12.5% compared to the baseline algorithm and 22.3% relative to the round-robin method during flash crowds. Additionally, in resource failure scenarios, the average response time improved by 20.7% compared to the baseline algorithm and 21.4% compared to the round-robin algorithm. These findings demonstrate that the novel algorithm exhibits greater resilience to varying loads and resource failures than the baseline algorithms.NWALBA algorithm was tested on a private cloud infrastructure running the training version of OpenStack. Apache JMeter, an open-source tool for load testing and performance analysis, was installed and operated on a dedicated standalone machine. HAProxy was set up alongside the load-balancing solution on two VM instances. Eight medium-sized VM instances were deployed as application servers, with two additional medium-sized VMs in standby mode, and four large VM instances were used as database servers. Each application server was equipped with technologies to host the case study applications and included a component of the load-balancing module responsible for monitoring utilization values.
NWALBAwork effectively improves average response times in both resource failure and flash crowd scenarios, demonstrating its ability to handle performance degradation during these conditions, which is identified as the advantage. The experiments have been limited to a specific case study e-commerce application, and further evaluation is needed to assess the algorithm's performance across various types of resource failures and different deployment environments, such as serverless and multi-cloud setups – which is the observed disadvantage of NWALBA work.

2.4. Task Scheduling Mechanism Based on Reinforcement Learning in Cloud Computing (QMTSF)

Yugui Wang et.al., proposed QMTSF work in 2023, to incorporate the application of Reinforcement Learning in Task Scheduling mechanism to balance computational loads in cloud servers. QMTSF introduces a Q-learning-based Multi-Task Scheduling Framework (QMTSF). The framework operates in two stages: initially, tasks are dynamically assigned to appropriate servers in the cloud environment based on server types. Subsequently, an enhanced Q-learning algorithm, known as UCB-based Q-Reinforcement Learning (UQRL), is employed on each server to allocate tasks to Virtual Machines. The agent makes decisions based on previous experiences and interactions with the environment, learning from rewards and penalties to develop an optimal task allocation strategy and schedule tasks across different VMs. Simulation experiments were conducted to assess the performance of the proposed framework against traditional scheduling methods such as Particle Swarm Optimization, random scheduling, and Round-Robin (RR). Results indicate that the QMTSF scheduling framework surpasses other methods in terms of makespan and average task processing time.
To evaluate the performance of the proposed QMTSF, task scheduling was simulated using the CloudSim platform. The simulation covered scenarios with varying numbers of tasks—100, 300, 500, 800, and 1000—each with randomly generated lengths ranging from 500 to 3000. The cloud environment included four servers, representing different task categories, each equipped with five Virtual Machines. To ensure the reliability of the results, the final experimental outcomes were based on the average of multiple trials. On average, ten experiments were conducted, and it was observed that increasing the number of trials beyond ten had only a minimal effect on the average results.
The observed advantage is that the QMTSF framework effectively minimizes makespan, ensures task deadlines are met, reduces average task processing time, and improves CPU utilization, leading to lower overall energy consumption in the cloud data center, whereas, the application to scenarios involving dynamic scaling will require further exploration of task classification and mapping, is the disadvantage of QMTSF work.

2.5. CSO-ILB: chicken swarm optimized inter-cloud load balancer for elastic containerized multi-cloud environment (CSO-ILB)
Saif, M.A.N., et.al., established CSO-LIB work in 2022 for the purpose of work load balancing in multi-cloud environments. An autonomic load balancer is designed in CSO-ILB work to enhance the elasticity of cloud systems and distribute user workloads among available containers in a multi-cloud environment. The approach utilizes the concept of multi-loop to facilitate efficient self-management prior to load balancing. Tasks are allocated to containers using an advanced scheduling algorithm known as Deadline-Constrained Make-span Minimization for Multi-Task Scheduling (DCMM-MTS). The proposed method was assessed using the Container CloudSim platform and compared with existing meta-heuristic algorithms, including Ant Colony Optimization, Bee Colony Optimization, Shuffled Frog Leaping Algorithm, and Cat Swarm Optimization (CSO). Simulation results demonstrated that the proposed method excelled in terms of reliability, CPU utilization, makespan, energy consumption, response time, execution cost, idle time, and task migration.
CSO-ILB technique significantly enhances performance by improving CPU utilization, reducing makespan, response time, execution cost, and energy consumption, while also increasing reliability and minimizing idle time and task migrations is the advantage of CSO-ILB work. The effectiveness of the CSO-ILBmethodology is evaluated using the ContainerCloudsim toolkit in a multi-cloud containerized environment, which may limit the generalizability of the results to different or larger-scale cloud scenarios is the identified limitation of CSO-ILB work.
A summary about the discussed methods are precisely highlighted in Table 1.

Author	**Work**	**Year**	**Methodology**	**Advantages**	**Limitations**
M. S. Al Reshan et al.,	A Fast Converging and Globally Optimized Approach for Load Balancing in Cloud Computing	2023	GWO, PSO	Reduced response time, increased throughput	Lesser Degree of Balance

P. Udayasankaran et.al.,	Energy efficient resource utilization and load balancing in virtual machines using prediction algorithms	2023	Dedicated energy prediction & resource allocation	Reduced Energy consumption, processing time	Migration Cost
Adewojo. A.A. et al.,	A novel weight-assignment load balancing algorithm for cloud applications	2023	Weight assigment based Load balancing	Improved response time	lack of dynamic cloud applicability
Yugui Wang et.al.,	Task Scheduling Mechanism Based on Reinforcement Learning in Cloud Computing	2023	Q-Reinforcement Learning	Improved response time	lack of dynamic cloud applicability
Saif, M.A.N., et.al.,	CSO-ILB: chicken swarm optimized inter-cloud load balancer for elastic containerized multi-cloud environment	2022	CSO, ILB	Enhanced performance	not suitable for large-scale clouds

Table 1: Existing methods' key points

3. Backgrounds

Linear Regression (LR) and Echo State Network (ESN) are the two basic building blocks used to construct the Linear Regression and Recurrent Neural Network Ensemble for Swift Tensile Load balancer for Static and Dynamic Clouds work. A crisp introduction to these technical strategies is required to explain the proposed LRESTL work more vividly.

3.1. Linear Regression (LR)

LR is a statistical method used to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. The main objective of LR is to find the best-fitting straight line through the data points, which can be used for predictive analysis and understanding the influence of different variables on the target outcome. In general, LR is divided into 2 categories such as Simple Linear Regression in which thereinvolves a single independent variable to predict the dependent variable, and Multiple Linear Regression that could have two or more independent variables to predict the dependent variable.

LR relies on several key assumptions to ensure the validity of its results. Firstly, it assumes that there is a linear relationship between the independent and dependent variables, meaning that changes in the predictors lead to proportional changes in the outcome. Secondly, the residuals (or errors) are expected to be normally distributed, which helps in making accurate predictions and reliable statistical inferences. Additionally, there should be no or little multicollinearity between the independent variables, ensuring that each predictor provides unique information [21]. Homoscedasticity is also assumed, indicating that the variance of the residuals remains constant across all levels of the independent variables. Finally, the observations should be independent of each other, meaning that the value of one observation does not influence or relate to another.

The LR equation to compute the output variable based on the explanatory independent variables is given below.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon \quad \text{Equation (1)}$$

where y is the dependent output variable, β_0 is the intercept, $\beta_1, \beta_2, \ldots \beta_n$ are the corresponding coefficients of independent input variables $x_1, x_2, \ldots x_n$, and ϵ represents the error term.

LR is often favored for its simplicity and interpretability, making it accessible for understanding the relationship between variables. It provides clear insights into how changes in independent variables affect the dependent variable through straightforward linear relationships. LR also requires fewer computational resources compared to more complex models, which is advantageous for both small and

large datasets. Its effectiveness is particularly notable when the assumptions of linearity, normality, and independence are met, leading to reliable and actionable predictions [22].
Linear Regression is widely used in various fields such as finance, economics, and biology for tasks like forecasting, risk assessment, and understanding relationships between variables In LRESTL work, a customized version of LR is derived to predict the computational and communication load based on the input covariates.

3.2. Echo State Network (ESN)
Echo State Networks are a type of recurrent neural network designed to efficiently handle time-series and sequence data. They utilize a fixed, randomly connected reservoir of neurons to generate dynamic state representations from input data, which simplifies the training process by only requiring adjustments to the output weights. This design leverages the echo state property, where the influence of inputs on the reservoir state eventually diminishes, ensuring manageable memory. ESNs are particularly useful for tasks involving complex temporal dependencies, such as time-series forecasting and speech recognition, due to their ability to capture intricate patterns in sequential data [23].The equations involved in ESN are given below.
Let t be the variable that refers the time, $x(t)$ is the state vector at time t, $x(t+1)$ be the state vector at time $t+1$, w_{in} is the input weight matrix, $u(t)$ is the input vector at time t, w_{res} is the fixed reservoir weight matrix, and b_{res} is the reservoir bios term, then the State update Equation of ESN will be as follows.

$$x(t+1) = \tanh(w_{in}u(t) + w_{res}x(t) + b_{res}) \quad \text{Equation (2)}$$

The training phase equation which is used to compute output weight w_{out} is given below.

$$w_{out} = YX^{\dagger} \quad \text{Equation (3)}$$

where Y is the target output matrix, X is the matrix of reservoir states, and $\dagger$ refers the pseudo inverse
The output Equation is

$$y(t) = w_{out}x(t) + b_{out} \quad \text{Equation (4)}$$

where $y(t)$ is the output vector, w_{out} is the trained output weight matrix, and b_{out} is the output bias coefficient

These equations illustrate the primary operations within Echo State Networks (ESNs). They detail how the internal state of the reservoir is continuously updated by incorporating both the current inputs and the previous state of the reservoir. Additionally, the equations describe how the state of the reservoir is translated into outputs through a process that involves adjusting and applying trained weights. This approach effectively combines the dynamic behavior of the reservoir with a linear mapping to generate the final output. ESN offer several notable advantages in computational modeling and time-series prediction [24]. Their primary strength lies in their ability to handle complex temporal patterns due to the reservoir's dynamic, nonlinear processing capabilities. Unlike traditional recurrent neural networks,ESNs do not require extensive training of the reservoir weights, which simplifies and speeds up the learning process.

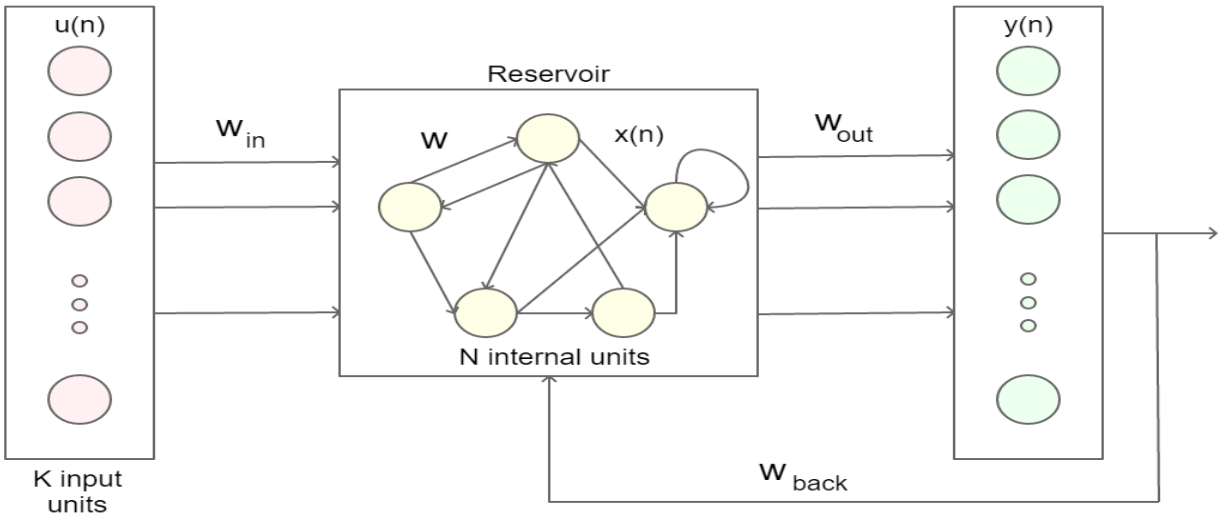

Figure 1: Echo State Network Architecture

The fixed, randomly initialized reservoir allows for efficient training as only the output weights need to be adjusted, reducing computational demands. Additionally, ESNs can effectively capture long-term dependencies and are highly adaptable to various types of sequential data, making them a versatile tool for tasks ranging from forecasting to signal processing.

4. Proposed Method

The LRESTL work is composed of three operational blocks namely Tailored Elastic Net Load Prognosticator (TENLP), Custom Echo State RNN Load Tracker (CESRLT), and Swift Tensile Consolidator (STC). A detailed depiction is provided in this section.

4.1. Tailored Elastic Net Load Prognosticator (TENLP)

The TENLP module is designed to prognosticate the communication load level based on the current input parameters in a cloud computing environment. The elastic net is used in TENLP module to overcome the overfitting issue of conventional LR. TENLP uses most important network communication overhead parameters such as Data size, Bandwidth, Latency, Number of requests, Number of Virtual Machines, Communication protocol overhead, Packet delivery ratio, and security overheads. Data size refers to the amount of data being transferred between cloud components such as client to server, server to server, or between different microservices.The data size is one of the directly proportional input parameters to the communication load. Bandwidth is the available network transmission capacity between the communicating entities. Higher bandwidth can reduce the communication load by allowing faster data transfer rates. Bandwidth is measured in Mbps units. Latency the time delay between sending and receiving data. High latency can increase the perceived communication load as it slows down data transmission. Latency is usually measured in milliseconds (mS).

Number of requests refers to the total volume of interactions being sent and received. A higher volume can increase the communication load, especially if each interaction carries significant data.The term Number of Nodes or Virtual Machines refers to the count of servers, nodes, or virtual machines participating in communication. An increase in the number of these communicating entities can lead to a higher load on the network.Communication Protocol Overhead refers to the inherent overhead associated with protocols such as HTTP, TCP/IP, and others. This overhead includes elements like headers and control messages. Such overhead can contribute to the overall communication load, particularly for protocols that necessitate frequent acknowledgments or handshakes. The communication protocol overhead is measured in Bytes in general. the packet delivery rate (PDR) is a metric that measures the percentage of data packets successfully delivered from a source to a destination over a network or between virtual machines. It is an indicator of network performance and reliability. PDR is used to measure in percentage (%). Security Overhead refers to the additional data load resulting from encryption, decryption, and security protocols. Measures such as VPNs, SSL/TLS, or IPSec can introduce extra data and processing time, thereby increasing the overall communication load. The security overhead is measured with a memory-processing time duo. The input parameters are stored and handled as a Set

initialized as $\Gamma = \{\gamma_d, \gamma_b, \gamma_l, \gamma_r, \gamma_v, \gamma_c, \gamma_p, \gamma_s\}$ where γ_d refers the data size, γ_b refers the bandwidth, γ_l refers the latency, γ_r refers the number of requests, γ_v refers the number of virtual machines, γ_c refers the communication overhead protocol, γ_p refers the packet delivery ratio, and γ_s refers the security overhead,

The security overhead can be calculated by normalizing and computing the average of memory and processing time in % as in the following equation

$$\gamma_s = \frac{1}{2}\left(\left(\frac{s_d}{s_{d_{max}}} \times 100\right) + \left(\frac{s_t}{s_{t_{max}}} \times 100\right)\right) \quad \text{Equation (5)}$$

where s_d refers the memory overhead, and s_t refers the security processing time overhead.

Then for any element $\gamma_x \in \Gamma$, the normalized proportional value $\gamma_x^\dagger$ can be calculated as follows.

$$\gamma_x^\dagger = \frac{-e^{i\pi}\gamma_x}{\gamma_{x_{max}}} \quad \text{Equation (6)}$$

The parameters which are directly proportional to the overhead, meaning that as they increase, so does the overhead. These include Data Size, as larger data packets require more transmission resources; Number of Requests, since more requests lead to higher transaction and coordination overhead; Number of Virtual Machines, where additional VMs can increase network traffic and coordination complexity; Communication Protocol Overhead, which adds extra data and processing demands from protocols; and Security Overhead, which includes extra data and processing time required for encryption and other security measures.

Conversely, some parameters are inversely proportional to communication overhead, meaning that as they increase, the overhead typically decreases. These include Bandwidth, where higher bandwidth allows for faster and more efficient data transmission; Latency, as lower latency results in quicker transmission and reduced communication delays; and Packet Delivery Ratio, where a higher ratio indicates fewer lost or retransmitted packets, thereby lowering overall overhead. Understanding these relationships is crucial for optimizing network performance and managing communication efficiency. The direct, and inversely proportional parameters are given in Figure 2.

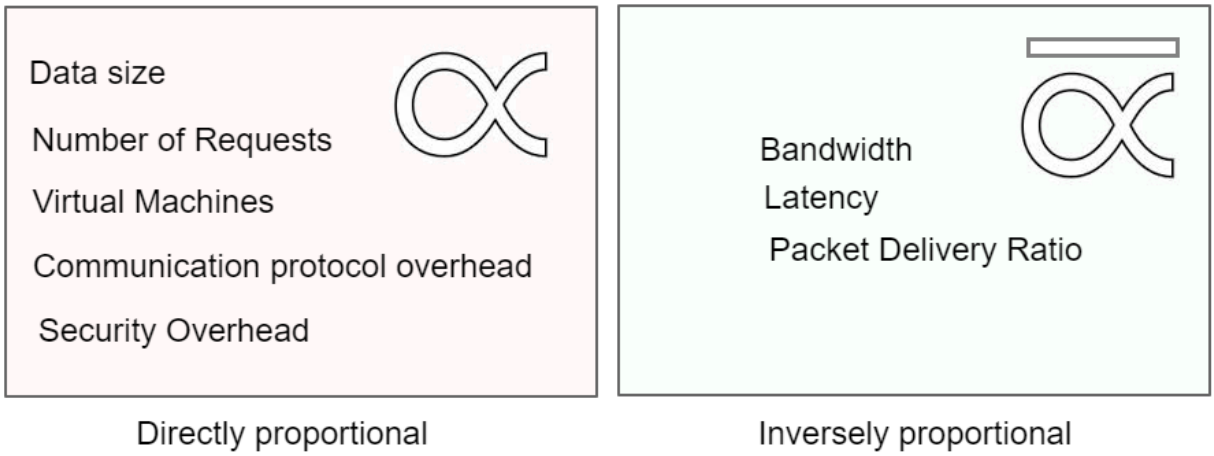

Figure 2: Parameters and their proportionality with communication load

Putting together the directly proportional and inversely proportional variables into a single LR equation will complicate the calculation of intercept β_0 and the slope coefficients $\beta_1, \beta_2 \ldots \beta_n$. Thus, TENLP method is equipped with a dual LR equations derived for simpler and faster parameter handling. Let δ be the primitive prediction variable for directly proportional parameters, the value of δ is calculated as follows.

$$\delta = \beta_0 + \beta_1\gamma_a^\dagger + \beta_2\gamma_r^\dagger + \beta_3\gamma_v^\dagger + \beta_4\gamma_c^\dagger + \beta_5\gamma_s^\dagger \quad \text{Equation (7)}$$

Similarly, let δ' be the primitive prediction variable for inversely proportional parameters, which can be calculated by the equation given below.

$$\delta' = \beta_0 + \beta_1\frac{1}{\gamma_b^\dagger} + \beta_2\frac{1}{\gamma_l^\dagger} + \beta_3\frac{1}{\gamma_p^\dagger} \quad \text{Equation (8)}$$

Then the prediction variable Δ is calculated using equation 9

$$\Delta = \frac{1}{8} \times (\delta + \delta') \quad \text{Equation (9)}$$

Then the communication load prediction T_P is achieved by equation 10

$$T_P = \begin{cases} \text{Low if } \Delta < 1/3 \\ \text{Medium if } 1/3 \leq \Delta \leq 1/6 \\ \text{High otherwise} \end{cases}$$ Equation (10)

As defined by the TENLP procedure the communication load prediction T_P is computed based on the input environmental parameters set Γ.

4.2. Custom Echo State RNN Load Tracker (CESRLT)

As the communication load is handled by the TENLP module, the CESRLT modules is crafted to rack the computational loads. There are several measurement tools such as CloudWatch, Azure Monitor, StackDriver, Prometheus, Grafana, Nagios, Datadog, New Relic, Zabix, Kibana, Dynatrace, and Splunk are in use to measure the current computational loads in a typical cloud computing environment. These tools offer extensive capabilities for monitoring and managing computational loads in cloud environments. They support optimal performance, efficient resource distribution, and system reliability by gathering, analyzing, and displaying data on different metrics associated with computational loads. Using these tools, cloud administrators can identify irregularities, anticipate load fluctuations, and make informed choices regarding scaling, resource optimization, and enhancing system performance.

Metrics such as CPU usage, memory consumption, network bandwidth, disk I/O operations, and storage utilization are systematically collected through either built-in monitoring agents or external monitoring tools. These metrics offer a detailed view of how effectively cloud resources are being utilized. By analyzing these data points, administrators can gain a comprehensive understanding of resource performance and usage efficiency, helping them make informed decisions regarding resource allocation, system optimization, and overall cloud infrastructure management.

Primary computational load decision making parameters such as Processor Utilization ξ_p, Memory consumption ξ_m, Storage operations ξ_s, Task queue length ξ_{tql}, Task arrival rate ξ_{tar}, Task execution time ξ_{tet}, Concurrency level ξ_c, Error rate ξ_e, and power consumption ξ_{pwr} are taken into consideration by the CESRLT module. The input parameters are represented by a set $\Xi = \{\xi_p, \xi_m, \xi_s, \xi_{tql}, \xi_{tar}, \xi_{tet}, \xi_c, \xi_e, \xi_p\}$ for the normalization process as in Equation 11 which is constructed in a way similar to Equation 6.

$$\xi_x^{\dagger} = \frac{-e^{i\pi}\xi_x}{\xi_{x_{max}}}$$ Equation (11)

A customized Echo State RNN architecture is designed for CESRLT module as illustrated in Figure 3.

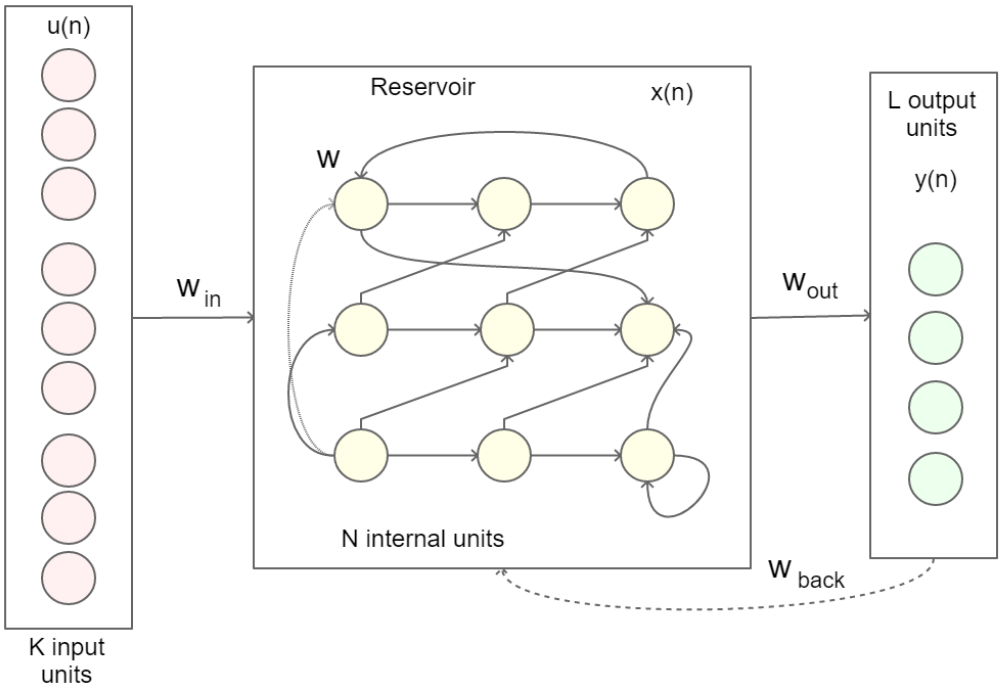

Figure 3: CESRLT ESN Architecture

There are 4 labels such as “Low”, “Moderate”, “High” and “Overload” are used to represent the computation load based on the current environmental input variables. The 9 normalized parameters $\xi_x^\dagger \in \Xi$ are given as the input to the CESRLT echo state network to get the corresponding computational load classification labelC_P.

4.3. Swift Tensile Consolidator (STC)

STC receives both the inputs T_p and C_p from TENLP and CESRLT modules respectively to allocate or to revoke computational resources in the cloud computing environment. The key points used in STC are, whenever there is a spike in the communication load, it can be reduced by some core functions such as data aggregation. At the same time, data aggregation will increase the computational complexity as well. Therefore, maintaining an excellent resonance between these loads is the key to achieve to get best performance – which is achieved through STC module. The data aggregation processes such as data filtering and data compression are utilized in STC module.

Reducing communication overhead through data aggregation in cloud computing is an effective strategy to enhance efficiency, minimize bandwidth usage, and optimize overall system performance. Set up a layered system where middle-tier nodes or servers first collect and combine data before forwarding it to the main server or cloud. This approach decreases the number of separate data transmissions and cuts down on the amount of data sent across the network.Utilize filtering methods that ensure only important or relevant data is transmitted, discarding redundant, non-essential, or low-priority information to reduce communication load. Implement threshold-based systems that trigger data transmission only when values surpass specific limits or show significant changes. This approach helps to minimize unnecessary data flow and optimize network efficiency.

Use of data compression techniques to shrink the size of data packets before they are transmitted could significantly reduce the communication costs. By compressing data, the amount of bandwidth required is reduced, which can effectively lower communication overhead. Methods like lossless compression are especially useful because they maintain the integrity of the original data, ensuring that no information is lost during the compression and decompression process.

Offloading concept is used in STC module to reduce the computational load accumulated in a single server. Transferring tasks to local servers nearer to the data source can significantly lower latency compared to central cloud data centers. This approach reduces the time required for data transmission and processing, resulting in quicker response times for users and applications. Distributing computational work across various resources alleviates the strain on cloud data centers, enabling them to concentrate on more complex and data-heavy tasks. This strategy optimizes performance and helps prevent bottlenecks

by effectively balancing the load. A dedicated algorithm is used to handle the computational and communication load balancing in STC module.

Algorithm: STC
Input: T_P and C_P
Output: Balanced process distribution
Step 1: Read T_P from TENLP
Step 2: Read C_P from CESRLT'
Step 3: If T_P = "Low"
Step 4: if C_P = "Low" Revoke resources
Step 5: else if C_P = "Moderate" retain resources
Step 6: else if C_P = "High" Initiate offloading
Step 7: else if C_P = "Overload" initiate offloading and allocate more resource
Step 8: end if // C_P
Step 9. If T_P = "Medium"
Step 10: if C_P = "Low" Initiate Aggregation (Data filtering, Data compression)
Step 11: if C_P = "Moderate" Initiate Aggregation (Data compression)
Step 12: if C_P = "High:" Initiate offloading, and Initiate Aggregation (Data filtering)
Step 13: if C_P = "Overload" Initiate offloading and Allocate more resource
Step 14: end if // C_P
Step 15: If T_P = High
Step 16: if C_P = "Low" Initiate Aggregation (Data filtering, Data Compression)
Step 17: if C_P = "Moderate" Initiate offloading, Initiate Aggregation (Data filtering, Data _ compression)
Step 18: if C_P = "High" Initiate offloading, Initiate Aggregation (Data filtering)
Step 19: if(C_P = "Overload" Initiate offloading, Allocate more Resource
Step 20: Continue

The efficient balancing between the network resources and computational resources by performing data filtering and data compression in STC algorithm improves the stability of the static clouds. In addition, the resource revoke and resource allocation processes provided in STC algorithm improves the overall load balancing efficiency in dynamic cloud environments.

5. Experimental Setup:

A computer equipped with an Intel i7 4.7 GHz processor, 16GB of RAM, and a 1TB SSD is utilized to develop the entire model. A dedicated User Interface (UI) is crafted using the Visual Studio IDE [25], and the methodology is implemented using the C++ 20.0 [26] programming language. The UI is designed to facilitate the uploading of resource management scripts that can interface with the cloud server's hypervisor. To assess the performance of the proposed method, a dedicated server is leased from i2k2 cloud service providers [27]. Communication between the UI and the server is carried out via the Common Cloud Gateway Interface, utilizing registered credentials provided by the service provider.The user interface uploads the resource allocation schema directly to the hypervisor of the dedicated server, and resource utilization data is retrieved from the server to generate log report files and create graphs. Performance metrics are recorded while the dedicated server processes real-time data shared by various sites such as Dropbox and Foxrenderfarm. Websites like dropbox.com generate higher demands for storage and data transfer, while sites like foxrenderfarm.com require more intensive processing tasks. This combination of data sources from these websites offers a broad spectrum of computational resource requirements, which are used to conduct experiments more effectively and authentically.

6. RESULTS AND DISCUSSIONS

Benchmark metrics for cloud resource management, including Resource Utilization, Degree of Balance, Average Response Time, Migration Cost, Throughput across varying numbers of tasks, and Throughput for different counts of Virtual Machines, are recorded during the experiments. A range of 100 to 1000 tasks is executed across 2 to 20 virtual machines, and the results are discussed in this section.

6.1. Resource Utilization

Resource utilization serves as a key benchmark in cloud computing due to its crucial impact on optimizing the efficiency, cost-effectiveness, and overall performance of cloud services and infrastructure. To measure resource utilization, data on various system resources and components, such as CPU, memory, storage, and network bandwidth, are monitored and analyzed to gauge their usage. Cloud providers typically offer built-in monitoring and management tools that track resource utilization. The resource utilization metrics for current and proposed methods are measured and outlined in the following table 2, and the comparison graph is provided in Figure 4.

Resource Utilization (%) / Number of Tasks						
Number of Tasks	**GWOPSO**	**EPRUA**	**NWALBA**	**QMTSF**	**CSOILB**	**LRESTL**
100	33.22564	29.02564	26.52564	35.07692	27.47692	36.22564
200	50.73666	48.89362	45.8172	50.65356	46.74284	54.8293
300	60.92432	60.59257	57.0614	59.83595	57.93576	65.78866
400	68.22205	68.83852	65.05748	66.33276	66.0344	73.43295
500	73.84283	75.20895	71.33665	71.34644	72.18536	79.42199
600	78.35841	80.40926	76.3786	75.54079	77.25296	84.28975
700	82.21568	84.77647	80.61192	78.99638	81.48628	88.47834
800	85.6305	88.60394	84.3234	82.06324	85.19776	92.11353
900	88.57171	92.08257	87.64845	84.72318	88.57408	95.22347
1000	91.2	95.02564	90.52564	87.07692	91.47692	98.10256

Table 2: Resource Utilization (%) / Number of Tasks

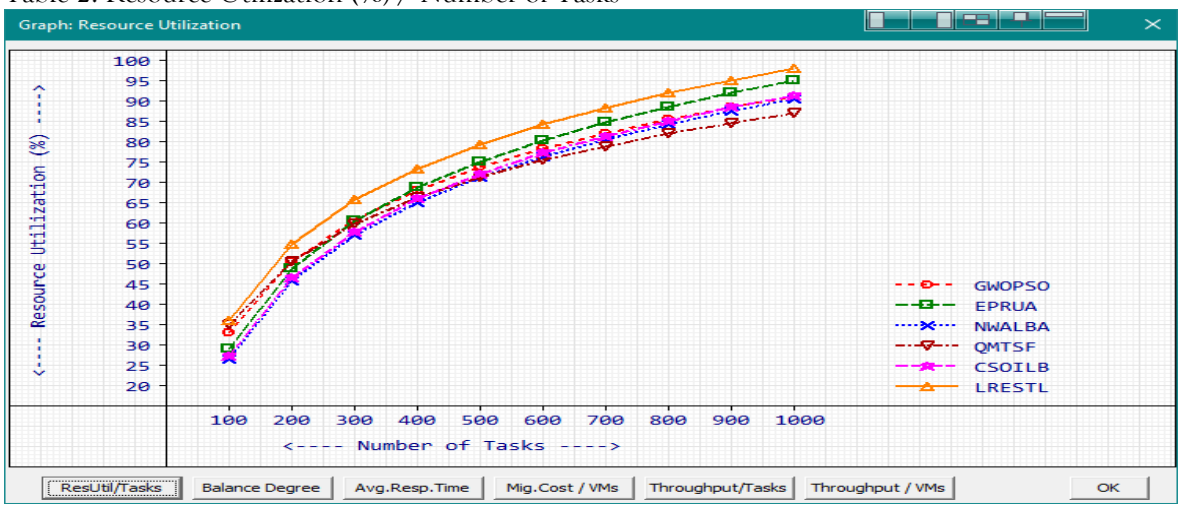

Figure 4: Resource Utilization (%) / Number of Tasks

It is observed that, as the number of tasks increases, resource utilization also rises for all methods. Among the algorithms, LRESTL consistently shows the highest resource utilization, reaching up to 98.10% at 1000 tasks. QMTSF and CSOILB also demonstrate significant resource utilization but slightly lower than LRESTL. In contrast, NWALBA and EPRUA generally report lower utilization percentages compared to other methods, with GWOPSO performing moderately well. This data suggests that LRESTL is the most effective in maximizing resource utilization across varying task loads.

6.2. Degree of Balance

The Degree of Balance parameter represents a combination of various cloud computing factors, including load balancing, resource balancing, auto-scaling, resource optimization, and availability index. It is calculated using the following formula.

$$\text{Degree of Balance} = \frac{\frac{1}{n}\sum_{i=1}^{n} d_{l_i} + \frac{1}{n}\sum_{i=1}^{n} d_{r_i} + \frac{1}{n}\sum_{i=1}^{n} d_{a_i} + \frac{1}{n}\sum_{i=1}^{n} d_{o_i} + \frac{1}{n}\sum_{i=1}^{n} d_{v_i}}{5}$$

where d_{l_i} refers the Load balancing index, d_{r_i} is the Resource balancing index, d_{a_i} is Auto-scaling index, d_{o_i} is the optimization index, d_{v_i} is the Availability index, and n is the number of tasks performed by the virtual machine.

The performance of a cloud computing environment is directly related to its degree of balance. A higher degree of balance signifies better performance of the computational system. The calculated Degree of Balance values for the methods discussed are provided in the following table.

Balance Degree (%) / Number of Tasks						
Number of Tasks	**GWOPSO**	**EPRUA**	**NWALBA**	**QMTSF**	**CSOILB**	**LRESTL**
100	78.21898	76.05129	76.02564	82.90257	75.95128	81.02564
200	82.58844	81.22008	80.21442	85.25797	80.75884	85.50207
300	85.37579	84.21362	82.6797	86.73206	83.46416	88.11625
400	87.24988	86.23502	84.48013	87.74157	85.43819	89.95284
500	88.48455	87.90813	85.81122	88.44768	87.02065	91.3005
600	89.74526	89.25421	86.89412	89.11311	88.27171	92.51575
700	90.74947	90.36667	87.83138	89.6284	89.35511	93.47422
800	91.69234	91.35253	88.6689	90.1226	90.2201	94.35235
900	92.38663	92.22212	89.43632	90.61285	91.0796	95.05301
1000	93.29197	93.05129	90.02564	90.93128	91.77692	95.7

Table 3: Balance Degree (%) / Number of Tasks

Based on the observed results, it is identified that the method LRESTL consistently achieves the highest balance degree, reaching up to 95.7% with 1000 tasks, indicating superior load balancing capabilities. QMTSF and CSOILB also demonstrate high balance degrees, but slightly lower than LRESTL, suggesting effective load distribution. EPRUA, GWOPSO, and NWALBA show moderate balance degrees, with NWALBA having the lowest balance degree among the methods. Overall, LRESTL emerges as the most effective method for maintaining a balanced distribution of computational load, ensuring optimal performance as task numbers grow.

A comparison graph is given below as Figure 5.

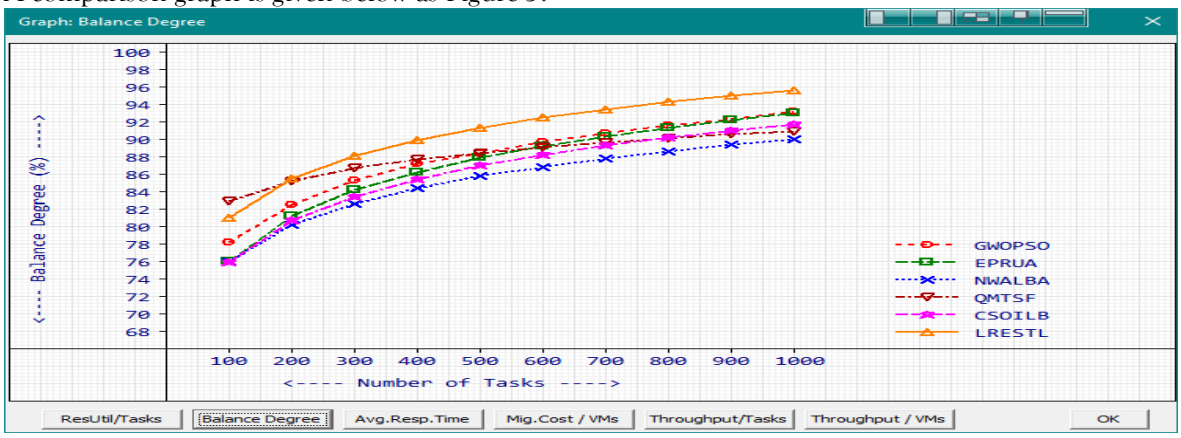

Figure 5: Balance Degree (%) / Number of Tasks

6.3. Average Response Time

Average response time is a crucial metric in cloud computing as it impacts user experience, cost-effectiveness, scalability, adherence to SLAs, competitive positioning, and the overall performance and success of cloud-based applications and services. Monitoring and improving response times are critical for businesses and organizations using cloud computing to meet their objectives. A shorter average response time correlates with higher service quality.

Common tools for measuring response times in cloud computing include application performance monitoring (APM) solutions, log analysis tools, and monitoring services specific to cloud providers. A desktop client specific to the cloud resource monitor provided by the service provider is used to measure the average response time. The standard unit for measuring average response time is milliseconds (mS). The average response time for the discussed methods was measured during experiments for every 100 tasks in the range from 100 to 1000, with the results recorded in the table 4, and a comparison graph is plotted and provided as Figure 6

Average Response Time (mS) / Number of Tasks						
Number of Tasks	**GWOPSO**	**EPRUA**	**NWALBA**	**QMTSF**	**CSOILB**	**LRESTL**
100	359	378	461	413	383	337

200	473	474	573	544	491	427
300	624	617	735	722	658	570
400	836	815	938	943	866	747
500	1092	1047	1197	1214	1119	979
600	1379	1343	1506	1540	1409	1261
700	1745	1657	1847	1908	1761	1584
800	2122	2058	2240	2318	2172	1984
900	2583	2469	2701	2816	2635	2385
1000	3077	2953	3184	3319	3122	2868

Table 4: Average Response Time (mS) / Number of Tasks

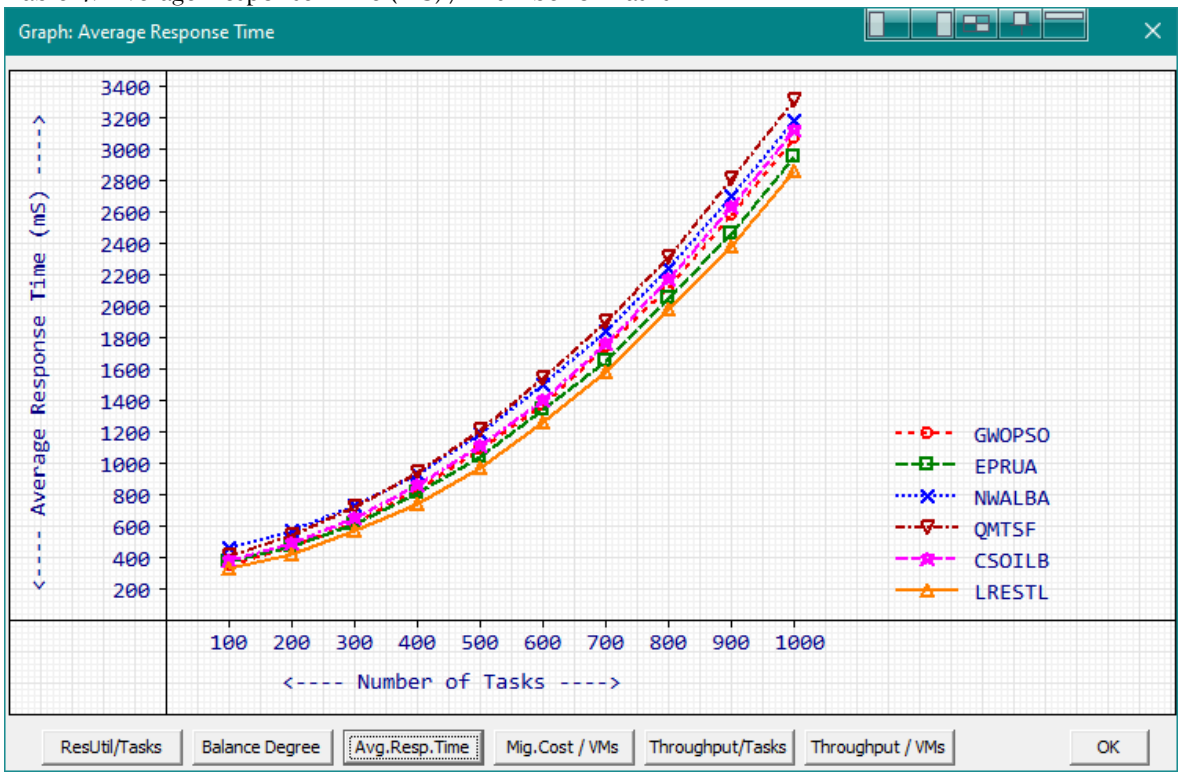

Figure 6: Average Response Time (mS) / Number of Tasks

The performance evaluation of average response times reveals notable differences among various methods. LRESTL consistently exhibits the lowest response times, indicating superior performance. For instance, with 100 tasks, LRESTL achieves a response time of 337 milliseconds, showing about a 6% improvement over the next best method, GWOPSO, which records 359 milliseconds. By the time the number of tasks reaches 1000, LRESTL maintains its edge with a response time of 2868 milliseconds, approximately 7% better than GWOPSO's 3077 milliseconds. Conversely, QMTSF and CSOILB present higher response times, with QMTSF recording 413 milliseconds at 100 tasks and 3319 milliseconds at 1000 tasks, and CSOILB showing 383 milliseconds and 3122 milliseconds, respectively. The EPRUA and NWALBA methods have the highest response times, with EPRUA displaying 378 milliseconds at 100 tasks and 2953 milliseconds at 1000 tasks, about 7% higher than LRESTL. Overall, LRESTL demonstrates the best performance with significant improvements over the other methods, while EPRUA and NWALBA lag with the highest response times.

6.4. Migration Cost

Migration cost is a critical factor in cloud computing, influencing budgeting, strategic decision-making, risk assessment, resource distribution, and the ability to achieve cloud-related advantages. Tools for cost tracking and management, offered by cloud service providers, are utilized to monitor and evaluate expenses during and after the migration process. Migration costs are tracked for a range of 2 to 20 virtual machines, with the recorded costs detailed in the table 5.

Migration Cost (%) / Number of VMs						
Number of VMs	**GWOPSO**	**EPRUA**	**NWALBA**	**QMTSF**	**CSOILB**	**LRESTL**

2	10.22564	16	15.02564	9.702564	17	9.051282
4	14.49134	18.57101	20.22007	16.876	22.19443	11.70927
6	16.93098	19.98932	23.21363	21.15348	25.18798	13.31973
8	18.62884	20.93689	25.31194	24.10072	27.26066	14.46982
10	20.0625	21.73155	26.95941	26.40092	28.98505	15.29073
12	21.19668	22.45776	28.27985	28.3782	30.33114	16.05464
14	22.08266	22.95544	29.46923	29.90799	31.39231	16.65716
16	22.89454	23.45662	30.37817	31.37672	32.45509	17.17909
18	23.63632	23.82479	31.22212	32.5531	33.22212	17.61382
20	24.30257	24.27692	32.10257	33.70256	34.07692	18.02564

Table 5: Migration Cost (%) / Number of VMs

The evaluation of migration costs reveals significant differences across the methods when considering varying numbers of virtual machines (VMs). LRESTL consistently demonstrates the lowest migration costs, reflecting efficient resource management. For instance, with 2 VMs, LRESTL incurs a migration cost of 9.05%, which is about 11% lower than GWOPSO's 10.23%. At 20 VMs, LRESTL maintains its advantage with a cost of 18.03%, approximately 26% lower than GWOPSO's 24.30%.

Conversely, EPRUA and NWALBA show higher migration costs, with EPRUA costing 16% and 24.28% at 2 and 20 VMs respectively, and NWALBA showing 15.03% and 32.10%. QMTSF and CSOILB also present higher costs, with QMTSF recording 9.70% at 2 VMs and 33.70% at 20 VMs, and CSOILB showing 17% and 34.08%. Overall, LRESTL provides the most cost-efficient solution, significantly outperforming the other methods in terms of migration cost efficiency across different VM configurations.

A Contrast Diagram for Migration Cost (%) Vs Number of VMs is given in Figure 7.

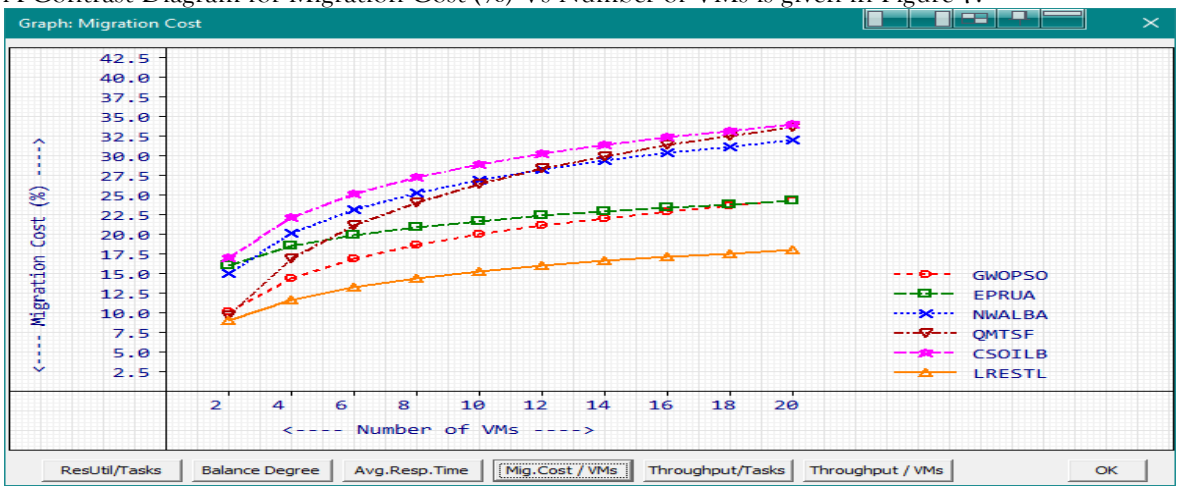

Figure 7: Migration Cost (%) / Number of VMs

6.5. Throughput

Throughput is a crucial metric in cloud computing because it affects user satisfaction, efficient use of resources, scalability, cost-effectiveness, and overall system performance. Ensuring high throughput is vital for the effective delivery of cloud-based services and applications. Throughput is assessed in relation to the number of tasks and the number of virtual machines during the experiments. The findings are shown in Table 6 and Table 7.

Throughput (kB) / Number of Tasks						
Number of Tasks	**GWOPSO**	**EPRUA**	**NWALBA**	**QMTSF**	**CSOILB**	**LRESTL**
100	24	16	18	24	16	23
200	45	40	40	43	35	50
300	68	66	62	63	54	77
400	89	87	84	81	74	99
500	103	109	102	95	90	120
600	118	124	118	108	100	141
700	126	137	128	116	114	148
800	132	144	131	125	114	156

900	133	147	139	126	120	159
1000	139	144	134	127	114	160

Table 6: Throughput (kB) / Number of Tasks

The throughput performance across various methods shows notable differences in data handling efficiency. LRESTL achieves the highest throughput in most scenarios, indicating superior performance in processing tasks. For example, at 100 tasks, LRESTL delivers 23 kB, about 4% more than GWOPSO's 24 kB. At 1000 tasks, LRESTL provides 160 kB, which represents a 15% improvement over GWOPSO's 139 kB. EPRUA and NWALBA also show strong performance but lag behind LRESTL. Specifically, EPRUA delivers 144 kB at 1000 tasks, which is approximately 10% lower than LRESTL. Similarly, QMTSF and CSOILB trail with lower throughputs, with QMTSF recording 127 kB at 1000 tasks, about 20% less than LRESTL. Overall, LRESTL consistently outperforms the other methods in throughput, demonstrating substantial improvements in data processing efficiency across different task volumes.

Throughput (kB) / Number of VMs						
Number of Tasks	**GWOPSO**	**EPRUA**	**NWALBA**	**QMTSF**	**CSOILB**	**LRESTL**
2	17	19	20	24	18	23
4	47	58	48	53	46	63
6	65	82	65	71	62	87
8	77	97	77	83	74	104
10	86	109	85	93	85	115
12	95	121	95	101	91	126
14	103	128	100	107	97	136
16	110	138	106	115	105	144
18	114	145	111	121	109	149
20	118	149	116	122	113	160

Table 7: Throughput (kB) / Number of VMs

The throughput metrics, measured in kilobytes (kB) across varying numbers of virtual machines show that LRESTL consistently outperforms other methods in terms of data handling efficiency. For instance, at 2 VMs, LRESTL achieves 23 kB, which is 35.3% higher than GWOPSO and 27.8% higher than CSOILB. The performance gap widens as the number of VMs increases. At 20 VMs, LRESTL reaches 160 kB, demonstrating a 35.6% improvement over GWOPSO and a 7.4% increase over EPRUA, highlighting its superior throughput capability across the range of tested configurations.

In comparison, EPRUA and CSOILB show notable improvements as well, though not as pronounced as LRESTL. EPRUA increases throughput from 19 kB at 2 VMs to 149 kB at 20 VMs, reflecting a 685.7% improvement. Similarly, CSOILB exhibits a 527.8% increase from 18 kB to 113 kB over the same range. QMTSF and NWALBA also improve but remain lower than the top performers, with QMTSF achieving a peak of 122 kB at 20 VMs, and NWALBA reaching 139 kB, indicating a more moderate improvement in throughput efficiency. In terms of performance metrics, LRESTL leads the rankings, achieving the highest scores, followed by EPRUA which holds the second position, with CSOILB coming in third, QMTSF in fourth, GWOPSO in fifth, and NWALBA ranking sixth.

The side-by-side benchmark graphs are provided in Figure 8 and 9.

International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 6, 2025
https://theaspd.com/index.php

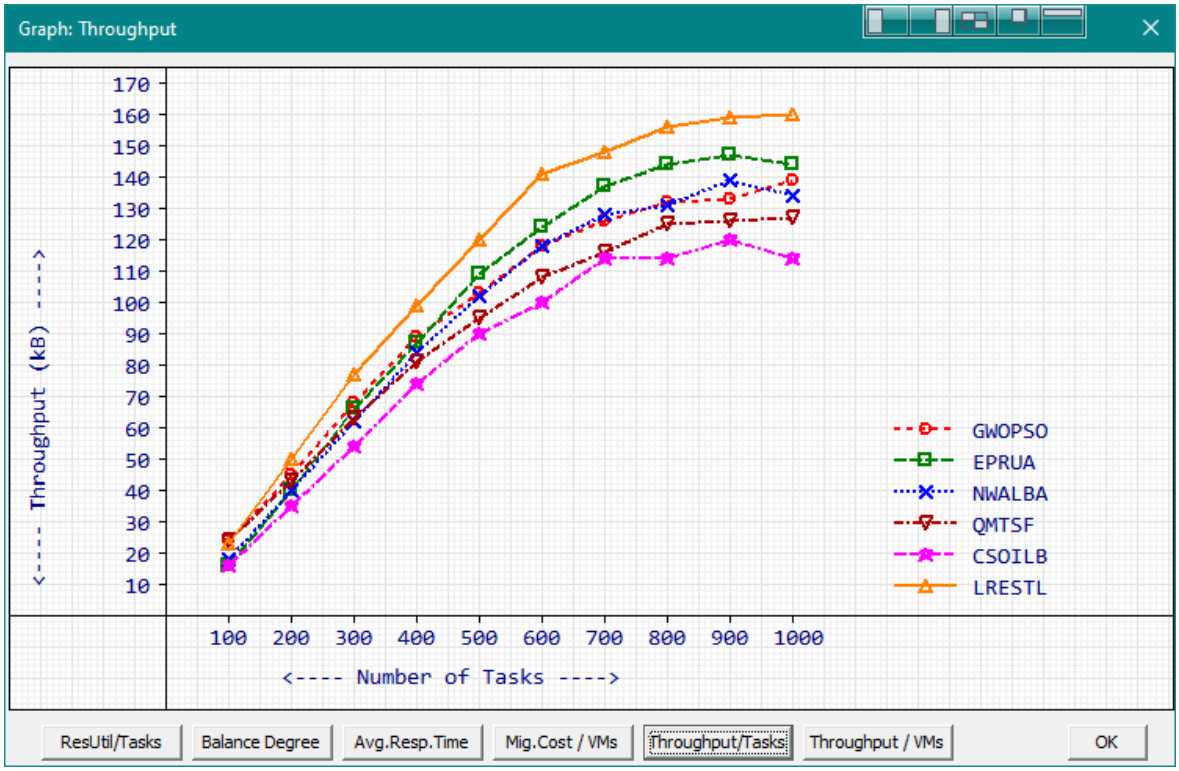

Figure 8: Throughput (kB) / Number of Tasks

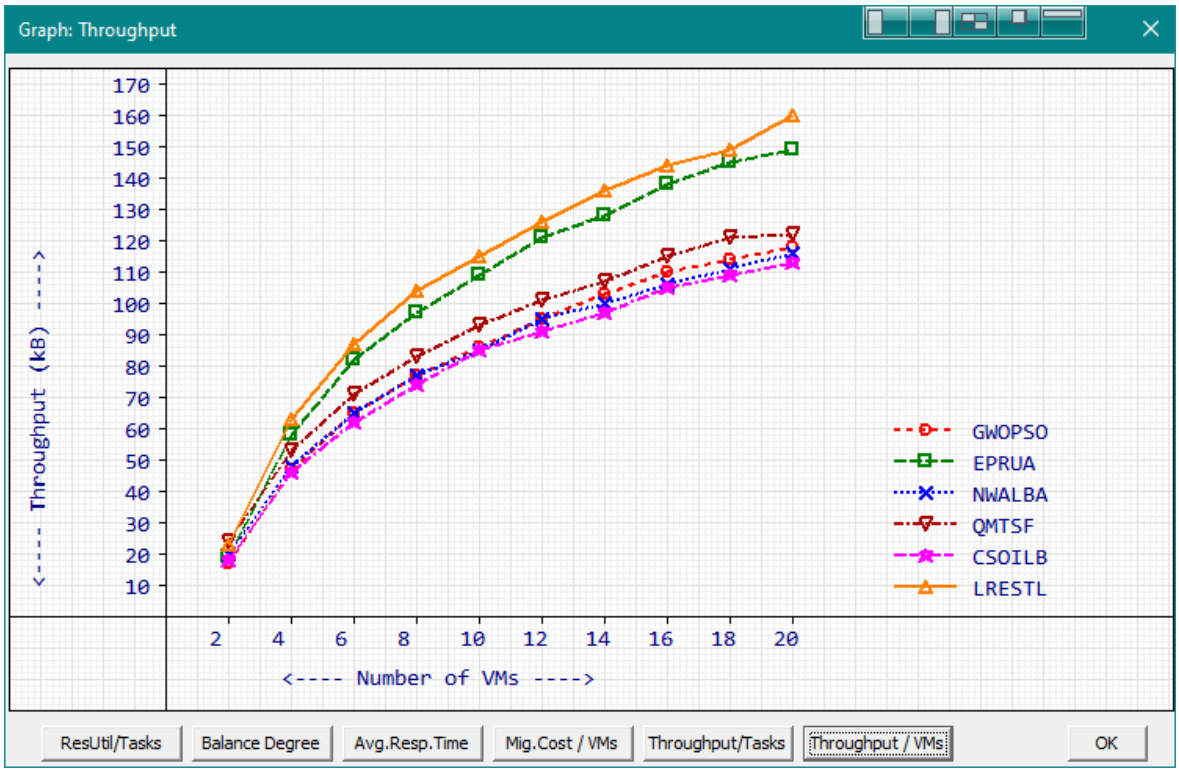

Figure 9: Throughput (kB) / Number of VMs

7. CONCLUSION

In conclusion, the Linear Regression and Recurrent Neural Network Ensemble for Swift Tensile Load Balancer for Static and Dynamic Clouds represents a sophisticated approach to optimizing load balancing in cloud computing environments. By integrating three innovative modules—Tailored Elastic Net Load Prognosticator, Custom Echo State RNN Load Tracker, and Swift Tensile Consolidator—LRESTL

addresses the critical need for effective management of both computational and communication loads. This approach harnesses real-time monitoring and machine learning to dynamically adjust resource allocation, enhancing performance and efficiency across various cloud scenarios.The implementation and evaluation of LRESTL in real-time cloud environments demonstrate its effectiveness in optimizing key benchmark metrics. The system shows significant improvements in resource utilization, balance degree, and throughput, while also reducing average response time and migration costs. By leveraging these advanced techniques, LRESTL provides a robust solution to the challenges posed by fluctuating workloads and diverse cloud requirements, ensuring a balanced, efficient, and responsive cloud infrastructure. A geographical location and communication weight-based optimization in offloading could achieve better performance, which is the anticipated future work.

Conflict of Interest: There is no conflict of interest between the authors in this work.

Dataset availability: Real-time data is used to evaluate the proposed model, thus, no dataset is used in this work.

Code availability: The source code of the entire implementation is shared in GitHub. The link will be provided by the authors on E-Mail request.

REFERENCES:

[1] Hamid K, Iqbal MW, Abbas Q, Arif M, Brezulianu A, Geman O. Cloud Computing Network Empowered by Modern Topological Invariants. Applied Sciences. 2023; 13(3):1399. https://doi.org/10.3390/app13031399

[2] Sukhpal Singh Gill, Huaming Wu, Panos Patros, Carlo Ottaviani, Priyansh Arora, Victor Casamayor Pujol, David Haunschild, Ajith Kumar Parlikad, Oktay Cetinkaya, Hanan Lutfiyya, Vlado Stankovski, Ruidong Li, Yuemin Ding, Junaid Qadir, Ajith Abraham, Soumya K. Ghosh, Houbing Herbert Song, Rizos Sakellariou, Omer Rana, Joel J.P.C. Rodrigues, Salil S. Kanhere, SchahramDustdar, Steve Uhlig, Kotagiri Ramamohanarao, Rajkumar Buyya, Modern computing: Vision and challenges, Telematics and Informatics Reports, Volume 13, 2024, 100116, ISSN 2772-5030, https://doi.org/10.1016/j.teler.2024.100116

[3] Prangon NF, Wu J. AI and Computing Horizons: Cloud and Edge in the Modern Era. Journal of Sensor and Actuator Networks. 2024; 13(4):44. https://doi.org/10.3390/jsan13040044

[4] Zhou, J., Lilhore, U.K., M, P. et al. Comparative analysis of metaheuristic load balancing algorithms for efficient load balancing in cloud computing. J Cloud Comp 12, 85 (2023). https://doi.org/10.1186/s13677-023-00453-3

[5] M. S. Al Reshan et al., "A Fast Converging and Globally Optimized Approach for Load Balancing in Cloud Computing," in IEEE Access, vol. 11, pp. 11390-11404, 2023, https://doi.org/10.1109/ACCESS.2023.3241279

[6] Ajay Jangra, Neeraj Mangla, An efficient load balancing framework for deploying resource schedulingin cloud based communication in healthcare, Measurement: Sensors, Volume 25, 2023, 100584, ISSN 2665-9174, https://doi.org/10.1016/j.measen.2022.100584

[7] Magotra, B., Malhotra, D. & Dogra, A.K. Adaptive Computational Solutions to Energy Efficiency in Cloud Computing Environment Using VM Consolidation. Arch Computat Methods Eng 30, 1789–1818 (2023). https://doi.org/10.1007/s11831-022-09852-2

[8] Y. Zhang, T. Gao, C. Li and C. W. Tan, "Coded Federated Learning for Communication-Efficient Edge Computing: A Survey," in IEEE Open Journal of the Communications Society, vol. 5, pp. 4098-4124, 2024, https://doi.org/10.1109/OJCOMS.2024.3423362

[9] D. Cao, M. Wu, N. Gu, R. S. Sherratt, U. Ghosh and P. K. Sharma, "Joint Optimization of Computation Offloading and Resource Allocation Considering Task Prioritization in ISAC-Assisted Vehicular Network," in IEEE Internet of Things Journal, https://doi.org/10.1109/JIOT.2024.3360962

[10] Shahid MA, Alam MM, Su'ud MM. Performance Evaluation of Load-Balancing Algorithms with Different Service Broker Policies for Cloud Computing. Applied Sciences. 2023; 13(3):1586. https://doi.org/10.3390/app13031586

[11] Shuaib M, Bhatia S, Alam S, Masih RK, Alqahtani N, Basheer S, Alam MS. An Optimized, Dynamic, and Efficient Load-Balancing Framework for Resource Management in the Internet of Things (IoT) Environment. Electronics. 2023; 12(5):1104. https://doi.org/10.3390/electronics12051104

[12] AL-Jumaili AHA, Muniyandi RC, Hasan MK, Paw JKS, Singh MJ. Big Data Analytics Using Cloud Computing Based Frameworks for Power Management Systems: Status, Constraints, and Future Recommendations. Sensors. 2023; 23(6):2952. https://doi.org/10.3390/s23062952

[13] Javad Dogani, Farshad Khunjush, Mehdi Seydali, "Host load prediction in cloud computing with Discrete Wavelet Transformation (DWT) and Bidirectional Gated Recurrent Unit (BiGRU) network," Computer Communications, Volume 198, 2023, Pages 157-174, ISSN 0140-3664, https://doi.org/10.1016/j.comcom.2022.11.018

[14] K. Ramya, SenthilselviAyothi, "Hybrid dingo and whale optimization algorithm-based optimal load balancing for cloud computing environment", in Emerging Telecommunications Technologies, Wiley Online Library, March 2023 https://doi.org/10.1002/ett.4760

[15] Saif, M.A.N., Niranjan, S.K., Murshed, B.A.H. et al. CSO-ILB: chicken swarm optimized inter-cloud load balancer for elastic containerized multi-cloud environment. J Supercomput 79, 1111–1155 (2023). https://doi.org/10.1007/s11227-022-04688-w

[16] M. S. Al Reshan et al., "A Fast Converging and Globally Optimized Approach for Load Balancing in Cloud Computing," in IEEE Access, vol. 11, pp. 11390-11404, 2023, https://doi.org/10.1109/ACCESS.2023.3241279

17] P. Udayasankaran, S. John Justin Thangaraj, “Energy efficient resource utilization and load balancing in virtual machines using prediction algorithms”, in International Journal of Cognitive Computing in Engineering,Volume 4,2023,Pages 127-134,ISSN 2666-3074,https://doi.org/10.1016/j.ijcce.2023.02.005.
[18] Adewojo, A.A., Bass, J.M. A Novel Weight-Assignment Load Balancing Algorithm for Cloud Applications. SN COMPUT. SCI. 4, 270 (2023). https://doi.org/10.1007/s42979-023-01702-7
[19] Wang Y, Dong S, Fan W. Task Scheduling Mechanism Based on Reinforcement Learning in Cloud Computing. Mathematics. 2023; 11(15):3364. https://doi.org/10.3390/math1115336
[20] Saif, M.A.N., Niranjan, S.K., Murshed, B.A.H. et al. CSO-ILB: chicken swarm optimized inter-cloud load balancer for elastic containerized multi-cloud environment. J Supercomput 79, 1111–1155 (2023). https://doi.org/10.1007/s11227-022-04688-w
[21] M. Aljebreen, M. Obayya, H. Mahgoub, S. S. Alotaibi, A. Mohamed and M. A. Hamza, "Chaotic Equilibrium Optimizer-Based Green Communication With Deep Learning Enabled Load Prediction in Internet of Things Environment," in IEEE Access, vol. 12, pp. 258-267, 2024, https://doi.org/10.1109/ACCESS.2023.3345803
[22]Al-Thaedan, A., Shakir, Z., Mjhool, A.Y. et al. Downlink throughput prediction using machine learning models on 4G-LTE networks. Int. j. inf. tecnol. 15, 2987–2993 (2023). https://doi.org/10.1007/s41870-023-01358-9
[23] Sebastián Basterrech, Gerardo Rubino, "Evolutionary Echo State Network: A neuroevolutionary framework for time series prediction," in Applied Soft Computing, Volume 144, 2023, 110463, ISSN 1568-4946, https://doi.org/10.1016/j.asoc.2023.110463
[24] Bai Y-T, Jia W, Jin X-B, Su T-L, Kong J-L, Shi Z-G. Nonstationary Time Series Prediction Based on Deep Echo State Network Tuned by Bayesian Optimization. Mathematics. 2023; 11(6):1503. https://doi.org/10.3390/math11061503
[25] https://visualstudio.microsoft.com/
[26] https://www.incredibuild.com/blog/what-you-need-to-do-to-move-on-to-c-20-the-complete-list
[27] https://www.i2k2.com/