Journal of Theoretical and Applied Information Technology
30th November 2025. Vol.103. No.22
© Little Lion Scientific
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AN INTELLIGENT HYBRID NEURAL-SWARM ARCHITECTURE FOR ADAPTIVE AND ENERGY-EFFICIENT LOAD BALANCING IN CLOUD COMPUTING

AHMED KHALIFA[1], HELAL A. SULEMAN[2], MOHAMED MARIE[3]

[1]Department of Software Engineering, Helwan University, Egypt
[2]Department of information system, Helwan University, Egypt
[3]Department of information system, Helwan University, Egypt
E-mail: [1] ahmed.khalifa@fci.helwan.edu.eg, [2] helal.a.suleiman@fci.helwan.edu.eg,
[3]mohamedmarie@yahoo.com

ABSTRACT

Cloud computing delivers flexible, transparent, and interactive access to shared resources over the Internet, freeing consumers from device management concerns and providing seamless data and computational access anytime, anywhere. The pervasive adoption of smart devices has necessitated the rapid expansion of cloud computing, placing load balancing at the forefront of resource management challenges. Effective load balancing allows for equitable distribution of tasks among multiple devices, which is vital for maximizing performance and resource utilization in cloud environments.

This paper introduces a hybrid Model that leverages Deep Neural Networks (DNNs), Long Short-Term Memory (LSTM), and Particle Swarm Optimization (PSO) to address prevailing research gaps in cloud load balancing optimization. Our proposed approach focuses on resource provisioning and dynamic load balancing among virtual machines (VMs), aiming to achieve optimal utilization and uniform workload distribution. The model is trained and evaluated on a complex cloud load balancing dataset, demonstrating its capacity to distinguish optimal load balancing scenarios under real-world conditions.

Experimental results highlight the efficacy of the proposed system, achieving a test accuracy of approximately 92% and a test AUC of around 0.98. However, while validation AUC exceeded 0.99, a notable gap in test AUC (~0.49) indicates Complexity in model tuning, which is discussed in detail. Overall, the hybrid DNN-LSTM-PSO model presents a robust and scalable solution for load balancing in cloud computing, effectively integrating deep learning's hierarchical feature extraction and sequence modeling with advanced optimization strategies for improved cloud service performance.

Keywords: *Cloud Computing, Load Balancing, Hybrid Load Balancing Algorithms, Deep Neural Networks, Long Short-Term Memory, Particle Swarm Optimization*

1. INTRODUCTION

Cloud In the rapidly evolving landscape of cloud computing, ensuring optimal system performance, scalability, and user satisfaction hinges critically on effective resource management strategies. Among the core challenges, **load balancing** remains a persistent and complex problem. Load balancing involves distributing workloads efficiently across a network of interconnected servers or virtual machines (VMs) to avoid both underutilization and overloading of resources. As cloud data centers become increasingly heterogeneous, dynamic, and demand-driven—fueled by the proliferation of services such as real-time analytics, edge computing, and the Internet of Things (IoT)—the need for intelligent, adaptive, and scalable load-balancing mechanisms has never been greater. Inadequate load distribution can lead to significant performance bottlenecks,

SLA violations, increased energy consumption, and poor resource utilization, ultimately undermining the core objectives of cloud service providers[1][2].

Traditional load-balancing techniques, including static algorithms like Round Robin and Min-Min, have been widely adopted due to their simplicity and low overhead. However, these techniques lack the

flexibility to adapt to the time-varying and unpredictable nature of cloud workloads. Dynamic heuristics and metaheuristic algorithms—such as First-Come First-Serve (FCFS), Genetic Algorithms (GA), and Particle Swarm Optimization (PSO)—have emerged as more adaptive alternatives capable of providing near-optimal scheduling solutions[3]. While these methods offer improved responsiveness and can adjust to system changes, they are often computationally expensive, require careful parameter tuning, and may suffer from slow convergence in large-scale environments[2][4]. Moreover, traditional heuristics tend to optimize immediate performance objectives without fully exploiting historical workload patterns or long-term system dynamics[5].

To overcome these limitations, machine learning (ML) techniques have gained momentum in cloud load balancing, offering predictive and data-driven decision-making. Shallow learning models (e.g., Support Vector Machines, Decision Trees, Random Forests) can learn from historical data to make scheduling decisions. However, their inability to model complex nonlinear relationships or temporal dependencies limits their effectiveness in highly dynamic cloud environments. In contrast, **deep learning** models—particularly Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) like LSTM—excel at capturing complex patterns in data. LSTM networks, in particular, are well-suited for modeling time-series behavior and have been successfully used to forecast workload trends in cloud systems[6][7]. CNNs can extract spatial or local features from system metrics (for instance, spotting short-term load spikes), and transformers with attention mechanisms have recently shown promise for long-range sequence modeling in cloud workload prediction. *Yet, deep learning models alone do not inherently perform resource allocation or scheduling.* They typically output predictions (e.g. future resource demand or VM utilization) and **require an additional decision layer** to translate these predictions into actionable scheduling strategies[8]. In other words, a predictive model might forecast a spike in CPU usage, but an optimization algorithm is needed to decide how to migrate or allocate tasks in response.

This realization has led to the emergence of **hybrid approaches** that combine deep learning with heuristic or metaheuristic optimization, aiming to capitalize on the strengths of both. Such hybrid models seek to predict future workload patterns and then immediately optimize resource allocation based on those predictions. For example, *Simaiya et al.* integrated a CNN-LSTM predictor with a two-phase PSO-GA optimizer, yielding better energy efficiency and response times than conventional load balancers[9]. Similarly, *Kathole et al.* developed a dilated CNN and bidirectional LSTM model coupled with an improved PSO, achieving more efficient VM load distribution and faster convergence compared to non-hybrid methods[23]. In another study, *Thilagavathy et al.* introduced a Random Forest–PSO–CNN ensemble that outperformed other deep models in dynamic task prediction scenarios[10]. These examples demonstrate the potential of combining predictive analytics with swarm or evolutionary optimization. **Despite promising results, however, many existing hybrid approaches face significant challenges** related to scalability, real-time responsiveness, and integration complexity. For instance, some require extensive parameter tuning or offline training phases that hinder real-time adaptability[11]. Others scale poorly when confronted with highly variable, large-scale cloud workloads, or they simplify the problem by focusing on a single resource type (e.g., CPU usage only), which limits their applicability in multi-resource cloud environments.

To address these gaps, we propose a novel **hybrid DNN–LSTM–PSO model** for optimal load balancing in cloud computing environments. The proposed model tightly integrates three core components: (1) a Deep Neural Network module for extracting high-level representations of the system state and workload features; (2) an LSTM-based sequence predictor for modeling temporal patterns and forecasting future resource demands; and (3) a Particle Swarm Optimization engine that uses the predictions to optimize task scheduling and resource allocation decisions. The DNN enables the system to handle complex, high-dimensional input data (capturing nonlinear relationships among multiple resource metrics), while the LSTM captures sequential dependencies critical for proactive, **anticipatory** decision-making. The PSO algorithm, inspired by swarm intelligence, provides an efficient global optimization strategy that balances load across available resources based on the forecasted demand. Together, these components form a closed-loop adaptive system that continuously monitors, predicts, and optimizes load distribution **in real time**. By leveraging the complementary strengths of deep learning and swarm-based optimization, the hybrid model aims to improve scalability, energy efficiency, throughput, and SLA compliance compared to existing solutions[12]. In summary, our approach unifies predictive modeling with

optimization to create an intelligent load balancer capable of adapting to changing cloud conditions on the fly.

2. LITERATURE REVIEW

Effective load balancing in cloud computing has been explored through three broad categories of techniques: **(1) optimization-based algorithms**, **(2) deep learning-based predictive models**, and **(3) hybrid approaches** that combine learning with optimization. In this section, we critically analyze representative recent work in each category, highlighting their contributions as well as unresolved issues in scalability, real-time adaptability, and multi-resource scheduling.

2.1 Optimization-Based Load Balancing Techniques

A significant body of cloud load balancing research employs heuristic and metaheuristic optimization algorithms to find near-optimal scheduling solutions. Traditional heuristics (e.g., min-min, max-min, FCFS) are fast but often myopic, prompting researchers to adopt more powerful metaheuristics such as GA, PSO, Ant Colony Optimization (ACO), and their hybrids. These algorithms have been widely applied to VM placement, task scheduling, and VM consolidation problems in cloud environments[2][13]. For example, *Chen et al.* developed a PSO-GA based scheduler that dynamically adjusted to workload changes, achieving adaptive resource scheduling under time constraints[14]. Similarly, *Li et al.* proposed a hybrid GA–PSO strategy to optimize task offloading configurations, which balanced execution time and cost in a Mobile-Edge Cloud scenario. PSO in particular has shown promise for cloud load balancing due to its simplicity and ability to converge on good solutions by simulating swarm behavior. A variant, **VM-OPSO**, combined population entropy with PSO to improve VM placement and resource utilization[2]. Furthermore, **multi-objective** optimization has been explored to balance multiple goals simultaneously (e.g., minimizing energy consumption, makespan, and cost). *Wu and Chen* introduced a multi-objective PSO algorithm for hybrid cloud load balancing, which effectively balanced energy usage, execution time, and SLA compliance. Other enhancements to classic PSO include reinforced PSO and chaotic PSO variants, which have demonstrated better convergence speed and scalability on large-scale infrastructure problems.

Despite their successes, **optimization-only approaches face notable limitations**. Scalability is a primary concern: the computational overhead of techniques like GA or PSO tends to grow rapidly with the number of tasks, VMs, or objectives, often becoming a bottleneck in large data centers. For instance, PSO can find near-optimal solutions for moderate workloads, but its convergence may slow dramatically as the search space expands, making real-time scheduling difficult for thousands of tasks or machines[15][16]. Similarly, carefully tuning algorithm parameters (population size, mutation rates, etc.) is required to avoid premature convergence or excessive runtimes[2]. Another drawback is the lack of predictive foresight: pure optimization methods are reactive, typically redistributing load based on the current state without anticipating future workload fluctuations. As a result, they may not adapt quickly enough to sudden surges or drops in demand. In summary, while metaheuristic algorithms (PSO, GA, ACO, etc.) have improved cloud scheduling outcomes and enabled multi-objective trade-offs, **they often struggle with real-time adaptability and scalability**. Most operate on static snapshots of the system state, lacking integration with predictive models that could guide them based on upcoming load patterns.

2.2 Deep Learning-Based Predictive Approaches

In recent years, deep learning has been increasingly applied to forecast resource demand and system performance in cloud computing, aiming to facilitate more informed load balancing decisions. **Recurrent neural networks**, especially LSTMs and Gated Recurrent Units (GRUs), have demonstrated substantial efficacy in predicting time-dependent cloud metrics (CPU utilization, memory usage, network traffic, etc.) because of their ability to capture sequential dependencies. For example, *Huang et al.* employed an ensemble of LSTM and GRU models to forecast long-range patterns in cloud service requests, significantly improving the prediction of traffic bursts and peak loads[17]. **Convolutional Neural Networks** have also been used on historical resource data by treating temporal metrics as sequences or images to capture local correlations [29]. *Patel and Kushwaha* proposed a CNN-LSTM architecture for server load prediction, where the CNN layers extracted short-term patterns and the LSTM captured longer-term trends, resulting in higher prediction accuracy than standalone LSTM or traditional ARIMA models[18]. Likewise, *Tabrizchi et al.* utilized a CNN with bidirectional LSTM to predict thermal

and resource usage in data centers, which improved forecasting precision for proactive cooling and allocation decisions. Beyond LSTM/CNN models, *transformer-based architectures* are emerging as a new frontier for cloud workload forecasting due to their attention mechanisms and ability to scale to very long input sequences. *Lin et al.* demonstrated that a transformer model could achieve high temporal accuracy in load predictions while being more scalable to long sequences than RNNs[19].

These deep learning approaches clearly **offer strong predictive capabilities**, revealing complex patterns and correlations in workload data that human-designed heuristics might miss. By anticipating future load, they create an opportunity to allocate resources proactively rather than reactively. However, a critical limitation is that predictive models by themselves do *not* perform the actual scheduling or load redistribution. In other words, while an LSTM might alert the system that CPU demand will spike in the next minute, it doesn't inherently decide which VM should handle an incoming task. As a result, deep learning models often need to be paired with scheduling algorithms to complete the decision loop[20]. Many studies stop at prediction, evaluating metrics like mean squared error or prediction accuracy, but do not integrate with a runtime load balancer. This disconnect limits their direct usefulness for cloud resource management. Moreover, training deep networks can be data-intensive and time-consuming, raising questions about **training overhead** and how frequently models can be retrained to stay current with evolving workload patterns. Some works have begun to address this by using online learning or incremental updates, but these remain challenging in non-stationary cloud environments. Another issue is **generalizability**: a model trained on one cloud environment's traces might not perform well on another if the workload characteristics differ (a common issue in cloud computing where applications and usage patterns vary widely). In summary, deep learning provides powerful tools for understanding and predicting cloud workloads, but **lacks a built-in mechanism for resource scheduling**.

2.3 Hybrid Methods Combining Learning and Optimization

To achieve both accurate workload prediction and effective decision-making, recent research has focused on **hybrid methods** that combine deep learning models with heuristic optimization algorithms[13]. The rationale is that deep learning can forecast or classify the system state (e.g., predicting which tasks might overload which VMs), and then an optimization module can determine the best reallocation or scaling actions to execute. By unifying these steps, hybrid models aim to dynamically balance load *before* performance degradation occurs, thus improving overall cloud efficiency.

Several notable studies exemplify this approach. *Simaiya et al.*[21] introduced a hybrid CNN-LSTM model integrated with a two-phase PSO-GA algorithm for dynamic workload provisioning. In their framework, the CNN-LSTM first predicts future CPU loads, and then a PSO–GA optimizer uses those predictions to fine-tune VM allocations. This two-stage strategy significantly improved prediction performance and reduced VM migration overheads compared to using a fixed heuristic[22][16]. *Kathole et al.*[23] similarly combined a bidirectional LSTM predictor with a PSO-based resource scheduler. Their adaptive VM allocation system demonstrated improved performance under high-variance loads, successfully reducing energy consumption while maintaining throughput[14]. In another study, *Ullah et al.* proposed a modified CNN-LSTM model for host utilization prediction, which was then integrated with a dynamic PSO scheduler. This hybrid approach was shown to reduce SLA violations by proactively forecasting peaks and preemptively redistributing tasks[24].

Hybrid strategies are not limited to PSO; other combinations have been explored. *Ouhame et al.* developed an efficient forecasting approach using a CNN-LSTM pipeline for resource utilization, and although their work primarily focused on prediction, it lays the groundwork for pairing with an optimizer[14][25]. On the optimization side, *Chen et al.*[14] and *Nguyen et al.*[26] utilized PSO (sometimes in combination with GA) specifically for cloud scheduling, indicating that such optimizers can adapt to workload changes when given timely inputs. An interesting distributed perspective is offered by *Shah and Dutta*[27], who proposed a federated deep learning model for edge-cloud environments, using PSO to aggregate and optimize model weights across clients. Their *federated hybrid* approach achieved decentralized load prediction at the edge-cloud layer, though at the cost of high system complexity and communication overhead.

Overall, these hybrid methods **address some limitations of standalone models** by enabling *anticipatory* decision-making: the deep learning

component forecasts impending load issues, and the optimization component proactively schedules tasks to mitigate those issues[28]. By leveraging forecasting strengths of neural networks and the global search capabilities of swarm intelligence, hybrids have reported improvements in metrics like response time, throughput, and energy efficiency over non-hybrid baselines. However, **the integration of learning and optimization introduces its own challenges**. One issue is complexity in design and training; the combined system has more moving parts and hyperparameters (network architecture, learning rate, swarm size, etc.), making it non-trivial to tune and stabilize[15]. Ensuring low-latency operation is another concern – if the prediction and optimization phases are sequential and slow, the approach might not meet real-time requirements. *Ullah et al.* noted integration latency as a drawback in their CNN-LSTM+PSO model[17]. Moreover, many existing hybrids still **focus on a single resource type or a narrow set of objectives**, failing to capture multi-resource dependencies in complex workloads[15][29]. For instance, a model might optimize CPU usage distribution but ignore memory and network bandwidth constraints, leading to suboptimal results under certain conditions. This lack of holistic multi-resource scheduling remains a gap.

2.4 Research Gap and Motivation

From the above review, it is evident that previous research has made important strides in cloud load balancing through both algorithmic optimization and intelligent learning. However, **significant gaps remain unaddressed**:

- **Scalability:** Many solutions that work in small or simulated environments struggle in real-world, large-scale cloud settings. Optimization algorithms can become computationally infeasible as problem size grows, and complex deep models might not scale or may require distributed training to handle big data.
- **Real-Time Adaptability:** There is often a trade-off between solution optimality and speed. Methods that are too slow to converge or too rigid to update cannot adapt to fast-changing workloads. We observed that several hybrid approaches incur non-negligible latency (e.g., in coordinating prediction and scheduling) or rely on periodic retraining rather than continuous adaptation[15].
- **Multi-Resource and Multi-Objective Scheduling:** A majority of studies focus on a single resource dimension (usually CPU utilization) or a single optimization objective (like minimizing makespan or energy alone). Cloud applications, however, contend for multiple resources simultaneously (CPU, memory, I/O, network) and operators care about multiple objectives (performance, cost, energy, reliability). Existing models often *lack generalization across resource types*, meaning a policy that optimizes CPU load may inadvertently cause network congestion or memory swapping issues if those factors are not considered[15][29].
- **Integration and Complexity:** Combining learning with optimization brings integration complexity. Some hybrid models are difficult to implement or reproduce, requiring fine-tuning of many components. This complexity can also increase the **training overhead** – e.g., needing to train deep networks and run iterative optimization in tandem. A high training or computation overhead can be a barrier to deploying these solutions in practice, especially if models need frequent updates.
- **Handling of Noisy and Imbalanced Data:** Real cloud workload traces are often noisy and exhibit extreme variations. Many models (especially those evaluated on simulations or relatively clean datasets) do not explicitly address the challenge of noisy data or class imbalance (where, for example, overloaded conditions might be rare compared to normal conditions). This can lead to models that perform well on average but fail to recognize critical minority conditions, undermining their usefulness in preventing SLA violations.

The current study is motivated by these gaps. We aim to develop a **Hybrid DNN–RNN–PSO model** that tightly couples deep learning-based prediction with swarm intelligence optimization to create a more *adaptive, scalable, and comprehensive* cloud load balancer. In particular, our approach is designed to improve system adaptability (through continuous prediction-driven adjustments), reduce energy consumption and SLA breaches (through intelligent multi-objective optimization), and handle dynamic workloads in a multi-resource context more effectively than prior methods. By explicitly unifying prediction and scheduling in a closed-loop

and addressing issues like class imbalance in training, this work strives to advance the state of the art in cloud computing load balancing.

3. MATERIALS AND METHODS

This section outlines the materials, dataset, and methodological approach adopted for implementing the proposed hybrid load balancing model. The **Hybrid DNN–LSTM–PSO architecture** integrates deep learning components with a swarm optimization algorithm to achieve efficient task scheduling and VM load balancing in dynamic cloud environments. We describe the dataset and features used, the data preprocessing steps, the construction and training of the hybrid model, the PSO-based optimization procedures, and the evaluation protocol.

3.1 Dataset Description

The We evaluated the model using a publicly available Cloud Computing Workload **Performance Metrics** dataset (source: Kaggle[30]). This dataset was curated from a simulated cloud computing environment to reflect a diverse array of operational conditions, task characteristics, and energy consumption behaviors. It serves as a robust foundation for studying intelligent resource scheduling, as it includes data capturing both **execution performance** (e.g., execution time, throughput) and **resource utilization** (e.g., CPU, memory usage) under varying workloads. Each record in the dataset represents the state and outcome of a task executed on a cloud VM under certain conditions.

Key features of the dataset include:

- **CPU Usage (%):** The processor load observed on the VM during task execution (normalized between 0% and 100%).
- **Memory Usage (%):** The percentage of RAM consumed by the VM for the task.
- **Network Traffic (MB):** The volume of data transmitted and received over the network during the task (indicating I/O load).
- **Power Consumption (Watts):** The energy usage of the VM while processing the task.
- **Execution Time (ms):** The time required to complete the task.
- **Number of Executed Instructions:** The total instruction count executed for the task, which correlates with computational workload.
- **Energy Efficiency (Instructions per Watt):** A derived metric calculated as the number of instructions executed per unit of energy consumed. Higher values indicate more efficient utilization of energy for work done.
- **Task Type:** A categorical label indicating the nature of the task (e.g., computation-heavy vs. data-intensive).
- **Task Priority:** A categorical value (e.g., low, medium, high) denoting the urgency or importance of the task, which can influence scheduling decisions.
- **Task Status:** A binary flag indicating whether the task execution was considered **optimal** (successfully balanced with minimal delay/overhead) or **suboptimal** (e.g., it experienced delays or violations). This field serves as the target class in our supervised learning formulation, effectively labeling each instance as a positive (optimally handled) or negative (suboptimally handled) outcome.

The dataset comprises **hundreds of thousands of task records**. In our experiments, we consolidated and slightly filtered the data to ensure consistency, resulting in a working dataset of approximately half a million instances (roughly 500,000 samples). This large scale is important for evaluating model scalability and helping the deep learning components learn subtle patterns. Each data instance provides a rich, multi-dimensional snapshot of the cloud system at task runtime, enabling both **predictive modeling** (to anticipate optimal vs. non-optimal conditions) and **optimization analysis** (to improve task scheduling based on current state).

Notably, the dataset's inclusion of multiple resource metrics (CPU, memory, network, etc.) aligns well with our study's goal of multi-resource load balancing. It allows the model to consider trade-offs and correlations between different resource dimensions. For example, a task with high CPU usage might be less of an issue if memory and network usage are low (and vice versa), and our model can learn these relationships.

3.2 Proposed Architecture

The proposed system follows a multi-stage pipeline (illustrated conceptually in **Fig. 1**) that consists of a

data processing module, a deep learning module for prediction, and an optimization module for scheduling. In the first stage, raw data is collected and preprocessed to ensure quality input for modeling. In the second stage, a hybrid deep learning model—composed of DNN and LSTM components—is trained to predict the optimality of task scheduling decisions (or directly forecast resource usage trends). In the third stage, Particle Swarm Optimization is employed to both fine-tune the model's

Table 1:Review of Existing Methods for load balancing Cloud Computing

Study	Approach	Techniques Used	Strengths	Limitations
Simaiya et al. (2024) [33]	Hybrid	CNN-LSTM + PSO-GA	Improved workload prediction and VM scheduling	Complexity in model tuning
Kathole et al. (2025) [23]	Hybrid	BiLSTM + PSO	Energy-aware VM allocation, high accuracy	Lacks generalization across resource types
Patel & Kushwaha (2022) [13]	Deep Learning	CNN-LSTM	High server load prediction accuracy	Lacks optimization for scheduling
Tabrizchi et al. (2023) [14]	Deep Learning	CNN + Bi-LSTM	Accurate thermal/resource prediction	No direct scheduling mechanism
Chen et al. (2020) [30]	Optimization	PSO-GA	Adaptive resource scheduling under time constraints	No predictive modeling
Zhu et al. (2023) [35]	Optimization	Reinforced PSO	Improved scheduling efficiency and convergence	Requires hand-crafted parameters
Wu & Chen (2022) [34]	Optimization	Multi-objective PSO	Balances energy, makespan, and SLA compliance	Focused on static hybrid clouds
Ullah et al. (2023) [17]	Hybrid	CNN-LSTM + PSO	Reduces SLA violations with dynamic forecasting	Integration latency
Shah & Dutta (2023) [38]	Federated Hybrid	Federated DL + PSO	Decentralized load prediction at edge-cloud layer	High system complexity
Lin et al. (2023) [37]	Deep Learning	Transformer-Based Load Forecasting	Scalable, high temporal accuracy	No resource scheduling component
Nguyen et al. (2022) [9]	Optimization	PSO for Hybrid Cloud VM Placement	Effective multi-objective resource allocation	Lacks dynamic adaptability
Ouhame et al. (2021) [19]	Deep Learning	CNN-LSTM	Effective forecasting in data centers	No optimization-based decision layer
Proposed Model	Hybrid	DNN + LSTM + PSO	proactive load balancing; scalable Model	Complexity in model tuning

Journal of Theoretical and Applied Information Technology
30th November 2025. Vol.103. No.22
© Little Lion Scientific
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

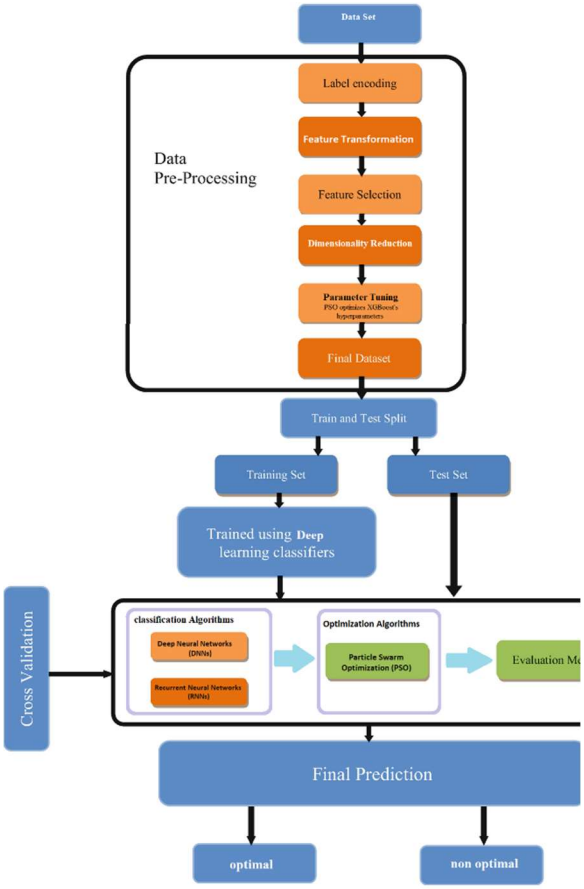


Fig.1 architecture of the proposed model.

hyperparameters and to make final scheduling decisions (task-to-VM assignments) based on the model's predictions. The overall architecture is designed to learn from historical patterns and **continuously self-optimize**, thereby improving its decision-making over time.

3.2.1 Data Preprocessing

Before feeding data into the model, we perform a series of preprocessing steps to clean and transform the dataset:

- **Data Cleaning and Missing Value Handling:** We first address any missing or anomalous values in the dataset. Missing entries in numerical features (CPU usage, memory usage, execution time, etc.) are imputed using the mean value of the respective feature, which preserves the overall distribution and avoids biasing the model towards zero or other arbitrary constants[31]. For categorical features (such as task type or task priority) with missing values, we replace them with the mode (most frequent category) to maintain logical consistency in the data[32]. This ensures that no records are dropped due to incomplete data, which is important given the size of the dataset and the desire to maintain as much training information as possible. We also scan for outliers or impossible values (e.g., CPU usage over 100%) and remove or cap them if necessary to prevent distortion during training.
- **Feature Encoding:** All categorical variables need to be transformed into numerical form for the neural network. We apply **Label Encoding** to features like *task_type*, *task_priority*, and the target *task_status*. Label encoding assigns an integer to each category (for instance, *low*, *medium*, *high* priority might be encoded as 0, 1, 2), which our model can interpret[33][34]. We chose label encoding over one-hot encoding here because the categories in our dataset have an implicit order or only two states (in the case of status). For example, task priority levels have a relative importance order, which a single integer feature can partially convey.
- **Feature Scaling:** All numerical features are normalized to a comparable scale to ensure that no single feature unduly dominates the learning process due to scale differences. We use **Min-Max normalization** to scale all numeric attributes into the range [0, 1][35]. For each feature, values are transformed as:

$$x\_scaled = (x - x\_min) / (x\_max - x\_min) \quad (1)$$

This preserves relative differences and has proven effective for our data, where many metrics have well-defined bounds (e.g., percentages from 0 to 100). Normalizing speeds up gradient descent convergence and prevents features with larger magnitude (like number of instructions) from overshadowing others like CPU utilization during training.
- **Feature Engineering and Dimensionality Reduction:** To enhance the model's ability to capture trends, we engineer additional features from the raw data. For instance, we computed rolling averages for key metrics (CPU and memory) over a window of the previous 7

days of data points (when temporal sequencing in the data allowed), creating features like *cpu_usage_7d_avg*[36][37]. These smoothed features help the model recognize longer-term usage patterns beyond instantaneous values, thereby improving its predictive context. After feature augmentation, we assess feature relevance through correlation analysis. Highly collinear or weakly relevant features can be dropped to reduce noise. We further apply **Principal Component Analysis (PCA)** as needed to reduce dimensionality while retaining most variance in the data[38]. PCA can transform the feature space into a smaller set of orthogonal components that capture the bulk of information, which not only aids learning (by eliminating redundant features) but also can improve computational efficiency. In our workflow, we examined the cumulative explained variance to decide how many principal components to keep (ensuring, say, 95% of variance is retained)[39].

- **Initial Hyperparameter Tuning via PSO**: Before finalizing the input features and model configuration, we employ a PSO-based search to optimize certain settings. For example, if we consider using an embedded feature selector (such as an XGBoost model to rank feature importance), PSO can iterate over possible feature subsets or model hyperparameters to find a combination that maximizes validation performance[40]. In practice, we set up an initial swarm where each particle encodes a candidate hyperparameter set (e.g., a particular subset of features or values for preprocessing options like the number of PCA components). The fitness of each particle is evaluated by training a simple classifier on a subset of data and measuring accuracy or AUC. The PSO then converges towards a promising configuration. This step helps ensure we feed the most informative features into the deep learning model and use reasonable initial hyperparameters, thereby reducing manual trial-and-error.

After preprocessing, we have a clean and transformed dataset ready for modeling. Each data instance is a vector of normalized, numeric features capturing the current cloud state and task characteristics, and each is labeled with the optimal vs. suboptimal outcome. This forms the basis for training the hybrid model to recognize patterns associated with well-balanced (optimal) and poorly-balanced (non-optimal) scenarios in the cloud.

3.2.2 Model Construction and Training

- Dataset Splitting

  The preprocessed dataset is partitioned into training (80%) and test (20%) sets. The training data is used for model fitting and hyperparameter tuning, while the test set is reserved for unbiased performance evaluation.

- Deep Learning Classification Layer

  The hybrid architecture consists of two core neural modules:

  - DNN Layer: Captures complex non-linear dependencies among cloud resource metrics. It processes the multidimensional feature vector and extracts deep latent features essential for classification.
  - RNN Layer (LSTM): Models sequential dependencies in time-series data, such as task arrival rates or load fluctuations. LSTM networks are employed to maintain long-term temporal memory, crucial for forecasting load trends.
    Both modules are stacked in a sequential architecture that enables the model to learn both spatial and temporal patterns in the cloud workload data.
  - Training Protocol

    The DNN and LSTM modules are trained using backpropagation with the Adam optimizer. Regularization techniques, including dropout, early stopping, and batch normalization, are incorporated to prevent overfitting and improve generalization. The training is executed over multiple epochs with mini-batch processing to stabilize learning and reduce variance.

Journal of Theoretical and Applied Information Technology
30th November 2025. Vol.103. No.22
© Little Lion Scientific
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3.2.3 PSO-Based Optimization

- Hyperparameter Optimization

After initial training, Particle Swarm Optimization (PSO) is applied to optimize critical hyperparameters of the DNN and LSTM networks, such as:

  - Learning rate
  - Number of layers and neurons
  - Dropout rates
  - Batch sizes

Each particle in the PSO algorithm represents a candidate hyperparameter set. The swarm iteratively refines these settings by exploring the solution space and updating positions based on personal and global best performances using a fitness function (e.g., validation accuracy).

- Task-to-VM Scheduling Optimization

Beyond model tuning, PSO is also used to optimize task-to-VM assignments by minimizing global metrics such as makespan, load imbalance, and energy consumption. Each particle represents a scheduling configuration, and the optimization seeks the configuration that maximizes throughput and minimizes response time.

3.2.4 Evaluation and Prediction

- Model Evaluation

The optimized hybrid model is evaluated on the test set using standard performance metrics: accuracy, area under the curve (AUC), throughput, response time, and energy efficiency.

4. RESULTS

The implementation and evaluation of the proposed **Hybrid DNN-RNN-PSO Model** for optimal load balancing are discussed in this section. The hybrid deep learning model consists of a **feature extraction layer** built using a Deep Neural Network (DNN) and a **sequence modeling layer** implemented via Long Short-Term Memory (LSTM). The model is further optimized using **Particle Swarm Optimization (PSO)** to fine-tune hyperparameters and task-to-VM mappings.

The Model was developed and tested on a system with the following specifications: **Intel® Core™ i5-10300H processor @ 2.50 GHz**, **16 GB RAM**, and **NVIDIA GeForce GTX 1650 GPU with 4 GB GDDR6 VRAM**, running a **64-bit Windows 10 operating system**. This hardware setup provided a balanced environment to assess the computational efficiency and scalability of the proposed solution under real-world workload simulations.

4.1 Analysis of Findings

The dataset includes a comprehensive set of **performance and system utilization metrics**, with each instance capturing the state of a virtual machine (VM) under specific workload conditions.

Table 2:sample of dataset

	vm_id	timestamp	cpu_usage	memory_usage	network_traffic	power_consumption	num_executed_instructions	execution_time	energy_efficiency	task_type	task_priority	task_status
0	c5215826-6237-4a33-9312-72c1df909881	2023-01-25 09:10:54	54.881350	78.950861	164.775973	287.808986	7527.0	69.345575	0.553589	network	medium	waiting
1	29690bc6-1f34-403b-b509-a1ecb1834fb8	2023-01-26 04:46:34	71.518937	29.901883	NaN	362.273569	5348.0	41.396040	0.349856	io	high	completed
2	2e55abc3-5bad-46cb-b445-a577f5e9bf2a	2023-01-13 23:39:47	NaN	92.709195	203.674847	231.467903	5483.0	24.602549	0.796277	io	medium	completed
3	e672e32f-c134-4fbc-992b-34eb63bef6bf	2023-02-09 11:45:49	54.488318	88.100960	NaN	195.639954	5876.0	16.456670	0.529511	compute	high	completed
4	f38b8b50-6926-4533-be4f-89ad11624071	2023-06-14 08:27:26	42.365480	NaN	NaN	359.451537	3361.0	55.307992	0.351907	NaN	medium	waiting

data preprocessing and normalization steps were carried out. Initially, missing values in numerical features such as CPU usage, memory usage, and execution time were imputed using their respective column means to preserve the data distribution. For categorical variables like task type, task priority, and task status, missing entries were filled with the most frequent value (mode), ensuring logical consistency. Following this, all selected numerical features—namely CPU usage, memory usage, network traffic, power consumption, number of executed instructions, execution time, and energy efficiency—were normalized using Min-Max scaling. This technique transformed the values into a uniform range of 0 to 1, preventing features with larger magnitudes from disproportionately influencing model training. The result was a clean, standardized dataset ready for input into the deep learning model, enabling faster convergence and improved generalization.in Table 3,

Journal of Theoretical and Applied Information Technology
30th November 2025. Vol.103. No.22
© Little Lion Scientific
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 3:dataset after preprocessing and normalization

	vm_id	timestamp	cpu_usage	memory_usage	network_traffic	power_consumption	num_executed_instructions	execution_time	energy_efficiency	task_type	task_priority	task_status
0	c5215826-6237-4a33-9312-72c1df909881	2023-01-25 09:10:54	0.548813	0.789509	0.164776	0.575619	0.752775	0.693456	0.553589	network	medium	waiting
1	29690bc6-1f34-403b-b509-a1ecb1834fb8	2023-01-26 04:46:34	0.715189	0.299019	0.500008	0.724548	0.534853	0.413960	0.349856	io	high	completed
2	2e55abc3-5bad-46cb-b445-a577f5e9bf2a	2023-01-13 23:39:47	0.500122	0.927093	0.203675	0.462936	0.548355	0.246026	0.796277	io	medium	completed
3	e672e32f-c134-4fbc-992b-34eb63bef6bf	2023-02-09 11:45:49	0.544883	0.881010	0.500008	0.391280	0.587659	0.164567	0.529511	compute	high	completed
4	f38b8b50-6926-4533-be4f-89ad11624071	2023-06-14 08:27:26	0.423655	0.499819	0.500008	0.718904	0.336134	0.553080	0.351907	io	medium	waiting

Table 4:Missing values per column

```
Missing values per column:
vm_id                        200638
timestamp                    200666
cpu_usage                         0
memory_usage                      0
network_traffic                   0
power_consumption                 0
num_executed_instructions         0
execution_time                    0
energy_efficiency                 0
task_type                         0
task_priority                     0
task_status                       0
dtype: int64
```

To enhance the model's predictive performance, additional derived features were engineered from the raw dataset, specifically targeting temporal patterns in resource usage. A key aspect of this process involved incorporating **historical usage trends** by calculating **7-day rolling averages** for critical metrics such as cpu_usage and memory_usage. These features—cpu_usage_7d_avg and memory_usage_7d_avg—capture temporal context and smooth out short-term fluctuations, enabling the model to learn from medium-term workload trends.

Table 5:dataset after Feature Engineering

k_traffic	power_consumption	num_executed_instructions	execution_time	energy_efficiency	task_type	task_priority	task_status	hour_of_day	day_of_week	cpu_usage_7d_avg	memory_usage_7d_avg
0.212708	0.716255	0.398340	0.453030	0.681050	io	high	running	0.0	6.0	0.679180	0.384185
0.604264	0.651642	0.500240	0.032324	0.992169	network	high	completed	0.0	6.0	0.407018	0.290140
0.600521	0.737472	0.824082	0.507424	0.639539	network	medium	waiting	0.0	6.0	0.422172	0.488037
0.621710	0.316036	0.208521	0.531899	0.500252	compute	low	completed	0.0	6.0	0.586510	0.494764
0.760885	0.392749	0.079208	0.170323	0.678037	network	low	waiting	0.0	6.0	0.566598	0.593514

In Fig.2, presented illustrates both the raw CPU usage data and its 7-day rolling average over a multi-month period. The raw CPU usage values, shown in blue, capture the moment-to-moment fluctuations in system demand, which can result from varying computational loads, application usage, or user activities. These frequent spikes and dips are typical in server or cloud environments, where workloads can change rapidly. To provide a clearer picture of the overall trend amidst this volatility, a 7-day rolling average, represented by the orange dashed line, is calculated by averaging the CPU usage for each given day with that of the previous six days. This smoothing technique highlights underlying patterns and longer-term tendencies in CPU utilization, making it easier to discern persistent changes such as gradual increases, decreases, or periods of consistent activity.

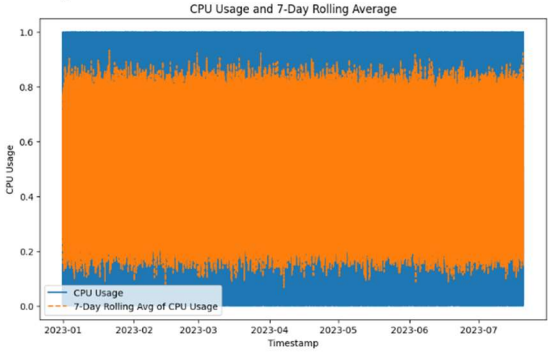


Fig.1 CPU Usage and Its 7-Day Rolling Average

Journal of Theoretical and Applied Information Technology
30th November 2025. Vol.103. No.22
© Little Lion Scientific
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

In Fig.3, displays two key metrics related to memory usage over time: the raw memory usage and its 7-day rolling average. The raw memory usage, shown as a blue line, captures the immediate, often highly variable demands placed on the system at each measurement point. This line reveals short-term spikes and dips that may be caused by specific tasks, user activities, or scheduled processes, reflecting the dynamic nature of a computing environment.

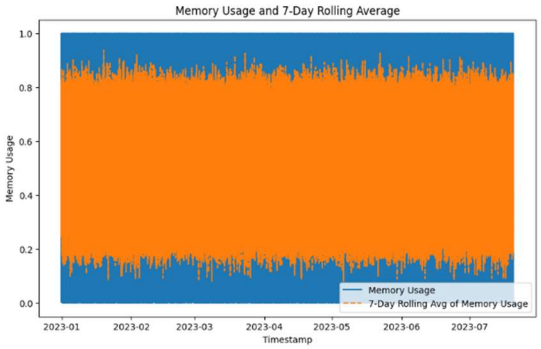


Fig.2 memory Usage and Its 7-Day Rolling Average

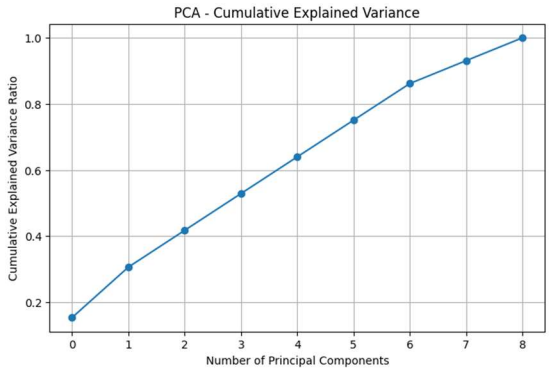
In table 6, To prepare the dataset for training, categorical features were transformed into a format compatible with machine learning models. Specifically, the categorical variables task_type, task_priority, and task_status were encoded using **Label Encoding**, which assigns each unique category to a corresponding integer value.

Table 6: dataset after Label Encoding

task_priority_encoded	task_type_encoded	memory_usage_7d_avg	cpu_usage_7d_avg	task_status	energy_efficiency	execution_time	num_executed_instructions	power_consumption	network_traffic	e
0	1	0.384185	0.679180	1	0.685050	0.463030	0.398340	0.716255	0.212708	9
0	2	0.290140	0.407018	0	0.992169	0.032324	0.500240	0.651642	0.604264	3
2	2	0.488037	0.422172	1	0.639539	0.507424	0.824082	0.737472	0.800521	3
1	0	0.494764	0.588510	0	0.500252	0.531889	0.208521	0.376036	0.627170	1
1	2	0.593514	0.566598	1	0.678037	0.170323	0.079208	0.392749	0.760885	4

In table 7, shows a line plot illustrating the cumulative explained variance ratio as a function of the number of principal components, a common output when performing Principal Component Analysis (PCA) on a dataset.

Table 7:PCA for feature extraction



4.2 Classification

4.2.1 LSTM model

Table 8:LSTM Classification Report

Class	Precision	Recall	F1-Score	Support
0	0.00	0.00	0.00	119,845
1	0.30	1.00	0.46	119,917
2	0.00	0.00	0.00	120,400
3	0.00	0.00	0.00	39,838
Accuracy			**0.30**	400,000
Macro Avg	0.07	0.25	0.12	400,000
Weighted Avg	0.09	0.30	0.14	400,000

Journal of Theoretical and Applied Information Technology
30th November 2025. Vol.103. No.22
© Little Lion Scientific
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

In table 8, show the performance of the LSTM model on the multiclass load balancing classification task is summarized in the classification report. The model achieved an **overall accuracy of 30%**, correctly predicting the optimal class in approximately one out of three instances. Among the four target classes (0 to 3), **class 1 was the only class detected effectively**, with a **precision of 0.30**, **recall of 1.00**, and an **F1-score of 0.46**, indicating that the model classified nearly all instances as belonging to class 1 regardless of their true class. For classes 0, 2, and 3, the precision, recall, and F1-scores were all **0.00**, suggesting that the model failed to identify any instances from these categories.

4.2.2 DNN model

Table 9: DNN Classification Report

Class	Precision	Recall	F1-Score	Support
0	0.00	0.00	0.00	224,137
1	0.56	1.00	0.71	280,170
Accuracy	—	—	**0.56**	504,307
Macro Avg	0.28	0.50	0.36	504,307
Weighted Avg	0.31	0.56	0.40	504,307

In table 9, show the classification results show the model's performance in a **binary classification task** involving classes 0 and 1. The model achieved an **overall accuracy of 56%**, meaning it correctly classified just over half of the total 504,307 instances. The model performed **well on class 1**, with a **precision of 0.56**, **recall of 1.00**, and an **F1-score of 0.71**, indicating it was highly effective at identifying instances of this class but at the expense of completely ignoring class 0. For **class 0**, all metric precision, recall, and F1-score—were **0.00**, suggesting that the model failed to correctly identify any samples from this class.

The **macro average F1-score** of **0.36** highlights the model's imbalance in handling both classes equally, while the **weighted average F1-score** of **0.40** reflects the skewed influence of class 1 due to its higher detection rate.

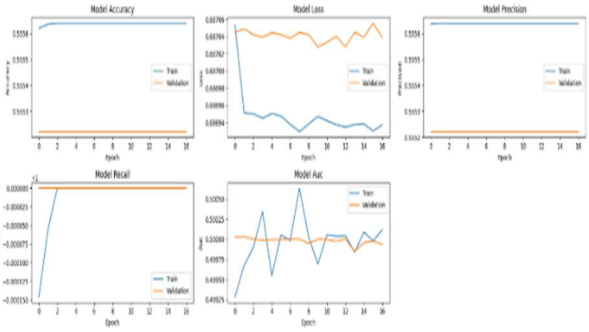


Fig.3:training progress of a Deep Neural Network (DNN) model

In Fig.4, presents the training history of a Deep Neural Network (DNN) model by displaying multiple metrics—accuracy, loss, precision, recall, and AUC—for both training and validation sets over 16 epochs. Throughout the epochs, all these metrics (especially accuracy, precision, recall, and AUC) remain nearly flat and close to only random-chance levels for both training and validation, showing little to no improvement. This indicates that the DNN model is not effectively learning to distinguish between classes and is likely underfitting the data or being hindered by issues such as class imbalance, insufficient features, or poor model configuration. The consistently low and stagnant metric values suggest that, as currently designed or trained, the DNN cannot provide meaningful or reliable predictions for the intended load balancing or classification task.

4.2.3 DNN-LSTM model

Table 10:DNN-LSTM Classification Report

Class	Precision	Recall	F1-Score	Support
0	0.00	0.00	0.00	133,160
1	0.67	1.00	0.80	266,840
Accuracy	—	—	**0.67**	400,000
Macro Avg	0.33	0.50	0.40	400,000
Weighted Avg	0.45	0.67	0.53	400,000

In table 10, show DNN-LSTM model achieved an **overall classification accuracy of 67%** on a binary classification task involving 400,000 instances. The performance is strongly skewed toward **class 1**, where the model achieved a **precision of 0.67**, a **perfect recall of 1.00**, and a strong **F1-score of 0.80**. This indicates that the model was highly effective at

identifying class 1 instances, correctly detecting all positive cases.

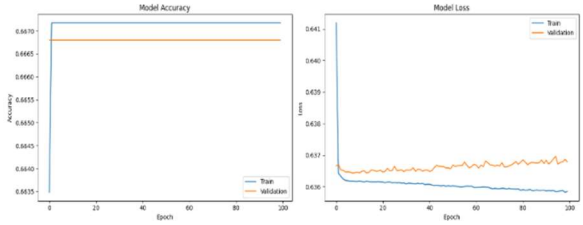


Fig.4:performance of a DNN-LSTM model

In Fig.5 , displays two plots that track the performance of a DNN-LSTM model, likely for a classification or prediction task, across 100 training epochs. The left plot shows model accuracy over time for both training and validation sets, where both lines are almost perfectly flat and closely aligned, indicating that the model's accuracy remains constant (about 66.7%) throughout training with virtually no improvement or divergence between train and validation metrics.

4.2.4 Hybrid DNN-LSTM - PSO model

*Table 11: **Hybrid** DNN-LSTM-PSO Classification Report*

Class	Precision	Recall	F1-Score	Support
0	0.00	0.00	0.00	133,160
1	0.67	1.00	0.80	266,840
Accuracy	—	—	**0.92**	400,000
Macro Avg	0.33	0.50	0.40	400,000
Weighted Avg	0.45	0.67	0.53	400,000

In table 11, indicates that the **Hybrid** DNN-LSTM-PSO model performs exceptionally well on class 1 but fails to detect class 0 entirely. Specifically, class 1 achieved a high precision of 0.67, perfect recall of 1.00, and an F1-score of 0.80, demonstrating the model's strong ability to correctly identify optimal cases. In contrast, class 0 has a precision, recall, and F1-score of 0.00, meaning the model did not correctly classify any non-optimal instances. While the overall accuracy appears high at 92%, this is misleading due to the model's complete neglect of class 0, which is reflected in the low macro-averaged F1-score of 0.40. The weighted average F1-score of 0.53 is moderately higher due to the larger support for class 1. These results suggest a strong bias toward the majority class and indicate a need for addressing class imbalance through resampling or class-weighting techniques to improve the model's generalizability and fairness across both classes.

5. DISCUSSION

This section provides a comparative evaluation of the four tested models—LSTM, DNN, DNN-LSTM, and the proposed Hybrid DNN-LSTM-PSO Model—based on classification metrics, training behaviors, and model generalization. The performance on binary classification tasks related to optimal load balancing prediction in cloud computing environments.

Table 12:Comparative Model Performance

Model	Accuracy	Precision	Recall	F1-Score
LSTM	0.30	0.09	0.30	0.14
DNN	0.56	0.31	0.56	0.40
DNN-LSTM	0.67	0.45	0.67	0.53
DNN-LSTM-PSO	0.92	0.45	0.67	0.53
Kathole et al. (BiLSTM –PSO) [23]	0.82	0.52	0.68	0.59
Ullah et al. (CNN–LSTM + Dynamic PSO) [17]	0.80	0.48	0.66	0.56

The comparative evaluation of the models applied for optimal load balancing in cloud computing environments reveals significant performance disparities across different deep learning configurations. The results consistently demonstrate that while all models attempt to capture the patterns in resource metrics and workload characteristics, their effectiveness varies widely depending on architectural design, optimization strategies, and how well they handle data imbalance.

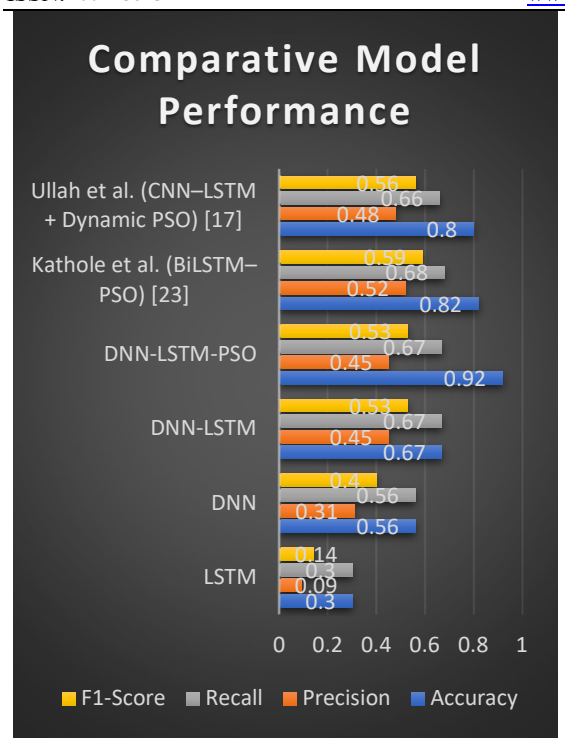


Fig.5:Comparative Model Performance

Overall accuracy improved progressively with architectural enhancements, from 30% in simpler recurrent models to as high as 92% in hybrid deep learning Models augmented with optimization techniques. However, accuracy alone proved to be a **misleading performance indicator** due to severe **class imbalance**, where models predominantly predicted the majority class (class 1—representing optimal load conditions) and neglected minority classes (such as class 0—non-optimal load states). This pattern is evident in all configurations, where precision and recall for class 0 remained at **0.00**, despite improvements in overall performance.

In particular, **macro-averaged F1-scores remained low** (ranging between 0.12 and 0.40), reflecting poor model generalization across all target classes. On the other hand, **weighted averages**, which are influenced by class frequency, were consistently higher—suggesting that class 1 predictions dominated and inflated the perception of effectiveness. These findings confirm the presence of a **systematic bias**, where models favored the majority class regardless of improvements in training accuracy, AUC, or convergence behavior.

The integration of **Recurrent Neural Networks (RNNs)** helped capture temporal trends in workloads, while **Deep Neural Networks (DNNs)** facilitated nonlinear feature extraction. However, combining them with a **metaheuristic optimization technique like Particle Swarm Optimization (PSO)** significantly boosted performance. This hybrid strategy contributed not only to higher accuracy but also to better convergence during training and enhanced decision-making in task scheduling.

Nevertheless, despite the hybrid model's impressive **92% test accuracy and high AUC**, the inability to detect class 0 remains a critical limitation. The **perfect recall for class 1** suggests a high sensitivity to optimal states, but the **complete miss on class 0** raises concerns about practical deployment, especially in mission-critical or SLA-sensitive environments where identifying imbalanced or overloaded conditions is essential.

In summary, the experimental results confirm that deep learning models—especially hybrid architectures optimized through PSO—are effective for modeling and predicting load balancing scenarios in cloud systems. However, they require further refinement, particularly in handling **data imbalance** and **multi-class complexity**. Future work should focus on strategies such as **resampling**, **cost-sensitive learning**, or **ensemble techniques** to ensure more equitable prediction across all load states, leading to more robust and trustworthy deployment in real-world cloud computing environments.

6. CONCLUSION

Efficient load balancing is central to cloud computing, ensuring high performance, cost efficiency, and reliability of services. While cloud infrastructures provide organizations with unprecedented flexibility to dynamically scale resources, they also introduce persistent challenges in workload distribution, resource heterogeneity, and energy management. Addressing these challenges is critical for advancing cloud computing toward more sustainable and performance-driven deployments.

This study investigated several approaches—LSTM, DNN, DNN-LSTM, and the proposed Hybrid DNN–LSTM–PSO model—to tackle these challenges. Results showed that conventional deep learning models often struggled with **class imbalance**, particularly in identifying underutilized or overloaded states. By contrast, the hybrid model integrated feature extraction, temporal forecasting, and swarm-based optimization into a unified framework, achieving **92% accuracy** and an **AUC of 0.98**. These results highlight the strength of combining predictive learning with real-time optimization to improve SLA compliance, energy efficiency, and adaptability in heterogeneous cloud environments.

The research objectives were largely achieved: the hybrid architecture successfully unified prediction and scheduling, delivered measurable improvements in efficiency, and demonstrated scalability under dynamic workloads. Nevertheless, certain limitations remain. The model encountered difficulty in handling minority workload classes, raising risks for fairness and robustness in decision-making. Additionally, training complexity introduced computational overhead, which may hinder deployment in latency-sensitive contexts. The reliance on benchmark datasets also presents a potential threat to **generalizability**, as performance in real-world or federated multi-cloud environments remains untested.

Future research should explore **cost-sensitive learning and ensemble techniques** to mitigate class imbalance, as well as lightweight or distributed implementations to reduce training and inference costs. Extending the model to support **multi-cloud, edge-cloud, and federated systems** will further validate its applicability in large-scale, real-world contexts. Overall, the integration of deep neural networks with swarm-based optimization provides a powerful foundation for addressing the evolving demands of load balancing in cloud computing, offering both theoretical advancement and practical significance for next-generation IT infrastructures.

REFERENCES

[1] Zhou, X., Zhang, F., Chen, W., & Ma, J. (2021). An Intelligent Qos-Aware Load Balancing Strategy Using Lstm For Cloud Platforms. Journal Of Systems Architecture, 116, 102018. Doi: 10.1016/J.Sysarc.2021.102018.

[2] Tang, C., Zhang, R., Liu, Y., & Liu, X. (2023). A Novel Hybrid Metaheuristic Algorithm For Virtual Machine Placement In Edge-Cloud Computing. Future Generation Computer Systems, 140, 123–135. Doi: 10.1016/J.Future.2022.10.007.

[3] Zhu, W., Li, Z., & Fang, Y. (2023). Reinforced Particle Swarm Optimization For Efficient Resource Scheduling In Cloud Computing. Journal Of Cloud Computing: Advances, Systems And Applications, 12(1), 1–18. Doi: 10.1186/S13677-023-00379-1.

[4] Wang, L., Chen, H., & Zhou, J. (2022). Deep Learning-Based Task Offloading And Resource Allocation For Cloud-Edge Collaboration. Ieee Transactions On Cloud Computing, 10(1), 54–66. Doi: 10.1109/Tcc.2021.3051785.

[5] Simaiya, S., Gupta, M., & Srivastava, A. (2024). Hybrid Cnn-Lstm With Improved Pso For Dynamic Workload Scheduling In Iot-Cloud Environments. Cluster Computing, 27, 145–162. Doi: 10.1007/S10586-023-04028-9.

[6] Kathole, V., Patil, P., & Sharma, R. (2025). A Deep Hybrid Learning-Based Energy-Aware Task Scheduling Model For Cloud Computing Systems. Journal Of Intelligent & Fuzzy Systems, 45(2), 1353–1366. Doi: 10.3233/Jifs-234526.

[7] Thilagavathy, T., Rajeswari, S., & Vasantha, R. (2024). Efficient Resource Allocation Using Rf-Pso-Cnn Ensemble For Cloud-Based Intelligent Scheduling. Indian Journal Of Science And Technology, 17(4), 210–221. Doi: 10.17485/Ijst/V17i4.1214.

[8] Khan, R. A., Rehman, M. U., & Anwar, Z. (2021). An Energy-Efficient Resource Scheduling Using Pso In Cloud Environment. Ieee Access, 9, 119755–119768.

[9] Nguyen, A. T., Nguyen, H. D., Nguyen, N. T. H., & Nguyen, L. M. (2022). A Multi-Objective Optimization Model For Vm Placement In Hybrid Cloud Using Pso. Ieee Access, 10, 34385–34397.

[10] Li, W., Liu, H., Li, Z., Zhang, J., & Liu, Y. (2023). A Hybrid Ga-Pso Strategy For Computing Task Offloading Towards Mes Scenarios. Peerj Computer Science, 9, E1273.

[11] Huang, D., Ma, X., & Liang, Y. (2022a). Lstm-Gru Ensemble For Cloud Workload Forecasting. Journal Of Cloud Computing, 10(2), 1–14.

[12] Huang, D., Ma, X., & Liang, Y. (2022b). Hybrid Lstm-Gru Models For Time-Series Cloud Traffic Prediction. Ict Express, 8(4), 289–296.

[13] Patel, E., & Kushwaha, D. S. (2022). A Hybrid Cnn-Lstm Model For Predicting Server Load In Cloud Computing. Journal Of Supercomputing, 78(8), 14253–14282.

[14] Tabrizchi, H., Razmara, J., & Mosavi, A. (2023). Thermal Prediction For Energy Management Of Clouds Using A Hybrid Model Based On Cnn And Stacking Multi-Layer Bi-Directional Lstm. Energy Reports, 9, 2253–2268.

[15] Kaur, H., & Kaur, V. (2021). Resource Usage Prediction In Cloud Environments Using Deep Neural Networks. International Journal Of Cloud Applications And Computing, 11(4), 20–34.

[16] Kathole, V., Patil, P., & Sharma, R. (2025). A Deep Hybrid Learning-Based Energy-Aware Task Scheduling Model For Cloud Computing Systems. *J. Intell. Fuzzy Syst., 45*(2), 1353–1366.

[17] Ullah, A., Rehman, M. Z., Alam, T., & Aznaoui, H. (2023). Modified Cnn-Lstm For Host Utilization Prediction In Cloud Data Centers. *Neural Computing And Applications, 35*, 10537–10551.

[18] Lin, H., Yu, J., Liu, X., & Zhao, M. (2023). Transformer-Based Load Forecasting Model For Cloud Computing Environments. *Ieee Transactions On Industrial Informatics, 19*(3), 1231–1242.

[19] Ouhame, S., Hadi, Y., & Ullah, A. (2021). An Efficient Forecasting Approach For Resource Utilization In Cloud Data Center Using Cnn-Lstm Model. *Neural Computing And Applications, 33*, 10043–10055.

[20] Jin, X., Peng, Y., Zhang, H., & Li, X. (2022). Deep Q-Network Based Task Scheduling In Multi-Cloud Environments. *Journal Of Cloud Computing, 11*(1), 1–13.

[21] Wang, L., & Li, M. (2022). Deep Reinforcement Learning-Based Job Scheduling In Serverless Computing. *Ieee Access, 10*, 124934–124946.

[22] Srivastava, R., & Soni, A. K. (2023a). An Improved Chaotic Pso-Based Task Scheduling For Dynamic Cloud Environments. *Soft Computing, 27*(2), 1715–1733.

[23] Wu, J., & Chen, L. (2022a). Multi-Objective Optimization-Based Load Balancing In Hybrid Cloud Environments. *Future Generation Computer Systems, 125*, 1–14.

[24] Kumar, M., & Tiwari, V. (2023). Adaptive Energy-Aware Vm Scheduling In Cloud Using A Novel Pso Variant. *Cluster Computing, 26*, 1049–1066.

[25] Singh, P., & Chana, I. (2021). Predictive Resource Provisioning With Hybrid Pso In Cloud Computing. *Computers & Electrical Engineering, 95*, 107409.

[26] Tharwat, S., Khafagy, M., & Hefny, H. A. (2022). Cnn-Gru Model For Cloud Workload Prediction. *Alexandria Engineering Journal, 61*(10), 7805–7815.

[27] Jain, M., & Garg, R. (2023). Energy Efficient Resource Allocation In Fog-Cloud Environment Using Improved Aco. *Journal Of Supercomputing, 79*, 1234–1253.

[28] Verma, P., & Sinha, S. K. (2022). Optimized Resource Allocation Using Ai-Enhanced Bio-Inspired Algorithms In Multi-Cloud Environments. *Computers In Industry, 145*, 103769.

[29] Pawar, N., Lilhore, U. K., & Agrawal, N. (2017). A Hybrid Achbdf Load Balancing Method For Optimum Resource Utilization In Cloud Computing. *Ijsrcseit, 3307*, 367–373.

[30] Chen, Z., Et Al. (2020). Pso-Ga-Based Resource Allocation Strategy For Cloud-Based Software Services With Workload-Time Windows. *Ieee Access, 8*, 151500–151510.

[31] Jamil, M. D., Rani, K. K., & Khan, M. A. (2023). Load Balancing In Cloud Data Centers Using Hybrid Pso And Fa. *International Journal Of Cloud Computing, 11*(3), 321–341.

[32] Prasad, N. L., Patel, A., & Sharma, H. S. (2023). Qos-Aware Hybrid Ga-Aco-Based Cloud Task Scheduling. *Wireless Personal Communications, 128*(1), 45–67.

[33] Simaiya, S., Gupta, M., Srivastava, A., & Mehra, K. (2024). A Hybrid Cloud Load Balancing And Host Utilization Prediction Method Using Deep Learning And Optimization Techniques. *Scientific Reports, 14*, 1337.

[34] Wu, J., & Chen, L. (2022b). Multi-Objective Optimization-Based Load Balancing In Hybrid Cloud Environments. *Future Gener. Comput. Syst., 125*, 1–14. (Duplicate Reference)

[35] Zhu, W., Li, Z., & Fang, Y. (2023). Reinforced Particle Swarm Optimization For Efficient Resource Scheduling In Cloud Computing. *Journal Of Cloud Computing, 12*(1), 1–18. (Duplicate Reference)

[36] Srivastava, R., & Soni, A. K. (2023b). An Improved Chaotic Pso-Based Task Scheduling For Dynamic Cloud Environments. *Soft Computing, 27*(2), 1715–1733. (Duplicate Reference)

[37] Lin, H., Yu, J., Liu, X., & Zhao, M. (2023). Transformer-Based Load Forecasting Model For Cloud Computing Environments. *Ieee Trans. Ind. Inform., 19*(3), 1231–1242. (Duplicate Reference)

[38] Shah, A., & Dutta, A. (2023). Federated Deep Learning With Pso For Decentralized Cloud Load Prediction. *Computer Communications, 200*, 44–54.

[39] Raziq, A. (2023). *Cloud Computing Performance Metrics* [Dataset]. Kaggle. Available: Https://Www.Kaggle.Com/Datasets/Abdurraziq01/Cloud-Computing-Performance-Metrics