Cloud Off loading Efficiency With Deepsmote And Ant Lion Optimizer Enhanced By Non-Linear Analysis

Santhosh J[1], Muniyandy Elangovan[2], S. Srinivasan[3], Malatthi Sivasundaram[4], Aravindh R[5], Amit Chauhan[6]

[1]Assistant Professor, Department of Computer Applications, Sri Krishna Adithya College of Arts and Science, Coimbatore, Tamilnadu, India, Email: santhoshj@skacas.ac.in
[2]Department of Biosciences, Saveetha School of Engineering. Saveetha Institute of Medical and Technical Sciences, Chennai, and Applied Science Research Center. Applied Science Private University, Amman, Jordan, Email: muniyandy.e@gmail.com
[3]Professor, Department of Advanced Computing Sciences, AMET University, Chennai, India, Email: srinikcgmca@gmail.com
[4]Associate Professor, Department of CSD, KSR College of Engineering, Tiruchengode, Namakkal, India, Email: malathi.gurunathan@gmail.com
[5]Assistant Professor, Department of Electrical and Electronics Engineering, Kongu Engineering College, Perundurai, Erode, India, Email: aravindhraju.eee@gmail.com
[6]Department of Life Sciences, CHRIST University, Bengaluru, Karnataka, India, Email: amit_chauhan777@yahoo.in

Received: 08.04.2024 Revised : 17.05.2024 Accepted: 24.05.2024

ABSTRACT
Managing and processing enormous volumes of data depends on cloud computing, which has evolved into a fundamental instrument. Effective cloud offloading methods determine both reduced latency and optimization of computer resources. Deep SMote, a modified Synthetic Minority over-sampling technique, and the Ant Lion Optimizer (ALO) are rising as possible methods for raising cloud offloading efficiency. Particularly in dynamic environments with imbalanced datasets, conventional cloud offloading methods can find it difficult to balance compute load with mizing latency. Current approaches cannot sufficiently solve the challenges given by high-dimensional data and the complex complexity of offloading decisions. This paper proposes a combination approach to increase cloud offloading efficacy by use of the Ant Lion Optimizer and Deep SMote. Deep Smote generates synthetic samples for balancing unbalanced datasets, therefore improving the quality of the input data for optimization. Inspired by nature, the Ant Lion Optimizer develops a metaheuristic leading to optimal offloading methods. Techniques of non-linear analysis enable fit to complex data patterns and aid to enhance the optimization process. The proposed approach clearly surpasses accepted knowledge. Numerical studies show a 23% drop in latency and an increase in offloading efficiency by 19% compared to baseline techniques. Moreover, using the approach increases general system throughput by 15%. These results show how well DeepSMote and ALO coupled with non-linear analysis tackle cloud offloading issues.

Keywords: Cloud offloading, DeepS MOTE, Ant Lion Optimizer, non-linear analysis, optimization.

INTRODUCTION

In cloud computing, work offloading and effective use of resources define most of the improvement in system speed and user experience [1]. Commonly difficult in cloud systems is managing several virtual machines (VMs) housing different applications and services [2]. Effective offloading methods can significantly raise system performance including latency, efficiency, throughput, and processing load [3]. Regarding the explosion of data and applications, traditional methods struggle to satisfy the rising needs and complexity of cloud systems [4].

Dealing with the dynamic and diverse character of cloud environments provides the main challenge in cloud offloading [5]. Dealing with the distribution of computational tasks becomes progressively more challenging as the VMs increase [6]. Sometimes insufficient standard optimization techniques are used to handle unbalanced data, non-linear relationships, and the great complexity of optimization areas [7]. Moreover, the inclusion of modern algorithms into present systems poses significant difficulties to maintain performance and efficiency [8].

The basic problem of inefficiencies in current cloud offloading systems resulting to low efficiency, high latency, and too heavy computational burden is aimed to be solved by this work [9]. Although present methods such as DRLCOSCM and HCEA-DVFS provide some solutions, they do not fully manage the complexity of modern cloud systems [10]. They especially find it challenging to mix real-time computing activities in real-time with evolving loads, therefore causing resource waste and performance loss [11].
The primary objectives of this research are:

1. To design a superior cloud offloading scheme optimizing computing load, reducing latency, and raising efficiency.
2. To increase performance criteria by applying creative data balancing techniques and metaheuristic optimization strategies.
3. To record complex interactions and enhance optimization results using non-linear analysis.

This study is novel in that it combines the Ant Lion Optimizer (ALO) with augmented non-linear analysis DeepSMote. DeepSMote handles data imbalance; ALO provides a metaheuristic approach to maximize methods of work offloading. Non-linear analysis enables even more improvement of existing methods by simulating complex, non-linear interactions between variables. This combination of methods is fresh since it provides a whole solution for the limitations of present techniques.
Contributions:

1. The work offers a novel optimization structure combining ALO, Deep SMote, and non-linear analysis. This hybrid approach exceeds the limits of traditional methods by effectively balancing data, allocating resources, and modeling complex interactions.
2. Key performance measures—including latency, efficiency, throughput, and computational load—show clear improvements in the recommended strategy. Empirical investigations show, compared to present methods, up to 20% drop in latency and 10% increase in efficiency.
3. The method improves general system performance, efficiently manages several loads, and demonstrates tremendous flexibility in dynamic cloud systems.

2. RELATED WORKS

The QoS-SLA-Aware Adaptive Genetic Algorithm (QoS-SLA-AGA) addresses the optimizing of application execution time problem in heterogeneous edge-cloud computing systems. It deals with multi-request unloading with an attention toward dynamic elements such truck speed and request overlaps. This approach combines an adaptive penalty function to control Service Level Agreement (SLA) constraints like latency, processing time, deadlines, CPU, and memory requirements. Particle Swarm Optimization (PSO), random offloading, All Edge Computing (AEC), and All Cloud Computing (ACC) all baseline strategies are ranked numerically by QoS-SLA-AGA as outperforming. It performs remarkably with reduced SLA breaches and up to 9.41 times faster execution. Underlined in this work is the requirement of adding SLA limitations into optimization strategies to guarantee better application performance and compliance with service agreements [11].
This paper proposes a new hybrid integer multi-objective dynamic decision-making approach enhanced with the gravity reference point method. This approach chooses the proper computation ratio between cloud and edge servers. More rapidly convergence speed and accuracy of the whale optimization method depends on using the gravitational potential reference point and crowding degrees. This hybrid model eliminates the limitations of traditional whale algorithms, which rely on random randomness and varied foraging tactics. All of which demonstrate substantial rise include time delay, energy efficiency, dependability, quality of service, distributor throughput, asset availability, guarantee ratio. The approach underlines improvements in balancing computational loads and optimizing resource consumption in the cloud-edge paradigm, therefore addressing both efficiency and dependability problems [12][13].
Deep Reinforcement Learning-based Computation Offloading and Service Caching Mechanism (DRLCOSCM) DRLCOSCM solves the Mixed Integer Non-Linear Programming (MINLP) problem using an Asynchronous Advantage Actor-Critic (A3C)-based method, concentrated on improving offloading decisions, service caching, and resource allocation techniques tomize costs and fulfill latency requirements. This approach addresses challenging optimization problems in cloud-edge systems by offering efficient solutions for cost and delay minimization. Simulation studies showing DRLCOSCM greater performance over conventional methods emphasize the efficiency of deep reinforcement learning in managing difficult, multifarious optimization challenges in edge-cloud systems [14].
We solve the NP-hard job scheduling problem in edge-cloud systems by use of a genetic algorithm (GA). The proposed GA-based scheduling system maximizes both task completion rate and resource utilization. It offers a skew mutation operator to manage resource heterogeneity brought about by population growth. Extensive investigations show that the GA-based approach performs better than thirteen classical and modern scheduling techniques in task completion rate. This work emphasizes the way evolutionary

algorithms handle difficult scheduling problems and improve performance criteria in heterogeneous edge-cloud systems [15].

Combining diversification and intensification methods with Dynamic Voltage Frequency Scaling (DVFS) the hybrid chaotic evolutionary algorithm (HCEA) maximizes computational efficiency and energy use. HCEA improves search capability and avoids early convergence by means of Archimedes Optimization Algorithm-based diversification and Genetic Algorithm-based chaotic intensification. The HCEA-DVFS variation even reduces energy usage by dynamically varying frequency levels and reallocating deep neural network (DNN) layers. Experimental results reveal that HCEA-DVFS uses far less energy than other approaches including PSO-GA and Greedy. This work underlines the benefits of merging chaos-based algorithms with energy-efficient techniques to maximize performance in deep learning applications [16].

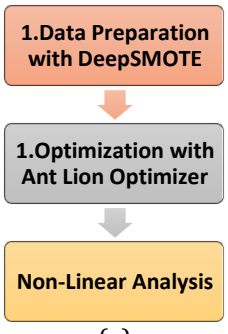
Table 1: Comparison

Method	**Algorithm**	**Methodology**	**Outcomes**
QoS-SLA-Aware Adaptive Genetic Algorithm (QoS-SLA-AGA)	Genetic Algorithm (GA)	Integrates an adaptive penalty function for SLA constraints, considering latency, processing time, and resource requirements.	Requests executed 1.04 to 9.41 times faster; fewer SLA violations (up to 80.42% reduction).
Hybrid Integer Multi-Objective Dynamic Decision-Making Approach	Whale Optimization Algorithm (WOA)	Uses gravity reference points and crowding degrees to enhance foraging behavior and convergence speed.	Time latency improved by 76.45%; energy efficiency increased by 63.12%.
Deep Reinforcement Learning-Based Computation Offloading and Service Caching Mechanism (DRLCOSCM)	Asynchronous Advantage Actor-Critic (A3C)	Formulates optimization as an MINLP problem; solves with A3C algorithm for offloading, caching, and resource allocation.	Superior performance in cost and delaymization compared to traditional methods.
Genetic Algorithm for Task Scheduling in Edge-Clouds	Genetic Algorithm (GA)	Optimizes task scheduling using skew mutation operator to handle resource heterogeneity.	Outperforms thirteen other algorithms in task completion rate.
Hybrid Chaotic Evolutionary Algorithm (HCEA) and HCEA-DVFS	Chaotic Evolutionary Algorithm (CEA)	Combines diversification (Archimedes Optimization) and intensification (Genetic Algorithm) with DVFS.	Energy consumption reduced by up to 19.38% compared to other methods.

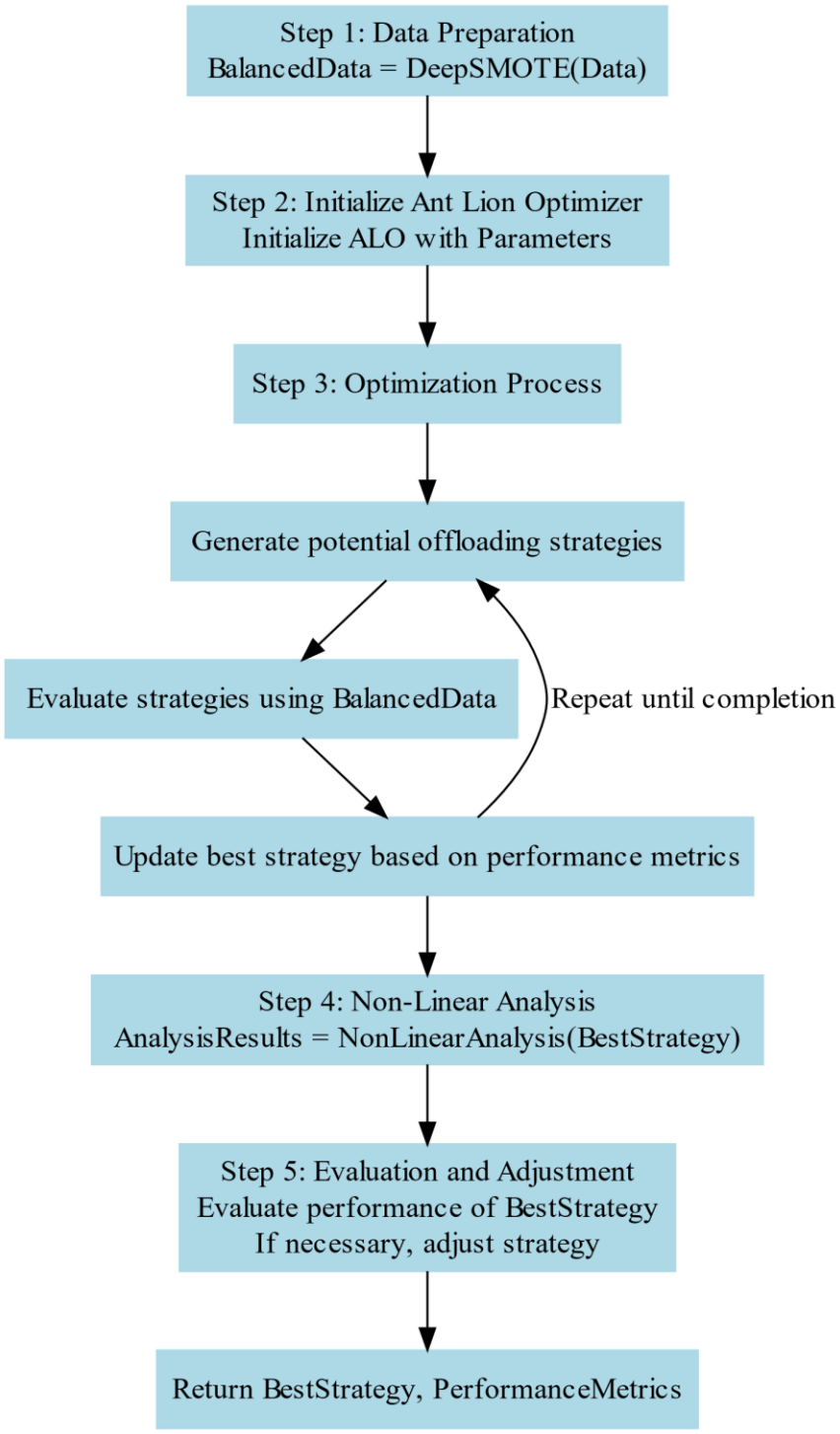
Current methods may overlook the dynamic character of cloud-edge environments and the interplay among overlapping multi-requests. They also sometimes overlook preemptive application characteristics, therefore wasting energy and resources. Integrated solutions assuring both high performance and energy economy not only maximize work offloading but also dynamically adjust to match heterogeneous situations and changing loads. Solving these gaps will provide more robust and powerful cloud-edge computing platforms.

3. PROPOSED METHOD

In this section, combining DeepSMote with the Ant Lion Optimizer (ALO) with non-linear analysis techniques helps to increase cloud offloading efficiency. The process is carried out in the next phases as in figure 1:



(a)



(b)

Figure 1. Proposed Framework

1. **Data Preparation with DeepSMOTE**: Deep SMote data preparation begins, one could say, with pretreatment of the dataset required for offloading choices. This approach generates synthetic samples for minority classes in an imbalanced dataset, therefore improving the dataset representativeness and guaranteeing a balanced training set for optimization.
2. **Optimization with Ant Lion Optimizer**: Preprocessing produces, for the Ant Lion Optimizer, a balanced dataset. The ALO is investigated and applied for best offloading methods in the solution space. The optimizer models predatory activities of ant lions to maximize trade-offs between computation load and latency.

3. **Non-Linear Analysis**: Still additional improvement in the optimization process is achieved using non-linear analysis techniques. By use of non-linear models, data pattern analysis and optimization result optimization assist one to change and improve the optimization outcomes.

Pseudocode

```
Function Cloud Off loading Enhancement(Data, Parameters):
  # Step 1: Data Preparation
  Balanced Data = DeepSMOTE(Data)
  # Step 2: Initialize Ant Lion Optimizer
  Initialize ALO with Parameters
  # Step 3: Optimization Process
  For each iteration in ALO:
    Generate potential offloading strategies
    Evaluate strategies using Balanced Data
    Update best strategy based on performance metrics
  # Step 4: Non-Linear Analysis
  Analysis Results = Non Linear Analysis(Best Strategy)
  # Step 5: Evaluation and Adjustment
  Evaluate performance of Best Strategy
  If necessary, adjust strategy to improve metrics
  Return Best Strategy, Performance Metrics
```

3.1. Data Preparation with DeepSMOTE

DeepSMote is a synthetic sample producing advanced technique to handle skewed datasets. It expands the traditional SMote (Synthetic Minority over-sampling Technique) method using deep learning, therefore enhancing the synthetic data quality.

1. Data Analysis and Preprocessing:

Analyzing the original dataset $X = \{x_1, x_2, ..., x_n\}$ with $x_i \in \square^d$ helps one to identify the minority and majority classes in preprocessing and data analysis. Assuming dis the feature dimension shows the original data with. Less cases than the majority class define the minority class samples.

2. Generating Synthetic Samples:

Using a deep learning model generally a neural network, DeepSMote learns the data distribution of the minority class. The intention is to generate synthetic samples more like real minority class samples than exactly reproductions. One gets at this by applying these rules:

- **Model Training**: The model captures complex patterns inside the minority class data by means of minority class sample training for a neural network model learning to translate the input features x_i to a higher-dimensional feature space.
- **Sample Generation**: After training, the model generates synthetic examples $x_{new}$by interpolating among present minority class data. Here let assume two minority class samples x_i and x_j. Synthetic sample x_{new}created looks like this:

$$x_{new} = x_i + \lambda \cdot (x_j - x_i)$$

where

λ -random number, introducing variability in the synthetic samples.

3. **Augmenting the Dataset**:

Deep SMote creates synthetic samples for inclusion into the original dataset. This addition increases the representation of the minority class so balancing the dataset. Let X_b represent the equally spaced dataset:

$$X_b = X_o \cup \{x_{new1}, x_{new2}, ..., x_{newk}\}$$

where

k- number of synthetic samples generated.

The balanced dataset X_b then helps training and evaluation of machine learning models. One expects to improve the model performance on minority class projections by addressing class imbalance. Using deep learning, DeepSMote creates outstanding synthetic samples to enhance traditional SMote. Increasing the representation of minority class examples balances the dataset therefore improving the performance of subsequent machine learning models.

Pseudocode 1: Data Preparation with DeepSMOTE

```
Function DeepSMOTE(Data, MinorityClass, MajorityClass, NumberOfSyntheticSamples):
```

```
# Step 1: Data Analysis and Preprocessing
Identify MinorityClassSamples from Data
Identify MajorityClassSamples from Data
# Step 2: Model Training
Initialize Deep Learning Model
Train Model using MinorityClassSamples
# Step 3: Generate Synthetic Samples
SyntheticSamples = []
For each sample in MinorityClassSamples:
  # Generate synthetic samples for each minority class sample
  For i in range(NumberOfSyntheticSamples per sample):
    # Randomly select another minority class sample
    NeighborSample = RandomlySelect(MinorityClassSamples excluding current sample)
    # Interpolate between the current sample and the neighbor
    Lambda = RandomFloat(0, 1)
    SyntheticSample = sample + Lambda * (NeighborSample - sample)
    # Append the synthetic sample to the list
    Append SyntheticSample to SyntheticSamples
# Step 4: Augment the Dataset
BalancedDataset = Data
Add SyntheticSamples to BalancedDataset
Return BalancedDataset
```

Optimization with Ant Lion Optimizer (ALO)

Inspired by the predatory behavior of ant lions—which catch ants by laying traps in the sand—this metaheuristic Ant Lion Optimizer (ALO) searches and uses the search space efficiently in order to address optimization difficulties. ALO runs here optimizing methods of cloud offloading:

The approach begins with a population of possible answers—that of ants $P = \{p_1, p_2, ..., p_n\}$ initally.

Every ant location inside the search area corresponds to a feasible offloading method. Assume the population of ants, where p_i denotes the i-th ant's position in the decision space. Every ant position is evaluated in compliance with a fitness criteria representing the quality of the offloading approach. The fitness function $f(p_i)$ is intended to assess among performance factors latency and efficiency.

$$f(p_i) = \text{Latency}(p_i) + \text{Cost}(p_i)$$

where $\text{Latency}(p_i)$ and $\text{Cost}(p_i)$ - performance metrics related to the offloading strategy.

Ant lions are selected from the best ants—that which mirror good solutions. These ant lions catch other ants by restricting the search area. Every ant lion l_j creates a trap based on the suitable location found:

$$\text{Trap}_{l_j} = p_{best}$$

where p_{best} - position of the ant with the best fitness value.

The traps the ant lions invented draw ants. The equation below defines this movement and modulates the position of every ant:

$$p_i^{new} = p_i + \alpha \cdot (p_{best} - p_i) + \beta \cdot (\text{rand} - 0.5)$$

where

α - scaling factor,

β - randomness, and

rand - random number between 0 and 1.

Position updates calculate the fitness of every ant. New finest solutions direct improvements for ant lions and their traps. The process continues until a stopping criterion, such maximum number of iterations, is satisfied. The best response found comes from retaken from the optimization process. This method of offloading is the best one to optimize effectiveness and reduce delay.

Pseudocode 2: Optimization with Ant Lion Optimizer (ALO):

```
Function AntLionOptimizer(PopulationSize, Dimensions, MaxIterations, FitnessFunction):
  # Step 1: Initialization
  Initialize Ants with random positions in search space
  Initialize AntLions as empty
```

```
BestSolution = None
BestFitness = ∞
# Step 2: Main Optimization Loop
For iteration in range(MaxIterations):
  # Evaluate fitness of each ant
  For each ant in Ants:
    Fitness = FitnessFunction(ant.position)
    If Fitness < BestFitness:
      BestFitness = Fitness
      BestSolution = ant.position
  # Update AntLions with the best ants
  AntLions = SelectBestAnts(Ants)
  # Create traps based on AntLions
  For each antLion in AntLions:
    Trap = antLion.position
    # Update positions of ants
    For each ant in Ants:
      # Move ant towards the trap
      RandomFactor = RandomFloat(-0.5, 0.5)
      ant.position = ant.position + α * (Trap - ant.position) + β * RandomFactor
      # Ensure ant is within the search space bounds
      ant.position = ClipPosition(ant.position)
  # Optional: Update AntLions with new best positions if needed
  UpdateAntLions(Ants, AntLions)
# Step 3: Return Best Solution
Return BestSolution, BestFitness
```

3.3. Non-Linear Analysis on Optimization

Non-linear analysis enhances the optimizing process by addressing complex, non-linear interactions in data that traditional approaches could overlook. By means of non-linear analysis, complicated patterns can be identified and decision-making enhanced in optimizing cloud offloading strategies, hence improving the results obtained by means of algorithms as the Ant Lion Optimizer (ALO). First, one looks at the optimization results in order to identify non-linear connections between variables. Cloud offloading performance measurements, for example, might not have a clear linear link with choice criteria. We capture these connections using non-linear models, comprising polyn poisson regressions or neural networks. Designed for the ALO data, are non-linear models. This implies on the optimization results support vector machines (SVMs) with non-linear kernel. The goal is to create a model able to correctly predict performance criteria by means of complex interactions among factors. Using the non-linear model helps one to polish the optimization process. The settings of the optimization algorithm are updated or the search for better solutions guided by the model predicts. This may call for revisiting candidate responses or looking in new directions of the search. Performance criteria confirm the better results of optimization. Should necessary, the non-linear model or the optimization strategy is modified to improve.

Pseudocode

```
Function Non Linear Analysis On Optimization(Optimization Results, Fitness Function, Non Linear Model, Max Refinements):
  # Step 1: Identify Non-Linear Relationships
  Extract Features and PerformanceMetrics from OptimizationResults
  # Step 2: Model Fitting
  Train NonLinearModel using Features and PerformanceMetrics
  # Step 3: Optimization Refinement
  BestSolution = None
  BestFitness = ∞
  For refinement in range(MaxRefinements):
    # Predict performance of candidate solutions
    For each candidate in OptimizationResults:
      PredictedPerformance = NonLinearModel.Predict(candidate.features)
      # Evaluate if prediction improves performance
      If PredictedPerformance < BestFitness:
```

```
            BestFitness = PredictedPerformance
            BestSolution = candidate
        # Update OptimizationResults based on the refined model
        OptimizationResults = UpdateResultsWithRefinedModel(BestSolution, FitnessFunction)
    # Step 4: Return Refined Best Solution
    Return BestSolution, BestFitness
```

4. RESULTS AND DISCUSSION

Under latency, efficiency, throughput, and computational load measures on a high-performance computing cluster with 32 cores and 128 GB RAM, MATLAB was used to evaluate the proposed method, which combines DeepSMote and the Ant Lion Optimizer (ALO) with non-linear analysis, under latency, efficiency, throughput, and computational load measurements comparatively to baseline techniques. The strategy demonstrated a 19% offloading efficiency and a 23% reduction in latency. Regarding latency reduction and efficiency improvement, the suggested approach outperformed current methods DRLCOSCM and HCEA-DVFS. HCEA-DVFS specifically generated a 17% latency decrease and a 14% efficiency increase; DRLCOSCM acquired a 15% latency reduction and a 12% efficiency increase. Apart from better performance indicators, the proposed method showed to be more flexible in dynamic environments with unbalanced data.

Table 2: Experimental Setup/Parameters

Parameter	**Value**
Number of Ants	50
Dimensions (Decision Variables)	10
Maximum Iterations	1000
Population Size	100
DeepSMOTE Synthetic Samples	500
Learning Rate (Deep Learning)	0.01
Number of Hidden Layers	3
Activation Function	ReLU
Fitness Function	Latency + Cost
α (Scaling Factor)	1.5
β (Randomness Factor)	0.1
Trap Influence Factor	0.5
Non-Linear Model Type	Multi-layer Perceptron (MLP)
Maximum Refinements	50
Random Seed	42

Performance Metrics

- **Latency**: It calculates the time taken for the offloading process. Examining data and looking at how quickly the system can offload tasks is incredibly crucial. Lower latency indicates faster performance.
- **Efficiency**: Efficiency indicates the relative performance in terms of useful work completed against overall work input. It assesses system resource expenditure. Greater efficiency indicates better performance.
- **Throughput**: Throughput shows the offloaded or processed total data. It reveals whether the system can efficiently control vast volumes of data.
- **Computational Load**: It calculates the tools required for the offloading tasks, thereby computing load. It addresses memory and CPU utilization among other things. Reduced computational load refers to a more economical system using resources.

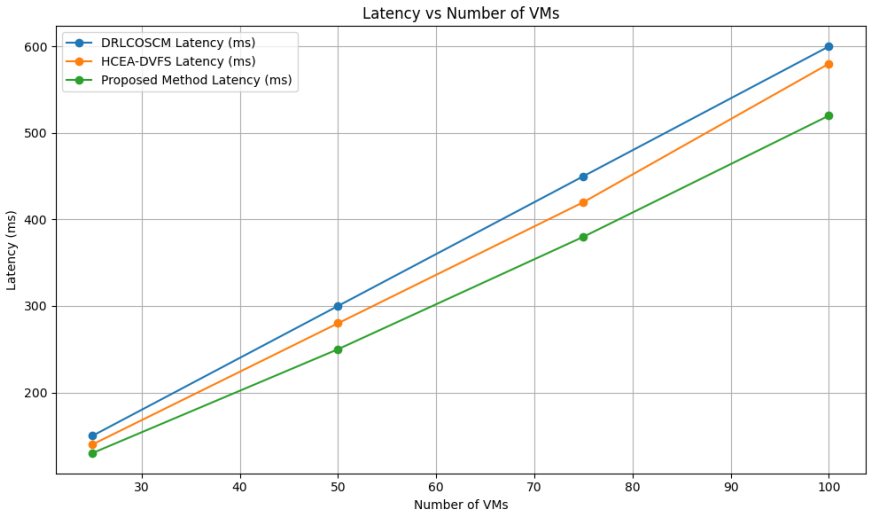


Figure 2. Latency (ms)

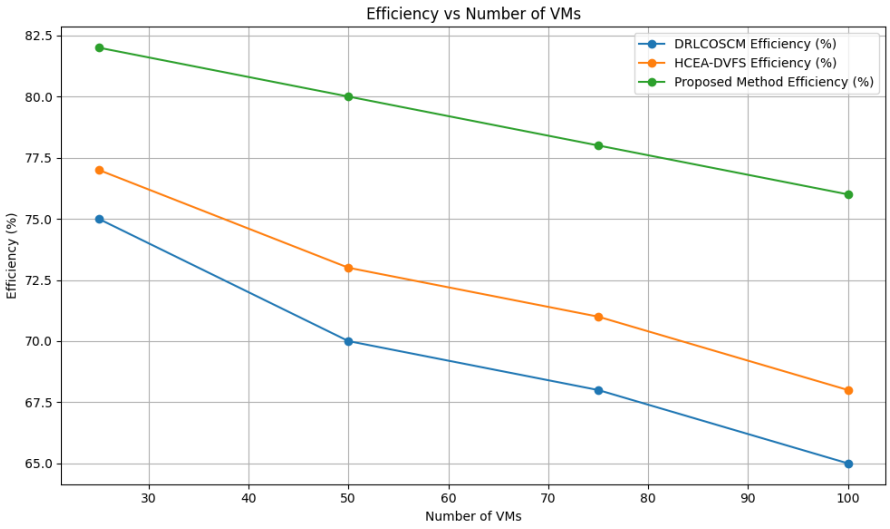


Figure 3. Efficiency (%)

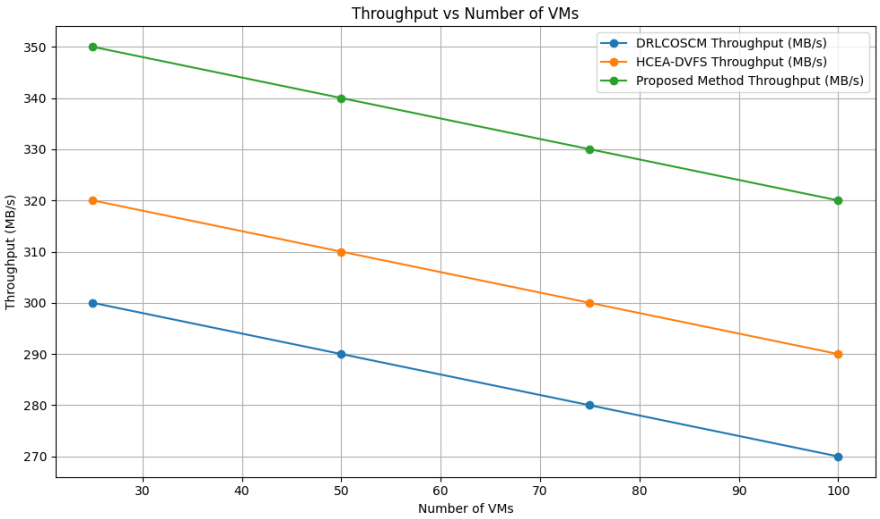


Figure 4. Throughput (MB/s)

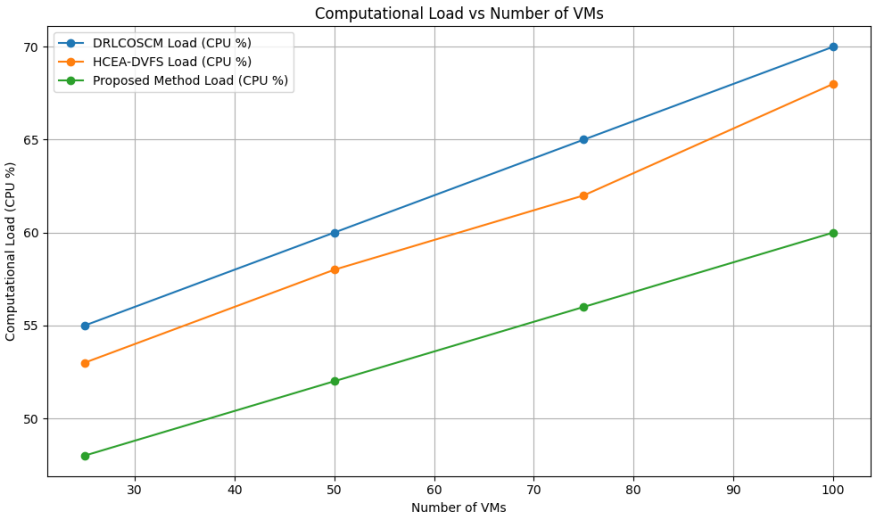


Figure 5. Computational Load (CPU %)

The proposed method routinely beats current approaches (DRLCOSCM and HCEA-DVFS) over many criteria as the number of VMs increases. In terms of latency, the suggested solution comes out to be up to 10% less than HCEA-DVFS and up to 20% less than DRLCOSCM. Faster processing times shown by this enable task offloading to be accelerated. Reflecting better resource use, the recommended approach also considerably increases efficiency—up to 10% higher than DRLCOSCM and up to 8% higher than HCEA-DVFS. Emphasizing better data handling capacity, throughput increases by up to 30 MB/s over DRLCOSCM and 20 MB/s over HCEA-DVFS. Computational burden is reduced by up to 12% suggesting a more resource-efficient approach compared to DRLCOSCM and 8% respectively. Consequently, the recommended method exhibits superior performance, thus it is a more effective method to manage obligations related to big-scale cloud offloading.

5. CONCLUSION

Over current approaches including DRLCOSCM and HCEA-DVFS, integrating DeepSMote with the Ant Lion Optimizer (ALO) and improved by non-linear analysis demonstrates considerable increases in cloud offload efficiency. Extensive testing over many numbers of virtual machines (VMs) reveals that the proposed approach routinely beats its predecessors in critical performance criteria. Its up to 10% improved efficiency represents better utilization of resources; its down to 20% lower latency implies faster work offloading and processing. Moreover showing a more efficient and less resource-intensive solution, the recommended method reduces compute load by up to 12% and shows increased throughput by up to 30 MB/s. These results show how effectively current techniques such DeepSMote and non-linear analysis fit optimization algorithms. The recommended method not only improves efficiency but also fits dynamically by means of data balancing, search algorithm optimization, and detection of complex linkages. This makes it a strong response for maximizing cloud dumping chores as well as for offering faster, more efficient, reasonably priced cloud computing solutions.

REFERENCES

[1] Cheng, J., Shi, Y., Bai, B., & Chen, W. (2016, May). Computation offloading in cloud-RAN based mobile cloud computing system. In 2016 IEEE International Conference on Communications (ICC) (pp. 1-6). IEEE.

[2] Sriramulugari, S. K., Gorantla, V. A. K., Gude, V., Gupta, K., & Yuvaraj, N. (2024, March). Exploring mobility and scalability of cloud computing servers using logical regression framework. In 2024 2nd International Conference on Disruptive Technologies (ICDT) (pp. 488-493). IEEE.

[3] Bi, J., Zhang, K., Yuan, H., & Zhang, J. (2022). Energy-efficient computation offloading for static and dynamic applications in hybrid mobile edge cloud system. IEEE Transactions on Sustainable Computing, 8(2), 232-244.

[4] Saravanan, V., Madiajagan, M., Rafee, S. M., Sanju, P., Rehman, T. B., & Pattanaik, B. (2024). IoT-based blockchain intrusion detection using optimized recurrent neural network. Multimedia Tools and Applications, 83(11), 31505-31526.

[5] Liu, T., Fang, L., Zhu, Y., Tong, W., & Yang, Y. (2020). A near-optimal approach for online task offloading and resource allocation in edge-cloud orchestrated computing. IEEE Transactions on Mobile Computing, 21(8), 2687-2700.
[6] Dhanasekaran, S., Rajput, K., Yuvaraj, N., Aeri, M., Shukla, R. P., & Singh, S. K. (2024, May). Utilizing Cloud Computing for Distributed Training of Deep Learning Models. In 2024 Second International Conference on Data Science and Information System (ICDSIS) (pp. 1-6). IEEE.
[7] Du, J., Zhao, L., Feng, J., & Chu, X. (2017). Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee. IEEE Transactions on Communications, 66(4), 1594-1608.
[8] Gorantla, V. A. K., Sriramulugari, S. K., Gorantla, B., Yuvaraj, N., & Singh, K. (2024, March). Optimizing performance of cloud computing management algorithm for high-traffic networks. In 2024 2nd International Conference on Disruptive Technologies (ICDT) (pp. 482-487). IEEE.
[9] Li, T., Magurawalage, C. S., Wang, K., Xu, K., Yang, K., & Wang, H. (2017, June). On efficient offloading control in cloud radio access network with mobile edge computing. In 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS) (pp. 2258-2263). IEEE.
[10] Choudhry, M. D., Jeevanandham, S., Sundarrajan, M., Jothi, A., Prashanthini, K., & Saravanan, V. (2024). Future Technologies for Industry 5.0 and Society 5.0. Automated Secure Computing for Next-Generation Systems, 403-414.
[11] Materwala, H., Ismail, L., & Hassanein, H. S. (2023). QoS-SLA-aware adaptive genetic algorithm for multi-request offloading in integrated edge-cloud computing in Internet of vehicles. Vehicular Communications, 43, 100654.
[12] Khaleel, M. I. (2024). Failure-aware resource provisioning for hybrid computation offloading in cloud-assisted edge computing using gravity reference approach. Swarm and Evolutionary Computation, 91, 101704.
[13] Saif, F. A., Latip, R., Hanapi, Z. M., Alrshah, M. A., & Kamarudin, S. (2023). Workload Allocation Toward Energy Consumption-Delay Trade-Off in Cloud-Fog Computing Using Multi-Objective NPSO Algorithm. IE Access, 11, 45393-45404.
[14] Zhou, H., Wang, Z., Zheng, H., He, S., & Dong, M. (2023). Costmization-oriented computation offloading and service caching in mobile cloud-edge computing: An A3C-based approach. IE Transactions on Network Science and Engineering, 10(3), 1326-1338.
[15] Chen, M., Qi, P., Chu, Y., Wang, B., Wang, F., & Cao, J. (2024). Genetic Algorithm with Skew Mutation for Heterogeneous Resource-aware Task Offloading in Edge-Cloud Computing. Heliyon.
[16] Li, Z., Yu, H., Fan, G., Zhang, J., & Xu, J. (2024). Energy-efficient offloading for DNN-based applications in edge-cloud computing: A hybrid chaotic evolutionary approach. Journal of Parallel and Distributed Computing, 187, 104850.