Article

Not peer-reviewed version

# RAP-Optimizer: Resource-Aware Predictive Model for Cost Optimization of Cloud AIaaS Applications

Kaushik Sathupadi , Ramya Avula , Arunkumar Velayutham , Sandesh Achar *

Posted Date: 8 October 2024

doi: 10.20944/preprints202410.0504.v1

Keywords: Deep Neural Network, Dynanimc Dropout Control, Overfitting Mitigation, Simulated Annealing, AIaaS, Cloud Resource Optimization, Cost-Efficiency, Resource Utilization, Profit Margin Enlarging.

*Article*

# RAP-Optimizer: Resource-Aware Predictive Model for Cost Optimization of Cloud AIaaS Applications

**Kaushik Sathupadi [1], Ramya Avula [2], Arunkumar Velayutham [3] and Sandesh Achar [4,*]**

1 Staff Engineer, Google LLC, Sunnyvale, CA
2 Business Information Developer Consultant Company, Carelon Research, Celina, TX
3 Cloud Software Development Engineer and Technical Lead at Intel, Arizona, USA
4 Department of Software Engineering, Walmart Global Tech, Sunnyvale, CA, USA
* Correspondence: sandeshachar26@gmail.com;

**Abstract:** AI-driven applications are rapidly growing, and more applications are joining the market competition. As a result, the AI-as-a-Service (AIaaS) model is experiencing rapid growth. Many of these AIaas-based applications are not properly optimized initially. Once they start experiencing a large volume of traffic, different challenges start revealing themselves. One of these challenges is maintaining a profit margin for the sustainability of the AIaaS application-based business model, which depends on the proper utilization of computing resources. This paper introduces the Resource Award Predictive (RAP) model for AIaaS cost optimization called RAP-Optimizer. It is developed by combining a Deep Neural Network (DNN) with the simulated annealing optimization algorithm. It is designed to reduce resource underutilization and minimize the number of active hosts in cloud environments. It dynamically allocates resources and handles API requests efficiently. The RAP-Optimizer reduces the number of active physical hosts by an average of 5 per day, leading to a 45% decrease in server costs. The impact of the RAP-Optimizer has been observed over a 12-month period. The observational data show a significant improvement in resource utilization. It effectively reduces operational costs from $2,600 to $1,250 per month. Furthermore, the RAP-Optimizer increases the profit margin by 179%, from $600 to $1,675 per month. The inclusion of the Dynamic Dropout Control (DDC) algorithm in the DNN training process mitigates overfitting, achieving a 97.48% validation accuracy and a validation loss of 2.82%. These results indicate that the RAP-Optimizer effectively enhances resource management and cost-efficiency in AIaaS application, making it a valuable solution for modern cloud environments.

**Keywords:** deep neural network; dynanimc dropout control; overfitting mitigation; simulated annealing; AIaaS; cloud resource optimization; cost-efficiency; resource utilization; profit margin enlarging

## 1. Introduction

The current cloud application trends demonstrate the rapid growth of AI-driven applications powered by AIaaS as the back-end [1]. As a result, the demand for efficient resource utilization has become paramount [2]. Many innovative cloud-based services, particularly those adopting the AI-as-a-Service (AIaaS) model, suffer from underutilization of resources, leading to escalating operational costs [3]. The initial resource utilization assumptions regarding the resource limitation of these applications change as the number of users and API requests increases. That is why service providers face the challenge of managing their physical and virtual resources effectively while maintaining the Quality of Services (QoS) [4]. The RAP-Optimizer presented in this paper addresses these issues by optimizing resource allocation and minimizing the number of active hosts in AIaaS environments. This system not only reduces server costs but also improves resource efficiency. As a result, it leads to better profit margins and energy savings. The potential of the RAP-Optimizer lies in its ability to dynamically balance API request loads across cloud servers. This is how it prevents the unnecessary activation of additional resources and ensures sustainable and scalable cloud operations.

The proposed RAP-Optimizer has been developed by integrating a Deep Neural Network (DNN) [5] with the Simulated Annealing algorithm [6] to create a robust framework for real-time resource management. The DNN predicts the optimal configuration for virtual machines (VMs) based on real-time data analysis, while the Simulated Annealing algorithm helps optimize resource allocation by minimizing the number of active hosts. Additionally, the system incorporates a Dynamic Dropout Control (DDC) algorithm to mitigate overfitting issues during the model training phase. The RAP-Optimizer operates in a multi-stage workflow, beginning with resource analysis through the Resource Analyzer (RAN) algorithm, which identifies underutilized hosts and redistributes API requests to ensure optimal cloud resource usage. By consolidating workloads and deactivating idle hosts, the system is able to enhance energy efficiency while maintaining the agreed Quality of Service (QoS) for users. The key contributions of this paper are summarized as follows:

- **Dynamic Resource Optimization:** A novel integration of DNN and Simulated Annealing for dynamically balancing API requests and resource utilization across active cloud hosts.
- **Cost Reduction Mechanism:** Demonstrated significant server cost reduction through the RAP-Optimizer, leading to improved profit margins and reduced energy consumption.
- **Multi-Stage Optimization Workflow:** Introduction of a multi-stage workflow utilizing the RAN algorithm for comprehensive resource analysis, ensuring effective redistribution of workloads across physical and virtual machines.
- **Handling Overfitting with DDC:** An innovative Dynamic Dropout Control (DDC) algorithm integrated into the DNN to overcome overfitting during model training and enhance prediction accuracy.
- **Revenue Margin Increase:** The proposed system improved profit margins by 179% over 12 months, increasing the average profit margin from $600 to $1,675.

The remainder of this paper is organized as follows. The literature review has been presented in Section II. Section III provides an in-depth problem analysis, identifying the challenges that led to the development of the RAP-Optimizer. Section IV outlines the methodology, describing the integration of DNN, Simulated Annealing, and the RAN algorithm, along with the dataset preparation and feature normalization steps. Section V presents the experimental results and evaluations, including the performance analysis of the proposed system and its ability to optimize cloud resource usage. Section VI discusses the limitations and potential future improvements for the RAP-Optimizer. Finally, Section VII concludes the paper, summarizing the key findings and contributions.

## 2. Literature Review

Resource optimization in cloud computing has been a widely researched topic, particularly in the context of Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS). However, there remains a significant gap in addressing optimization strategies specifically tailored for AI-as-a-Service (AIaaS) models, which are characterized by fluctuating workloads and high computational demands [7]. This section reviews recent studies on cloud resource optimization, workload balancing, and overfitting issues in deep learning models, identifying gaps that the proposed RAP-Optimizer aims to address.

### *2.1. Cloud Resource Optimization*

According to the survey conducted by Mohammadzadeh et al., [8], resource optimization for traditional cloud services, such as virtual machine (VM) allocation and CPU/RAM resource distribution, is the predominant field of research in cloud resource optimization. AIaaS differs from traditional cloud services, which require real-time, scalable computation [9]. Furthermore, most solutions, like the Hill-Climbing (HC) algorithm, operate reactively and often activate new physical hosts without fully utilizing existing resources, resulting in higher operational costs [10]. The proposed RAP-Optimizer addresses this gap by integrating a DNN with Simulated Annealing [6] to dynamically allocate resources based on real-time workloads and optimize resource utilization.

### *2.2. Workload Balancing and API Request Handling*

Numerous studies have examined workload balancing in cloud environments [11–13]. However, these approaches generally focus on static or semi-dynamic strategies that do not adapt quickly to the rapidly changing workload patterns seen in AIaaS platforms. For example, Kumar et al. [14] presented an approach for balancing workloads across cloud servers but did not consider the possibility of reducing the number of active hosts when resources are underutilized. Additionally, [15] emphasized load distribution based on CPU and memory but did not consider network bandwidth and other critical factors such as disk I/O [16]. In contrast, the RAP-Optimizer efficiently reallocates API requests by utilizing fewer physical hosts while ensuring maximum CPU and memory utilization, thus overcoming the limitations of static workload balancing methods.

### *2.3. Overfitting in Deep Neural Networks*

Handling overfitting in DNNs is an active field of research with numerous regularization techniques [17]. According to Alnagashi [18] et al., dropout is an effective way to mitigate the overfitting issue. A review conducted by Salehin et al. [19] on different dropout techniques reveals that most approaches use a fixed dropout rate. The literature shows that while Dropout can be effective, it is not adaptive to different layers of the network, leading to either underfitting or overfitting in complex applications [20]. This paper fills this gap by introducing the Dynamic Dropout Control (DDC) algorithm, which dynamically adjusts the dropout rate layer-by-layer, reducing the overfitting issue without compromising the model's predictive performance.

### *2.4. Energy-Efficient Cloud Systems*

The optimization of energy consumption in cloud data centers has been explored in various studies [21–23]. The primary focus of these is to reduce energy consumption by consolidating VMs or activating power-saving modes on underutilized hosts [24]. However, the existing methods primarily focus on reducing the number of active physical hosts without considering the trade-off between resource optimization and maintaining service quality [25]. The proposed RAP-Optimizer not only reduces the number of active hosts by an average of five per day, but it also improves resource utilization, resulting in substantial energy savings without degrading Quality of Service (QoS).

### *2.5. Revenue Impact and Cost Optimization*

Revenue impact and corresponding cost optimization is an under-explored field of research for AIaaS [26]. Multiple studies revolve around traditional cloud services [27–29]. However, these studies do not account for the dynamic and unpredictable nature of AIaaS platforms, where the operational cost can escalate quickly due to inefficient resource management [30]. The RAP-Optimizer directly addresses this gap by significantly reducing server costs—by 45% on average—while maintaining consistent service delivery. This allows AIaaS platforms to stabilize their profit margins over time, which is often overlooked in traditional cloud optimization research.

## **3. Problem Analysis and Objective**

This paper developed a solution for an AI-driven image enhancement web application. It follows the AI-as-a-Service model with a pay-as-you-go billing method. The initial margin between server cost and the overall subscription fee of the application is convenient. However, eventually, with the popularity, the margin narrowed down. As a result, the application's revenue flow started impacting the business model's overall sustainability. Figure 1. The reason behind this fall in revenue is the increase in the number of active hosts. Further investigation shows the computing resources in the active hosts are not fully utilized. Although there are scopes for handling more API calls with the existing hosts, the system activates new hosts. As a result, the operational cost increases. The challenge is identifying the under-utilized host servers, allocating the new requests to these hosts, and reducing the number of active hosts. In this way, the operational cost can be minimized, increasing the profit margin at a sustainable level. The proposed methodology has been developed to achieve this objective.

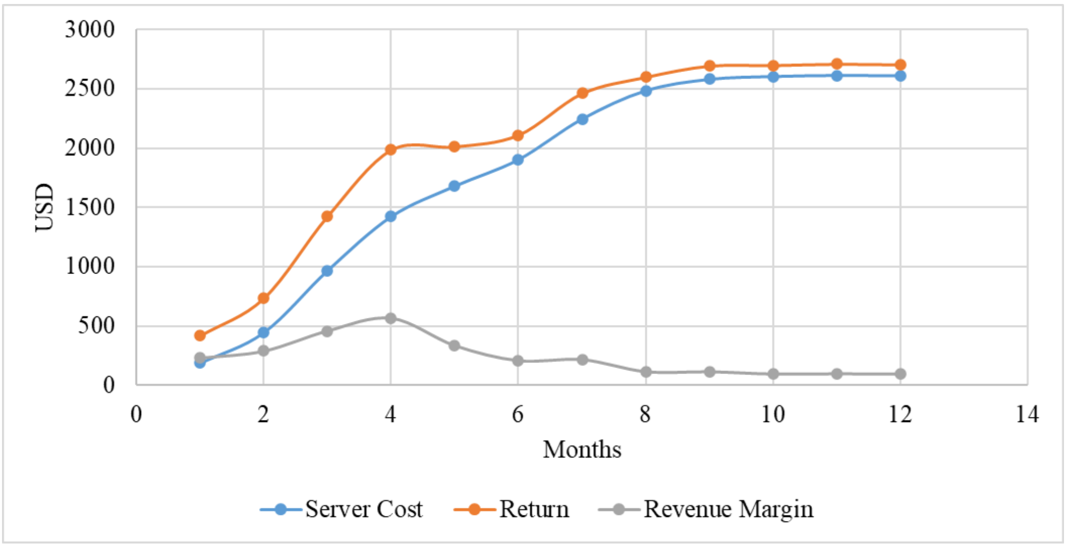

**Figure 1.** The relation among server cost, return of the investment, and revenue margin.

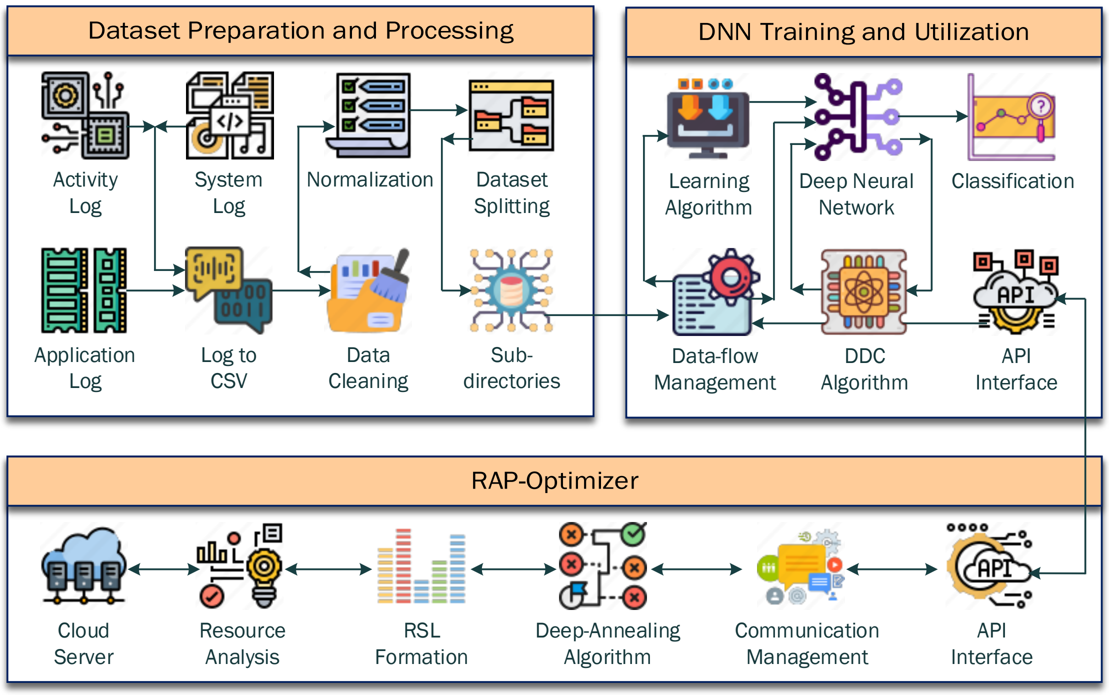

**Figure 2.** The methodological overview of the proposed RAP-Optimizer.

## 4. Methodology

The proposed methodology starts with dataset processing, constructed from the experiment's AIaaS application log files. Later, a well-optimized DNN architecture is developed to predict the appropriate configuration for a service request. After that, an innovative approach of reducing the overfitting effect and shifting to balance learning is incorporated into the network during the training process. The trained DNN has been used to develop the RAP-Optimizer, which consists of a resource analysis module, resource space landscape, and a novel Deep-Annealing algorithm.

## *4.1. Dataset Preparation*

This study uses a unique dataset prepared from the AIaaS application log and a sample of the dataset is presented in Table 1. The application maintains three types of log files, which are the Activity Log ($Log_{ac}$), System Log ($Log_{st}$), and Application Log ($Log_{ap}$). The relationships among these logs are illustrated in Figure 3. The $Log_{ac}$ keeps track of the user interaction, $Log_{st}$ is responsible for keeping track of computational resource usage, and $Log_{ap}$ is dedicated to application-related data. All data from these logs have been converted into Comma Separated Value (CSV) format [31]. After that, the instances have been categorized into five classes as presented in Table 1. The Pearson Correlation Coefficient (PCC) score, calculated using Equation 1, has been used to identify the relevant features that have a strong correlation with the five classes [32].

**Table 1.** A modified sample of the dataset with all target variables.

| Peak Frequency | Active Time (hours) | API Initiation Count | Service Requests | vCPU | vRAM (GB) | vDisk (GB) | Energy Usage (Wh) | Cloud Configuration |
|---|---|---|---|---|---|---|---|---|
| 2 | 0.25 | 12 | 1500 | 2 | 1.5 | 0.6 | 10 | Basic |
| 4 | 0.45 | 30 | 1800 | 4 | 2.5 | 1 | 40 | Standard |
| 7 | 1.5 | 220 | 6200 | 5 | 4 | 2.5 | 50 | Intermediate |
| 8 | 3.2 | 340 | 8300 | 7 | 6 | 3.5 | 75 | Advanced |
| 11 | 5.8 | 550 | 11200 | 9 | 8 | 5 | 95 | Premium |

$$r = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2} \quad (1)$$

In Equation 1, $X_i$ represents the individual feature values, $Y_i$ stands for the individual target variable, $\overline{X}$ is the mean of the features, $\overline{Y}$ is the mean of the target variable, and $n$ is the total data points. It generates the linear correlation score ($r$) between $X$ and $Y$ with a range from $-1$ to $+1$ where the former represents a perfect negative correlation, and the latter means a perfect positive correlation.

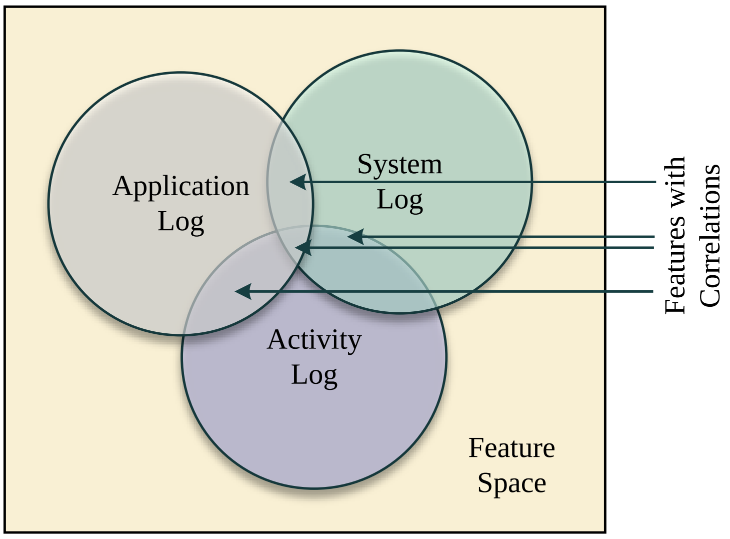

**Figure 3.** The overlapping features from three different log files.

### 4.1.1. Dataset Description

The dataset contains records of the experimental application for 365 days. After cleaning, there are 1,42,784 instances in the dataset. The feature variables are Peak Frequency, Active Time (hours), API Initiation Count, Service Requests, Virtual CPU, Virtual RAM (GB), Virtual Disk (GB), Energy

Usage (Wh), and Cloud Configuration. Except for the target variable, all other features are numerical. The target variable is categorical, which has five categories.

### 4.1.2. Dataset Cleaning

The CSV file constructed from $Log_a c$, $Log_s t$, and $Log_a p$ contains numerous incomplete rows. These rows were created for multiple reasons, including incomplete service requests, API initialization failure, and network issues. In addition, there are multiple duplicate rows. The uncleaned dataset has 1,63,150 instances. This dataset has been cleaned by following the mathematical principle defined in Equation 2.

$$C^* = \{y \in C \mid \forall z_j \in Z, y_j \neq \text{null}\} \tag{2}$$

In Equation 2, $C$ represents the uncleaned dataset. It contains $n$ records, where $C = y_1, y_2, ..., y_n$ and each $y_i$ corresponds to the $i^{th}$ observation vector across all variables $Z = z_1, z_2, ..., z_p$. The cleaned dataset $C$ includes only those observations from $C$ where no variable in the observation vector is missing, denoted by "null." Additionally, outliers from $C$, determined using the mean $\mu_c$ and standard deviation $\sigma_c$, have been eliminated based on the rule in Equation 3 [33]. A data point $y \in C$ is classified as an outlier if it satisfies $y < \mu_c - b \cdot \sigma_c \quad \text{or} \quad y > \mu_c + b \cdot \sigma_c$, where $b$ is a constant set to 3 according to the empirical 68-95-99.7 rule. The result is a dataset $C$ with no outliers [34].

$$C^* = \{y \in C \mid \mu_c - b \cdot \sigma_c \leq y \leq \mu_c + b \cdot \sigma_c\} \tag{3}$$

The cleaned dataset $C^*$ has no missing values or outliers. Although this process reduces the number of entries, the final count is 1,42,784 after cleaning, which is sufficient to train a deep neural network (DNN) to classify the target variables using the input features.

### 4.1.3. Feature Normalization

The feature variables exhibit a wide range of variations in the dataset, so feature normalization is essential [35]. Z-Score normalization has been chosen to normalize the features because it effectively handles data with varying scales and distributions. Figure 4 illustrates the difference in data range before and after performing the normalization. The Z-Score normalization process for a feature $x_i$ is defined in Equation 4 [36].

$$x_i' = \frac{x_i - \mu_x}{\sigma_x} \tag{4}$$

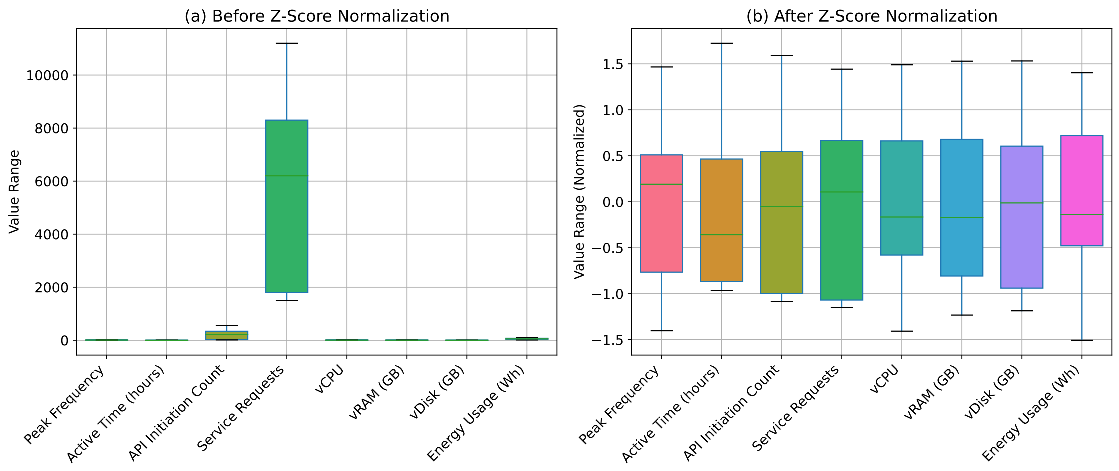

**Figure 4.** The feature variable ranges before and after performing the Z-score normalization.

In Equation 4, $x_i$ is the original feature value, $\mu_x$ is the mean of the feature $x$, $\sigma_x$ is the standard deviation of the feature $x$, and $x_i^{'}$ is the normalized feature value. After applying Z-Score normalization, the feature values are transformed such that the dataset has a mean of zero and a standard deviation of one, ensuring all features contribute equally during model training [37].

#### 4.1.4. Dataset Splitting

Ullah et al. [38] conducted a systematic review on Machine Learning (ML) applications, and it shows that most of the state-of-the-art approaches use 70:15:15 dataset splitting ratio of training, testing, and validation dataset respectively. The same ratio was used in this study. There are a total of 1,42,784 instances in the cleaned dataset. At the 70:15:15 ratio, there are 99,948 instances for training, 21,417 instances for training, and the same number for validation.

### *4.2. Network Architecture*

The proposed RAP-Optimizer uses a six-layer six-layer, fully connected deep neural network, as shown in Figure 5. The network architecture has been carefully designed to predict the cloud configuration requirements when AIaaS applications are initiated. An innovative Dynamic Dropout Control (DDC) algorithm has been integrated with the network to adjust the dropout rate of each layer dynamically for optimal unbiased performance. The network has four hidden layers, with 32 hidden neurons in each layer. The input layer is designed to accept an input vector $\boldsymbol{\xi} \in \mathbb{R}^8$, representing eight different features denoted by $\eta$, where $\eta = 8$. The working principle of the input layer is expressed by Equation 5, which transposes the input vectors [39].

$$\boldsymbol{\xi} = [\xi_1, \xi_2, \ldots, \xi_\eta]^T \tag{5}$$

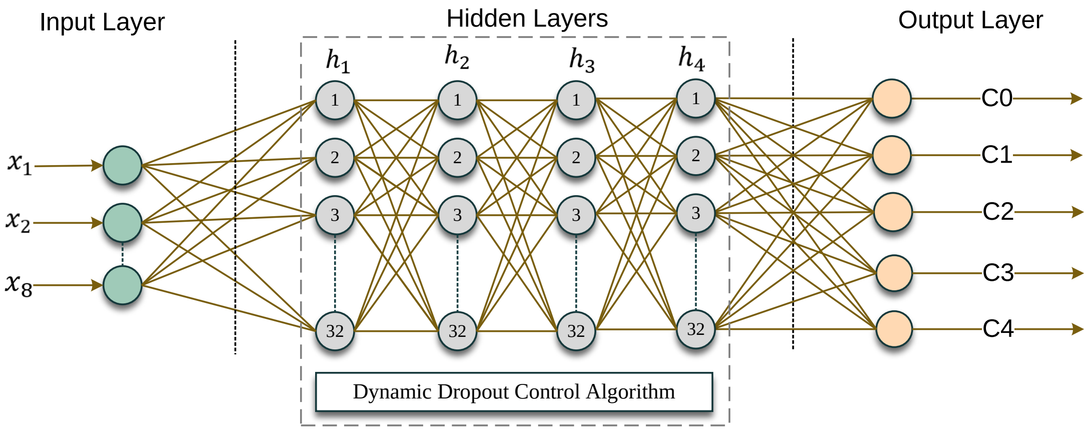

**Figure 5.** The network architecture of the 6-layer deep fully connected Neural Network.

In the proposed network, each of the four hidden layers processes its input $\chi^{[l-1]}$ using a weight matrix $\boldsymbol{\Omega}^{[l]}$, a bias vector $\boldsymbol{\zeta}^{[l]}$, and applies the Rectified Linear Unit (ReLU) activation function, denoted as $\varphi(\cdot)$, described in Equation 7. For layer $l$, the transformation is formulated in Equation 6, where $l = 1, 2, \ldots, 5$, and the dimensions of the weight matrices and bias vectors are $\boldsymbol{\Omega}^{[l]} \in \mathbb{R}^{32\times 8}$ and $\boldsymbol{\zeta}^{[l]} \in \mathbb{R}^{32}$, respectively [40].

$$\chi^{[l]} = \varphi(\boldsymbol{\Omega}^{[l]}\chi^{[l-1]} + \boldsymbol{\zeta}^{[l]}) \tag{6}$$

$$\varphi(\xi) = \begin{cases} \xi & \text{if } \xi > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

The output layer of this neural network contains five nodes corresponding to the five output classes. The activation function used in this layer is the Softmax function, described in Equation 8, which converts the raw output values into probabilities. Given an input $\boldsymbol{\xi}$, the logits $\varsigma_i$ are obtained through a linear transformation, followed by the Softmax function to compute the probabilities $v_i$ for each class $i$, where $i = 1, \ldots, 5$ [41].

$$v_i = \frac{\exp(\varsigma_i)}{\sum_{j=1}^{5} \exp(\varsigma_j)} \tag{8}$$

## 4.3. *Training the Network*

The network is trained using the back-propagation algorithm [42]. During the forward pass, input data $\boldsymbol{\xi}$ propagates through the network, and each layer $l$ calculates its output $\boldsymbol{\chi}^{[l]}$ as a function of the input from the previous layer $\boldsymbol{\chi}^{[l-1]}$, employing its weight matrix $\boldsymbol{\Omega}^{[l]}$, bias vector $\boldsymbol{\zeta}^{[l]}$, and activation function $\sigma^{[l]}$. This process continues through all layers until the output prediction $\hat{\boldsymbol{v}}$ is made. The prediction is then compared to the true labels $\boldsymbol{v}$ using a loss function, defined in Equation 9, which measures the difference between predicted and actual values [43]. In this equation, $\Gamma$ represents the number of output classes, $v_i$ is the true label, and $\hat{v}_i$ is the predicted probability for class $i$ [44].

$$\mathcal{L}(\boldsymbol{v}, \hat{\boldsymbol{v}}) = -\sum_{i=1}^{\Gamma} v_i \log(\hat{v}_i) \tag{9}$$

### 4.3.1. Learning Algorithm

The Adaptive Moment Estimation (ADAM) optimizer is used to update the weights of the network's hidden nodes. Initially, the vectors $\mu_0$ and $\nu_0$ are set to zero, and the time step $\tau$ is initialized to zero. The learning rate is represented by $\alpha$, and $\beta_1$ and $\beta_2$ are the decay rates for moment estimates, initialized to 0.90. At each time step $\tau$, the gradients $\nabla_\theta \mathcal{J}(\theta)$ with respect to the parameters $\theta$ are calculated. The updates for $\mu_\tau$ and $\nu_\tau$ are defined by Equations 10 and 11 [45].

$$\mu_\tau = \beta_1 \mu_{\tau-1} + (1 - \beta_1) \nabla_\theta \mathcal{J}(\theta) \tag{10}$$

$$\nu_\tau = \beta_2 \nu_{\tau-1} + (1 - \beta_2) (\nabla_\theta \mathcal{J}(\theta))^2 \tag{11}$$

The bias-corrected estimates for $\mu_\tau$ and $\nu_\tau$ are given by Equations 12 and 13.

$$\hat{\mu}_\tau = \frac{\mu_\tau}{1 - \beta_1^\tau} \tag{12}$$

$$\hat{\nu}_\tau = \frac{\nu_\tau}{1 - \beta_2^\tau} \tag{13}$$

Finally, the weights $\omega$ are updated using Equation 14, where $\epsilon$ is a small constant for numerical stability.

$$\omega = \omega - \alpha \frac{\hat{\mu}_\tau}{\sqrt{\hat{\nu}_\tau} + \epsilon} \tag{14}$$

At the 70:15:15 ratio, there are 99,948 instances for training, 21,417 instances for training, and the same number for validation.

### 4.3.2. Learning Curve

The training dataset contains 99,948 instances. To enhance training efficiency, mini-batches of size 64 are utilized. The learning curve, depicting the progress of both accuracy and loss for training and validation, is presented in Figure 6. It showcases the network's performance over 50 epochs with

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 October 2024 doi:10.20944/preprints202410.0504.v1

1,11,550 iterations, taking approximately 8 hours and 27 minutes to complete. The validation accuracy achieved is 97.48%, with a validation loss of 2.82%, while the training accuracy and loss are 95.15% and 4.85%, respectively.

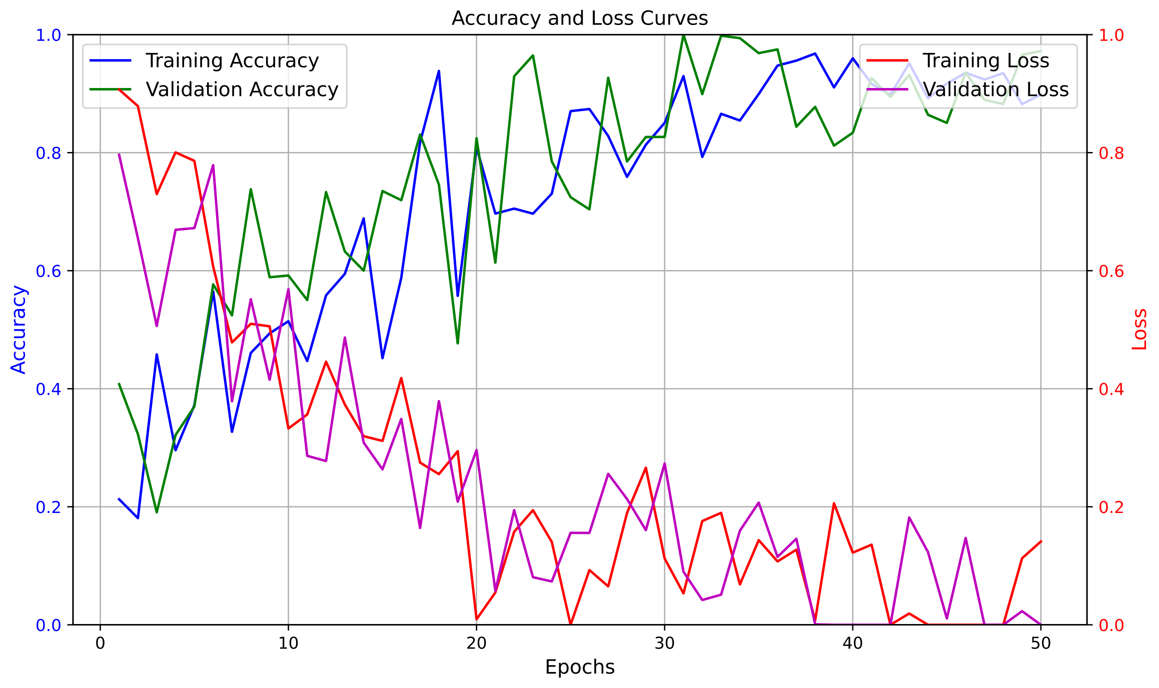

**Figure 6.** The learning progress in terms of training accuracy, validation accuracy, training loss, and validation loss.

### 4.3.3. DDC Algorithm

During the development phase of the RAP-Optimizer, it was observed that the network overfits after training. The number of hidden layers and nodes of the existing layers was reduced to minimize overfitting. After the modification, the network exhibits underfitting characteristics. Table 2 shows different configurations explored during the development and the characteristics of the network along with other parameters. To ensure balanced fitting, the innovative DDC algorithm, presented as Algorithm 1, was developed and integrated into the network. It gradually discovers the optimal number dropout rate for different hidden layers and balances an overfitting network.

**Table 2.** Impact of Different Network Configurations on Overfitting and Underfitting Characteristics in the RAP-Optimizer.

| Network Configuration | Number of Hidden Layers | Neurons per Layer | Characteristics | Observed Behavior |
|---|---|---|---|---|
| Initial Configuration | 6 | 32 | Overfitting | High training accuracy, low validation accuracy. |
| Modified Configuration 1 | 5 | 32 | Overfitting | Overfitting persists, validation accuracy slightly improves but still significantly lower than training. |
| Modified Configuration 2 | 4 | 32 | Overfitting | Moderate overfitting; slight improvement in validation performance, but gap remains. |
| Modified Configuration 3 | 3 | 16 | Overfitting | Reduced overfitting but validation accuracy still does not match training accuracy. |
| Modified Configuration 4 | 4 | 8 | Underfitting | Model starts underfitting; both training and validation accuracy are low. |
| Modified Configuration 5 | 4 | 4 | Underfitting | Significant underfitting; both accuracies remain low, model complexity too reduced. |
| Modified Configuration 6 | 2 | 16 | Underfitting | Underfitting persists, accuracy too low for both training and validation. |
| Modified Configuration 7 | 1 | 32 | Underfitting | Severe underfitting; network too shallow to capture complex patterns. |

**Algorithm 1** Dynamic Dropout Control (DDC) Algorithm

```
 1: Input: Trained network $\mathcal{N}$, Training set $X_{train}$, Validation set $X_{val}$, Initial dropout rate $p_0$, Number
    of layers $L$, Dropout increment $\Delta p$, Threshold $\delta$
 2: Output: Modified network with minimized overfitting
 3: Initialize $p \leftarrow p_0$
 4: Compute initial accuracies $\text{Acc}_{train}, \text{Acc}_{val}$
 5: $\Delta \leftarrow \text{Acc}_{train} - \text{Acc}_{val}$
 6: while $\Delta > \delta$ do
 7:     for $l = 1$ to $L$ do
 8:         if first iteration then
 9:             $\mathbf{h}^{[l]} \leftarrow \text{Dropout}(\mathbf{h}^{[l]}, p)$
10:         else
11:             Increase dropout: $p \leftarrow p + \Delta p$
12:             $\mathbf{h}^{[l]} \leftarrow \text{Dropout}(\mathbf{h}^{[l]}, p)$
13:         end if
14:     end for
15:     Recompute $\text{Acc}_{train}, \text{Acc}_{val}$
16:     Update $\Delta \leftarrow \text{Acc}_{train} - \text{Acc}_{val}$
17: end while
18: Return $\mathcal{N}$ with reduced overfitting
```

### 4.4. *RAP-Optimizer*

The proposed RAP-Optimizer is an innovative approach that starts with resource analysis. After that, the resource space landscape is generated to evaluate the resource distribution. Finally, it combines DNN with the Simulated Annealing optimization to optimize the cloud resource optimization without compromising the service quality.

#### 4.4.1. Resource Analysis

The RAP-Optimizer starts with resource analysis, performed using the Resource Analyzer (RAN) algorithm, presented as Algorithm 2, developed in this study. The RAN algorithm requires access and scanning permission for both Virtual Machines (VMs) and their physical hosts. Initially, it scans the entire system, identifies the active VMs through their unique ID (VID), retrieves their current configurations, and fetches the source consumption history. After that, it explores the physical hosts supporting the VMs. It performs similar operations on physical hosts. Finally, it scans the idle physical hosts. The RAN algorithm maintains a Resource Status Table (RT) which is frequently updated. Based on the resource consumption, it generates a Resource Space (RS) represented by bar graphs.

**Algorithm 2** Resource Analyzer (RAN) Algorithm

```
 1: procedure RAN_OPTIMIZER
 2:     Input: Access permissions for VMs and physical hosts
 3:     Output: $RS$, Updated $RT$
 4:     Initialize: $RT \leftarrow Init(RoutingTable)$
 5:     Scan system and fetch VMs and Hosts
 6:     for $vm \in VMs$ do                                   ▷ For each VM
 7:         Identify active VMs: $vm \leftarrow ID(vm)$
 8:         Get VM configurations: $cfg(vm)$
 9:         Fetch resource history: $res\_hist(vm)$
10:     end for
11:     for $host \in Hosts$ do                               ▷ For each host
12:         Get host configurations: $cfg(host)$
13:         Fetch resource history: $res\_hist(host)$
14:     end for
15:     Scan idle hosts: $idle\_hosts \leftarrow Scan(Hosts)$
16:     Update: $RT \leftarrow Update(cfg, res\_hist)$
17:     Generate: $RS \leftarrow (VID, Resource)$
18:     return $RS, RT$               ▷ Return both Resource Space and Routing Table
19: end procedure
```

#### 4.4.2. Resource Space Landscape (RSL)

The RAN algorithm returns the RS and RT, where the VM configurations, current resource consumption rate, IDs of the VMs ($VID$), corresponding physical hosts, and other relevant data are available. These data are used to prepare the RSL, showing active VMs in physical hosts. The RSL of a random instance is illustrated in Figure 7. The data center offering the AIaaS consists of 16 physical hosts. According to the organization's policy, a maximum of 10 VMs are allowed concurrently in a single host to maintain the Terms of Services (TOS). Each physical host is powered by a 10-core CPU with 128GB primary memory.

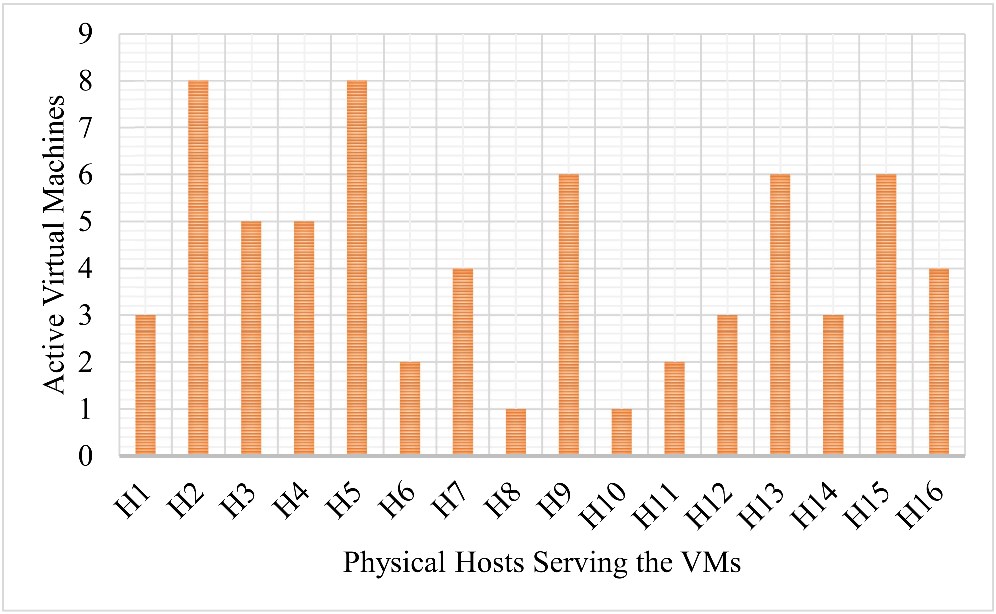

**Figure 7.** The State Space Landscape shows the number of active VMs running on hosts.

#### 4.4.3. Deep-Annealing Algorithm

The RAP-Optimizer combines the DNN presented in Section 4.2 and the Simulated Annealing algorithm, which has been named as Deep-Annealing and presented in Algorithm 3. The process starts by connecting to the AIaaS Data Center (AIDC) with permission to access the status of all physical and virtual devices. It applies the RAN Analyzer algorithm to retrieve the Resource Space (RS) and Resource Table (RT), which include the VM configurations, current resource usage, and corresponding physical hosts. These outputs provide a comprehensive view of the system's Resource Space ($RSS$). After that, the deep-annealing algorithm utilizes the DNN to predict the VM configuration and respond to the request for subsequent AI services. The Deep-Annealing algorithm identify hosts for VM deployment. It explores potential hosts iteratively, evaluating the cost ($cost$) and objective ($obj$) functions to select a suitable host. The process begins at a high-temperature state, allowing probabilistic acceptance of less optimal solutions to escape local minima. The $HasCapacity()$ function is used to check whether the current host has sufficient resources for the new VM whose configuration was predicted by the DNN. If there are sufficient resources, the VM is deployed using $CreateInstance()$. If the current host lacks the required resources, the algorithm utilizes $NextHost()$, which selects another host based on a probability distribution influenced by the current temperature, allowing for a broader exploration of the available configurations.

**Algorithm 3** Deep-Annealing Algorithm

```
1:  $AIDC \leftarrow \text{Connect}(credentials)$
2:  $(RS, RT) \leftarrow \text{RAN Analyzer}(AIDC)$
3:  $\hat{C}_{VM} \leftarrow \text{DNN}(RS, RT)$
4:  $RSS \leftarrow \text{RetrieveResourceSpace}(RS, RT)$
5:  Initialize temperature $T$
6:  $(H_{start}) \leftarrow \text{SimulatedAnnealing}(RSS, obj, cost, T)$
7:  while $\neg$InstanceCreated and $T > T_{\text{min}}$ do
8:      if HasCapacity$(H_{start}, \hat{C}_{VM})$ then
9:          CreateInstance$(H_{start}, VM)$
10:         $InstanceCreated \leftarrow$ True
11:     else
12:         $H_{start} \leftarrow \text{NextHost}(H_{start}, RSS, T)$
13:         Decrease temperature $T \leftarrow \alpha T$
14:     end if
15: end while
16: for $vm \in AIDC$ do
17:     if CanMigrate$(vm, H_{start})$ then
18:         Migrate$(vm, H_{start})$
19:     end if
20: end for
```

If the initially selected host lacks sufficient resources for the VM deployment, $NextHost()$ identifies alternative hosts based on a probability distribution influenced by the current temperature $T$. This ensures the search process avoids local optima and explores a wider range of configurations. As the temperature gradually decreases with each iteration by a factor $\alpha$, the exploration process narrows down to focus on more optimal hosts for VM deployment. Once a suitable host is found and the VM instance is deployed, the algorithm evaluates potential migrations for existing VMs to optimize overall resource usage. The function $CanMigrate()$ checks whether a VM can be moved to a host that offers better resource efficiency, and if migration is feasible, the $Migrate()$ function performs the transfer. This process ultimately aims to consolidate workloads, improve resource utilization, and reduce the operational costs associated with idle hosts.

## 5. Experimental Result and Evaluation

### 5.1. Evaluation Metrics

The overall performance of the RAP-Optimizer depends on the prediction quality of the DNN developed in this study. The performance of this DNN has been evaluated using accuracy, precision, recall, and F1-score. These are the most frequently used evaluation metrics for Machine Learning approaches [46]. These metrics are defined in Equations 15, 16, 17, and 18, respectively, which are dependent on True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). This study retrieves these values from the confusion matrix illustrated in Figure 8.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{15}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{16}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{17}$$

$$\text{F1 Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}} \tag{18}$$

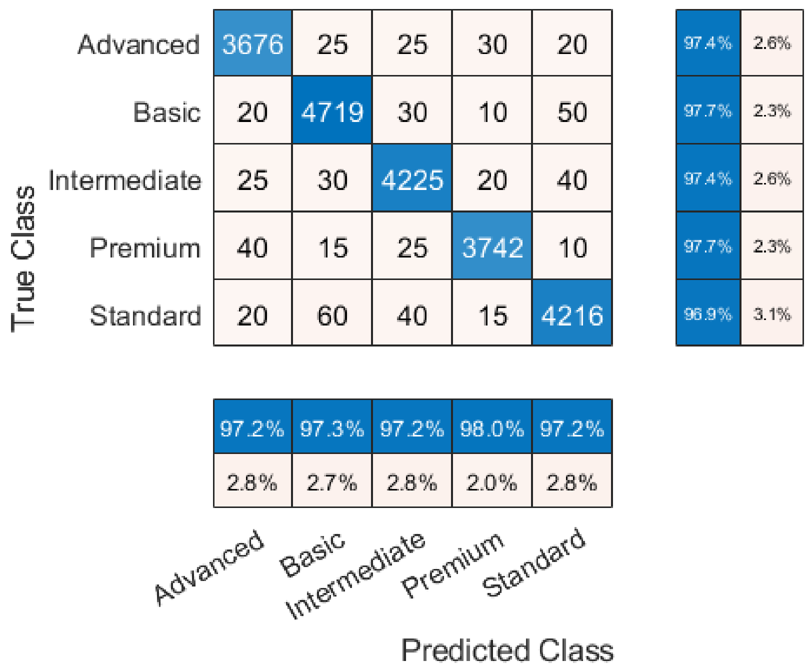

**Figure 8.** The confusion matrix obtained from the test dataset with 21,417 instances.

Apart from ML performance evaluation metrics, the RAP-Optimizer performance has been evaluated based on the capability of reducing the number of active hosts, improving API request management, and resource optimization [11,47]. Besides, the overall objective achievement has been used as an evaluation metrics as well.

## *5.2. Confusion Matrix Analysis*

The confusion matrix analysis reveals the performance of the classification model across five target classes: Basic, Standard, Intermediate, Advanced, and Premium, with an overall accuracy of 96.1%. The precision values vary slightly among the classes, with Basic achieving the highest precision at 97.7%, followed by Premium at 97.7%, Intermediate at 97.4%, Advanced at 97.3%, and Standard at 96.9%. Recall values indicate how well the model identifies each class, with Premium leading at 98.0%, followed by Basic at 97.3%, Standard at 97.2%, Intermediate at 97.4%, and Advanced at 97.2%. The F1-scores, which balance precision and recall, are consistent across the classes, with Basic at 97.5%, Standard at 97.0%, Intermediate at 97.4%, Advanced at 97.3%, and Premium at 97.8%. These metrics suggest that the model performs well overall, but slight variations exist between classes in terms of how accurately they are classified and identified.

## *5.3. K-Fold Cross Validation*

K-Fold Cross-Validation is a robust method used to evaluate the performance of machine learning models by splitting the dataset into multiple folds. In this analysis, six folds were used, as shown in Table 3, with slight variations observed across each fold for metrics such as accuracy, precision, recall, and F1-score. The average values were calculated across all folds to provide a reliable overall assessment. Accuracy remained consistent, ranging from 96.0% to 96.4%, while precision ranged between 97.3% and 97.7%. Recall and F1-score also showed slight variations, with recall values between 96.8% and 97.6%, and F1-scores between 97.0% and 97.3%. The spider chart in Figure 9 visually represents the performance across the metrics, emphasizing the stability and effectiveness of the model across the different folds, providing confidence in its generalizability.

**Table 3.** Performance Evaluation using K-Fold Cross-Validation.

| Metrics | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 | Fold 6 | Average |
|---|---|---|---|---|---|---|---|
| Accuracy | 0.962 | 0.961 | 0.964 | 0.96 | 0.963 | 0.961 | 0.9618 |
| Precision | 0.975 | 0.976 | 0.973 | 0.977 | 0.974 | 0.976 | 0.9751 |
| Recall | 0.97 | 0.969 | 0.971 | 0.968 | 0.97 | 0.969 | 0.9695 |
| F1-Score | 0.972 | 0.971 | 0.973 | 0.97 | 0.972 | 0.971 | 0.9715 |

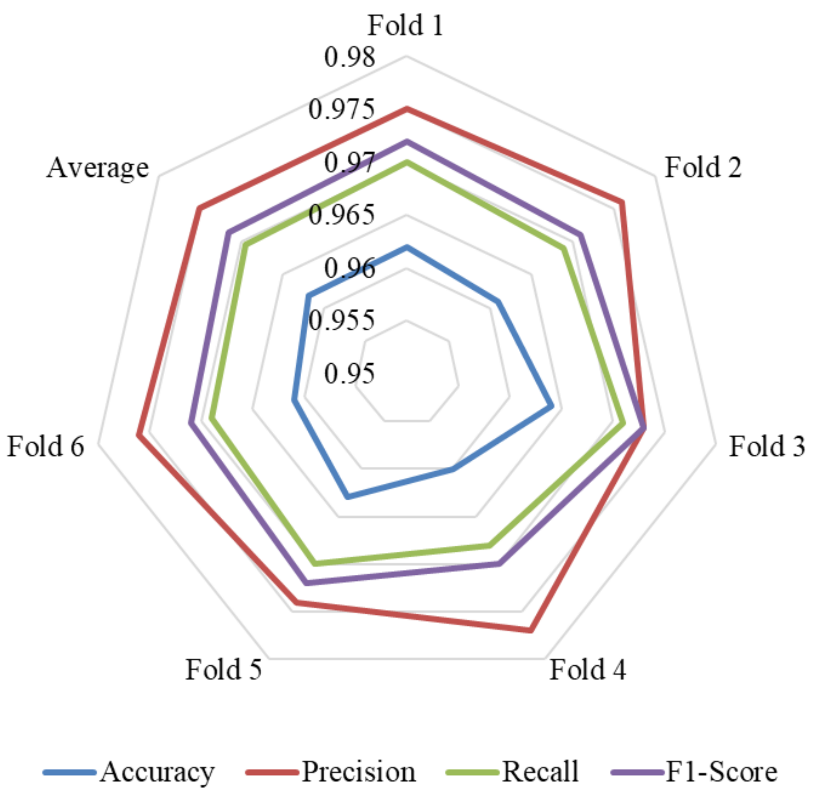

**Figure 9.** The resource configuration prediction performance analysis using k-fold cross validation.

### *5.4. Active Physical Host*

Table 4 presents a comparative analysis of the number of active hosts per 24 hours before and after the implementation of the Deep-Annealing algorithm over a twelve-week period. Before the introduction of the algorithm, the average number of active hosts per day was recorded at 33. With the deployment of the Deep-Annealing algorithm, this average was reduced to 28 active hosts per day, signifying an average reduction of 5 hosts. This demonstrates the efficacy of the Deep-Annealing algorithm in optimizing resource utilization within the data center environment.

**Table 4.** The number of active hosts per 24 hours before and after using the Deep-Annealing algorithm.

| **Weeks** | **Number of Active Host Per 24 hours** | | **Reduction** |
|---|---|---|---|
| | **Without Deep-Annealing** | **With Deep-Annealing** | |
| 1 | 36 | 30 | 6 |
| 2 | 29 | 22 | 7 |
| 3 | 37 | 33 | 4 |
| 4 | 31 | 27 | 4 |
| 5 | 35 | 29 | 6 |
| 6 | 33 | 27 | 6 |
| 7 | 39 | 34 | 5 |
| 8 | 30 | 25 | 5 |
| 9 | 38 | 32 | 6 |
| 10 | 32 | 27 | 5 |
| 11 | 34 | 28 | 6 |
| 12 | 29 | 24 | 5 |
| Average | 33 | 28 | 5 |

A closer examination, illustrated in Figure 10, reveals a fluctuating yet consistently positive impact of the Deep-Annealing algorithm across the weeks. The most notable improvement was observed in the second week, where the number of active hosts was reduced by 7, from 29 without the algorithm to 22 with it. The least improvement was seen in weeks 3 and 4, where a reduction of 4 hosts was achieved. Weeks 1, 5, 6, 9, and 11 saw a reduction of 6 hosts, while weeks 7, 8, 10, and 12 observed a decrease of 5 hosts. These results underscore the capacity of the Deep-Annealing algorithm to effectively manage and reduce the number of active hosts required to support the operational demands of a SaaS application data center.

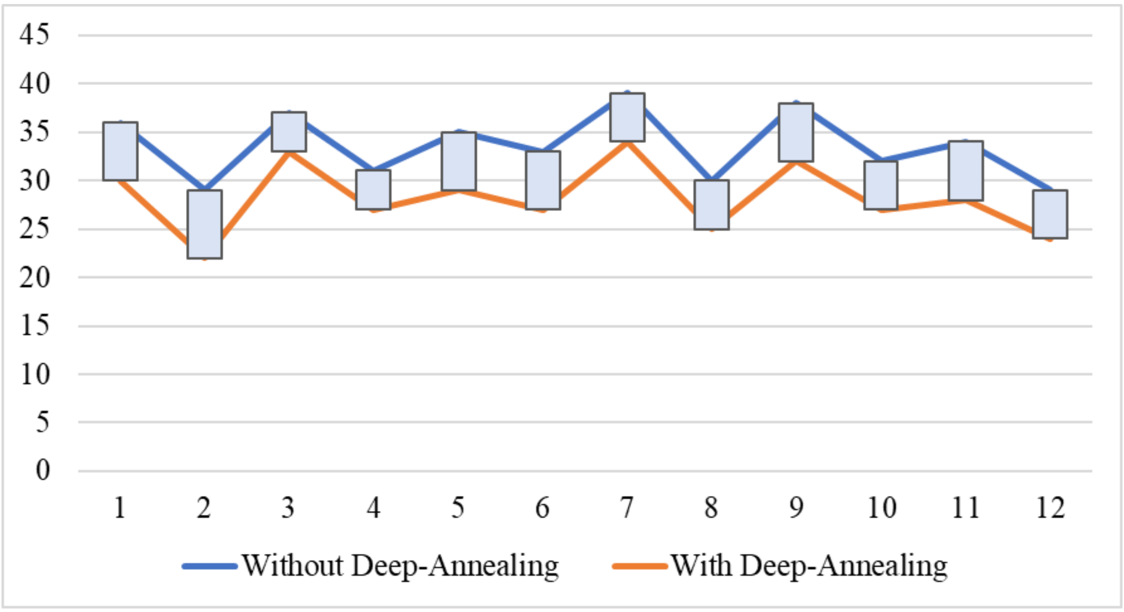

**Figure 10.** The comparison of the number of active hosts before and after using the Deep-Annealing algorithm.

## 5.5. Request Optimization

Table 5 presents the resource utilization and API request handling before and after implementing the proposed RAP-Optimizer. Initially, all physical hosts were active, handling a random number of API requests, with some hosts underutilized in terms of CPU and RAM. After applying the RAP-Optimizer, the system efficiently allocated API requests to fewer hosts, maximizing resource utilization before activating additional hosts. As a result, five physical hosts were placed into idle mode, conserving resources and reducing energy consumption. The remaining hosts handled the same workload but at higher efficiency, utilizing their CPU and RAM to near full capacity. Before the method was implemented, many hosts processed fewer API requests, with CPU cores and RAM underutilized. For example, hosts 4, 6, and 10 handled only a fraction of their capacity, running only 3 to 7 API requests, leaving a significant portion of resources idle. After applying the Deep-Annealing method, these API requests were consolidated onto hosts with higher resource availability, allowing some physical hosts to transition into idle mode. This approach optimized resource utilization across the active hosts while reducing operational overhead. The results, as shown in Table 5, demonstrate that the proposed system can handle the same number of API requests using fewer physical hosts, which leads to significant resource savings and improved data center efficiency.

**Table 5.** API request handling and resource optimization before and after using the Deep-Annealing algorithm.

| Host | Resource Capacity | | API Requests Before RAP-Optimizer | | API Requests After RAP-Optimizer | |
|---|---|---|---|---|---|---|
| | CPU (Cores) | RAM (GB) | Processed | CPU Cores | Processed | CPU Cores |
| 1 | 10 | 128 | 12 | 8 | 18 | 10 |
| 2 | 10 | 128 | 15 | 9 | 20 | 9 |
| 3 | 10 | 128 | 16 | 10 | 22 | 10 |
| 4 | 10 | 128 | 7 | 5 | Handled by Host 1-3 | Idle Mode |
| 5 | 10 | 128 | 14 | 9 | 19 | 10 |
| 6 | 10 | 128 | 6 | 4 | Handled by Host 1-3 | Idle Mode |
| 7 | 10 | 128 | 9 | 7 | 18 | 9 |
| 8 | 10 | 128 | 13 | 8 | 20 | 9 |
| 9 | 10 | 128 | 10 | 6 | 16 | 9 |
| 10 | 10 | 128 | 5 | 3 | Handled by Host 5-9 | Idle Mode |
| 11 | 10 | 128 | 8 | 6 | Handled by Host 5-9 | Idle Mode |
| 12 | 10 | 128 | 4 | 3 | Handled by Host 5-9 | Idle Mode |

### *5.6. Resource Optimization*

The RAP-Optimizer demonstrates significant resource optimization potential. Figure 11 shows the before and after effect comparison. Initially, before implementing the RAP-Optimizer, the server costs steadily increased from \$200 to \$2,600 over the 12 months, as the system inefficiently activated additional hosts to handle the growing number of API requests. This led to underutilized resources and inflated operational costs. After applying the proposed RAP-Optimizer, the server cost was reduced each month, stabilizing at around \$1,250 by the 12th month. This reduction is attributed to the system's ability to consolidate API requests to fully utilize existing active hosts before activating new ones. As a result, unnecessary activation of hosts was minimized, leading to significant cost savings.

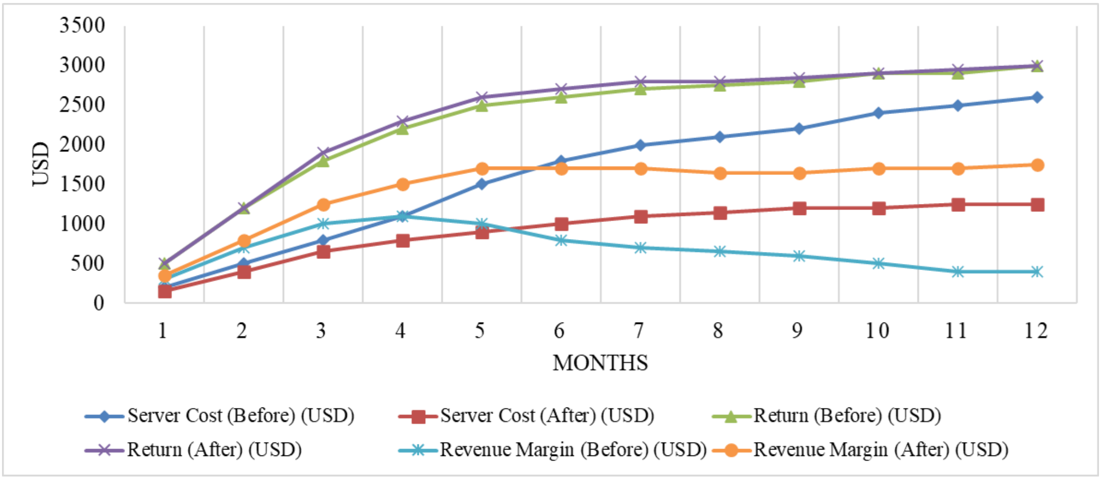

**Figure 11.** The cost optimization before and after using the proposed method.

Despite the optimization, the application's return remained consistent, showing that user demand and subscription revenues were unaffected by the changes. The table highlights a clear increase in the revenue margin after optimization—from an initial margin of \$300 in the first month, the margin grew to \$1,750 by the 12th month. This improvement demonstrates the effectiveness of the proposed system in reducing operational costs while maintaining consistent service delivery, thus ensuring the profitability and sustainability of the AI-driven image enhancement web application over time. By efficiently managing resources and reducing the number of active hosts, the system successfully increases profit margins, even as server usage grows.

### *5.7. Objective Achievement*

The primary objective of this study is to increase the profit margin. After implementing the proposed RAP-Optimizer, server cost, return, and revenue margin have been observed for 12 months. During this observation period, the existing approach and the system with RAP-Optimizer were active simultaneously. The number of users allowed to use the optimized system was kept similar to the existing system for fair comparison. The data observed over the 12-month period are listed in Table

6. After using the RAP-Optimizer, on average, the server cost was reduced by approximately 45%, dropping from an initial $2,600 to a stable $1,250 by the 12th month. This resulted in an average monthly cost reduction of $1,150. In parallel, the profit margin increased from an average of $600 per month before optimization to $1,675 after optimization, reflecting a 179% increase in profit over the 12-month period. Additionally, the reduction in the number of active hosts, as shown in Table 5, led to more efficient resource utilization, ensuring that the same number of API requests were handled with fewer servers, further driving operational cost savings and profit maximization.

**Table 6.** Numerical Analysis of the Objective Achievement.

| Months | Server Cost in USD | | Return in USD | | Revenue Margin in USD | |
|---|---|---|---|---|---|---|
| | **Before** | **After** | **Before** | **After** | **Before** | **After** |
| 1 | 200 | 150 | 500 | 500 | 300 | 350 |
| 2 | 500 | 400 | 1200 | 1200 | 700 | 800 |
| 3 | 800 | 650 | 1800 | 1900 | 1000 | 1250 |
| 4 | 1100 | 800 | 2200 | 2300 | 1100 | 1500 |
| 5 | 1500 | 900 | 2500 | 2600 | 1000 | 1700 |
| 6 | 1800 | 1000 | 2600 | 2700 | 800 | 1700 |
| 7 | 2000 | 1100 | 2700 | 2800 | 700 | 1700 |
| 8 | 2100 | 1150 | 2750 | 2800 | 650 | 1650 |
| 9 | 2200 | 1200 | 2800 | 2850 | 600 | 1650 |
| 10 | 2400 | 1200 | 2900 | 2900 | 500 | 1700 |
| 11 | 2500 | 1250 | 2900 | 2950 | 400 | 1700 |
| 12 | 2600 | 1250 | 3000 | 3000 | 400 | 1750 |

## 6. Limitations and Future Scope

While the proposed RAP-Optimizer has shown promising results in reducing operational costs and optimizing resource utilization, there are several areas where the approach can be further improved. This section outlines the key limitations of the current system and proposes future directions to enhance its functionality, scalability, and efficiency in handling cloud-based AI services.

### *6.1. Resource Utilization Focus*

One limitation of the proposed RAP-Optimizer is that it primarily focuses on optimizing CPU and memory utilization, while other critical factors, such as network bandwidth, disk I/O, and latency, are not fully considered. These elements play some role in the overall system performance. When the number of traffic increases, the impact of these elements becomes more vital. One of the approaches to overcome the limitations is to expand the dataset features and incorporate these elements as well. However, the scope of this extension will be explored in the subsequent version of this study.

### *6.2. Uniform API Request Handling*

The experimenting AIaaS application offers image processing services only. As a result, the cloud servers don't have any configuration variations. However, applications that offer multi-modal services, such as audio and video processing, will depend on multiple different hardware configurations at the back end. As a result, selecting the appropriate hardware to transfer the API call will require an additional logical layer to make appropriate decisions. The proposed RAP-Optimizer doesn't incorporate this complexity. This is another limitation of this approach. A potential solution to overcome this weakness is categorizing the API requests based on their nature, which will be explored in the future of this method.

The RAP-Optimizer considers that all API requests can be uniformly distributed across available hosts, without accounting for hardware differences or the geographical location of hosts, which may affect response times or performance. Addressing this limitation could involve enhancing the system to factor in host-specific attributes like hardware capabilities and geographic proximity, allowing for smarter resource allocation that reduces latency and improves service quality.

### *6.3. Predictive Optimization Approach*

The current version of the proposed RAP-Optimizer is a predictive approach. The overall effectiveness of the process depends on the accuracy of the prediction. One of the major weaknesses of this approach is if the configurations are predicted incorrectly, the RAP-Optimizer fails to handle it. One of the solutions to overcome this limitation is to incorporate the false positive rate into the decision-making process. The potential of this solution will be explored in the future scope of this study.

### *6.4. Single-Cloud Focus*

Multi-cloud or hybrid architectures offer more flexibility and cost-effective structure. The proposed AIaaS application runs on a single-cloud architecture, so the entire experiment is conducted on it. The findings are relevant, and the proposed approach is suitable for single-cloud architecture only. This is one of the weaknesses of the proposed system. However, hundreds of AIaaS runs on single-cloud architecture, which makes the proposed approach feasible and effective. The subsequent version of this paper will focus on multi-cloud or hybrid-cloud architecture in the future.

## 7. Conclusion

The RAP-Optimizer presented in this paper demonstrated an effective way to overcome the challenge of resource utilization and cost optimization in AIaaS applications. Although it focuses on an image enhancement application only, this study plays an instrumental role in creating the blueprint for optimization techniques for similar applications. This study also demonstrated an innovative application of the combination of DNN and Simulated Annealing algorithm in cloud resource optimization. The experimental results presented in this paper show significant reductions in the number of active physical hosts and server costs without compromising service quality. Over a twelve-month observation period, the RAP-Optimizer achieved an average reduction of 45% in server costs. It further shows a 179% increase in profit margins, clearly highlighting the practical benefits of the system. The DNN was designed and trained in this study to predict the resource configuration and maintain high classification accuracy across all five service classes. The confusion matrix shows that the DNN achieved 97.48% validation accuracy. K-fold cross-validation was performed to validate the performance. In the cross-validation, the performance of the DNN prediction remained consistent for all folds. That means the DNN is both accurate and well-trained and has good generalization capability. Besides, the introduction of the unique Dynamic Dropout Control (DDC) algorithm contributed to mitigating the overfitting effects. Combining the Simulated Annealing algorithm with the DNN unblocked a new dimension of AIaas cloud resource optimization, which dynamically manages the cloud resources. It effectively reduces the number of active hosts, which maximizes the existing resource utilization without activating new physical hosts, which saves a significant amount of resources and energy. This efficiency directly translates into improved sustainability and cost-effectiveness for AIaaS applications.

The innovative design, effective performance, and real-world positive impact on enlarging profit margin in AIaaS application business demonstrated the potential of the proposed RAP-Optimizer. Despite multiple advantages, the RAP-Optimizer suffers from certain limitations. Excluding other resources except for CPU and memory utilization, lack of heterogeneous API request management analysis, and experiment on the single-cloud environment are some of the limitations of the experiment presented in this paper. In conclusion, the RAP-Optimizer provides a scalable, efficient, and cost-effective solution for resource management in cloud-based AI services, offering substantial improvements in operational efficiency, energy savings, and profit margins. As AI-driven applications continue to grow in scale and complexity, solutions like the RAP-Optimizer will play a crucial role in ensuring that cloud resources are utilized to their fullest potential, paving the way for more sustainable and profitable cloud-based business models.

**Author Contributions:** Conceptualization, K.S. and S.A.; methodology, R.A. and A.V.; software, K.S. and A.V.; validation, K.S., R.A., and S.A.; formal analysis, K.S. and R.A.; investigation, R.A. and A.V.; resources, S.A.; data curation, R.A.; writing—original draft preparation, K.S. and R.A.; writing—review and editing, A.V. and S.A.; visualization, K.S. and S.A.; supervision, S.A.; project administration, K.S. and R.A.; funding acquisition, S.A. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

**Data Availability Statement:** The data utilized in this study were generated from an AIaaS application and are considered confidential. However, the data can be made available upon reasonable request. Requesters must agree to the terms and conditions, which stipulate that the data cannot be used for any commercial or research purposes without prior permission from the owner.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Deng, S.; Zhao, H.; Huang, B.; Zhang, C.; Chen, F.; Deng, Y.; Yin, J.; Dustdar, S.; Zomaya, A.Y. Cloud-native computing: A survey from the perspective of services. *Proceedings of the IEEE* **2024**.
2. Tuli, S.; Mirhakimi, F.; Pallewatta, S.; Zawad, S.; Casale, G.; Javadi, B.; Yan, F.; Buyya, R.; Jennings, N.R. AI augmented Edge and Fog computing: Trends and challenges. *Journal of Network and Computer Applications* **2023**, *216*, 103648.
3. Badshah, A.; Ghani, A.; Siddiqui, I.F.; Daud, A.; Zubair, M.; Mehmood, Z. Orchestrating model to improve utilization of IaaS environment for sustainable revenue. *Sustainable Energy Technologies and Assessments* **2023**, *57*, 103228.
4. Horchulhack, P.; Viegas, E.K.; Santin, A.O.; Ramos, F.V.; Tedeschi, P. Detection of quality of service degradation on multi-tenant containerized services. *Journal of Network and Computer Applications* **2024**, *224*, 103839.
5. Miikkulainen, R.; Liang, J.; Meyerson, E.; Rawal, A.; Fink, D.; Francon, O.; Raju, B.; Shahrzad, H.; Navruzyan, A.; Duffy, N.; others. Evolving deep neural networks. In *Artificial intelligence in the age of neural networks and brain computing*; Elsevier, 2024; pp. 269–287.
6. Pardalos, P.M.; Mavridou, T.D. Simulated annealing. In *Encyclopedia of Optimization*; Springer, 2024; pp. 1–3.
7. Zhou, G.; Tian, W.; Buyya, R.; Xue, R.; Song, L. Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions. *Artificial Intelligence Review* **2024**, *57*, 124.
8. Mohammadzadeh, A.; Chhabra, A.; Mirjalili, S.; Faraji, A. Use of whale optimization algorithm and its variants for cloud task scheduling: a review. *Handbook of Whale Optimization Algorithm* **2024**, pp. 47–68.
9. Musabimana, B.B.; Bucaioni, A. Integrating AIaaS into Existing Systems: The Gokind Experience. International Conference on Information Technology-New Generations. Springer, 2024, pp. 417–426.
10. Kurian, A.M.; Onuorah, M.J.; Ammari, H.M. Optimizing Coverage in Wireless Sensor Networks: A Binary Ant Colony Algorithm with Hill Climbing. *Applied Sciences* **2024**, *14*, 960.
11. Faruqui, N.; Yousuf, M.A.; Kateb, F.A.; Hamid, M.A.; Monowar, M.M. Healthcare As a Service (HAAS): CNN-based cloud computing model for ubiquitous access to lung cancer diagnosis. *Heliyon* **2023**, *9*.
12. Hossen, R.; Whaiduzzaman, M.; Uddin, M.N.; Islam, M.J.; Faruqui, N.; Barros, A.; Sookhak, M.; Mahi, M.J.N. Bdps: An efficient spark-based big data processing scheme for cloud fog-iot orchestration. *Information* **2021**, *12*, 517.
13. Achar, S.; Faruqui, N.; Bodepudi, A.; Reddy, M. Confimizer: A novel algorithm to optimize cloud resource by confidentiality-cost trade-off using bilstm network. *IEEE Access* **2023**.
14. Kumar, P.; Kumar, R. Issues and challenges of load balancing techniques in cloud computing: A survey. *ACM computing surveys (CSUR)* **2019**, *51*, 1–35.
15. Xu, W.; Jang-Jaccard, J.; Singh, A.; Wei, Y.; Sabrina, F. Improving performance of autoencoder-based network anomaly detection on nsl-kdd dataset. *IEEE Access* **2021**, *9*, 140136–140146.
16. Shi, J.; Fu, K.; Wang, J.; Chen, Q.; Zeng, D.; Guo, M. Adaptive QoS-aware Microservice Deployment with Excessive Loads via Intra-and Inter-Datacenter Scheduling. *IEEE Transactions on Parallel and Distributed Systems* **2024**.
17. Vuillod, B.; Zani, M.; Hallo, L.; Montemurro, M. Handling noise and overfitting in surrogate models based on non-uniform rational basis spline entities. *Computer Methods in Applied Mechanics and Engineering* **2024**, *425*, 116913.
18. Alnagashi, F.A.K.Q.; Rahim, N.A.; Shukor, S.A.A.; Hamid, M.H.A. Mitigating Overfitting in Extreme Learning Machine Classifier Through Dropout Regularization. *Applied Mathematics and Computational Intelligence (AMCI)* **2024**, *13*, 26–35.

19. Salehin, I.; Kang, D.K. A review on dropout regularization approaches for deep neural networks within the scholarly domain. *Electronics* **2023**, *12*, 3106.
20. Zhang, Z.; Xu, Z.Q.J. Implicit regularization of dropout. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **2024**.
21. Poobalan, A.; Sangeetha, S.; Shanthakumar, P. Performance Optimization and Energy Minimization of Cloud Data Center Using Optimal Switching and Load Distribution Model. *Sustainable Computing: Informatics and Systems* **2024**, p. 101013.
22. Buyya, R.; Ilager, S.; Arroba, P. Energy-efficiency and sustainability in new generation cloud computing: A vision and directions for integrated management of data centre resources and workloads. *Software: Practice and Experience* **2024**, *54*, 24–38.
23. Katal, A.; Choudhury, T.; Dahiya, S. Energy optimized container placement for cloud data centers: a meta-heuristic approach. *The Journal of Supercomputing* **2024**, *80*, 98–140.
24. Mongia, V. EMaC: Dynamic VM Consolidation Framework for Energy-Efficiency and Multi-metric SLA Compliance in Cloud Data Centers. *SN Computer Science* **2024**, *5*, 643.
25. Rajagopalan, A.; Swaminathan, D.; Bajaj, M.; Damaj, I.; Rathore, R.S.; Singh, A.R.; Blazek, V.; Prokop, L. Empowering power distribution: Unleashing the synergy of IoT and cloud computing for sustainable and efficient energy systems. *Results in Engineering* **2024**, p. 101949.
26. Sun, Y.; Wang, Z.J.; Deveci, M.; Chen, Z.S. Optimal releasing strategy of enterprise software firms facing the competition from cloud providers. *Expert Systems with Applications* **2024**, *236*, 121264.
27. Khan, A.Q.; Matskin, M.; Prodan, R.; Bussler, C.; Roman, D.; Soylu, A. Cloud storage cost: a taxonomy and survey. *World Wide Web* **2024**, *27*, 36.
28. Nezafat Tabalvandani, M.A.; Hosseini Shirvani, M.; Motameni, H. Reliability-aware web service composition with cost minimization perspective: a multi-objective particle swarm optimization model in multi-cloud scenarios. *Soft Computing* **2024**, *28*, 5173–5196.
29. Chi, Y.; Dai, W.; Fan, Y.; Ruan, J.; Hwang, K.; Cai, W. Total cost ownership optimization of private clouds: a rack minimization perspective. *Wireless Networks* **2024**, *30*, 3855–3869.
30. Moreira, L.F.R.; Moreira, R.; Travençolo, B.A.N.; Backes, A.R. An Artificial Intelligence-as-a-Service Architecture for deep learning model embodiment on low-cost devices: A case study of COVID-19 diagnosis. *Applied Soft Computing* **2023**, *134*, 110014.
31. Debinski, M.; Breitinger, F.; Mohan, P. Timeline2GUI: A Log2Timeline CSV parser and training scenarios. *Digital Investigation* **2019**, *28*, 34–43.
32. Jayaweera, C.; Aziz, N. Reliability of principal component analysis and Pearson correlation coefficient, for application in artificial neural network model development, for water treatment plants. IOP Conference Series: Materials Science and Engineering. IOP Publishing, 2018, Vol. 458, p. 012076.
33. Faruqui, N.; Yousuf, M.A.; Chakraborty, P.; Hossain, M.S. Innovative automation algorithm in micro-multinational data-entry industry. Cyber Security and Computer Science: Second EAI International Conference, ICONCS 2020, Dhaka, Bangladesh, February 15-16, 2020, Proceedings 2. Springer, 2020, pp. 680–692.
34. Racherla, S.; Sripathi, P.; Faruqui, N.; Kabir, M.A.; Whaiduzzaman, M.; Shah, S.A. Deep-IDS: A Real-Time Intrusion Detector for IoT Nodes Using Deep Learning. *IEEE Access* **2024**.
35. Demircioğlu, A. The effect of feature normalization methods in radiomics. *Insights into Imaging* **2024**, *15*, 2.
36. Geem, D.; Hercules, D.; Pelia, R.S.; Venkateswaran, S.; Griffiths, A.; Noe, J.D.; Dotson, J.L.; Snapper, S.; Rabizadeh, S.; Rosh, J.R.; others. Progression of Pediatric Crohn's Disease Is Associated With Anti–Tumor Necrosis Factor Timing and Body Mass Index Z-Score Normalization. *Clinical Gastroenterology and Hepatology* **2024**, *22*, 368–376.
37. Trivedi, S.; Patel, N.; Faruqui, N. NDNN based U-Net: An Innovative 3D Brain Tumor Segmentation Method. In 2022 IEEE 13th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)(pp. 0538-0546), 2022.
38. Ullah, U.; Garcia-Zapirain, B. Quantum machine learning revolution in healthcare: a systematic review of emerging perspectives and applications. *IEEE Access* **2024**.
39. Faruqui, N.; Yousuf, M.A.; Whaiduzzaman, M.; Azad, A.; Alyami, S.A.; Liò, P.; Kabir, M.A.; Moni, M.A. SafetyMed: a novel IoMT intrusion detection system using CNN-LSTM hybridization. *Electronics* **2023**, *12*, 3541.

40. Shahiwala, A.F.; Qawoogha, S.S.; Faruqui, N. Designing optimum drug delivery systems using machine learning approaches: A prototype study of niosomes. *AAPS PharmSciTech* **2023**, *24*, 94.
41. Faruqui, N.; Yousuf, M.A.; Whaiduzzaman, M.; Azad, A.; Barros, A.; Moni, M.A. LungNet: A hybrid deep-CNN model for lung cancer diagnosis using CT and wearable sensor-based medical IoT data. *Computers in Biology and Medicine* **2021**, *139*, 104961.
42. Wang, L.; Ye, W.; Zhu, Y.; Yang, F.; Zhou, Y. Optimal parameters selection of back propagation algorithm in the feedforward neural network. *Engineering Analysis with Boundary Elements* **2023**, *151*, 575–596.
43. Xie, G.; Lai, J. An interpretation of forward-propagation and back-propagation of dnn. Pattern Recognition and Computer Vision: First Chinese Conference, PRCV 2018, Guangzhou, China, November 23-26, 2018, Proceedings, Part II 1. Springer, 2018, pp. 3–15.
44. Cong, S.; Zhou, Y. A review of convolutional neural network architectures and their optimizations. *Artificial Intelligence Review* **2023**, *56*, 1905–1969.
45. Paula, L.P.O.; Faruqui, N.; Mahmud, I.; Whaiduzzaman, M.; Hawkinson, E.C.; Trivedi, S. A novel front door security (FDS) algorithm using GoogleNet-BiLSTM hybridization. *IEEE Access* **2023**, *11*, 19122–19134.
46. Hossain, M.E.; Faruqui, N.; Mahmud, I.; Jan, T.; Whaiduzzaman, M.; Barros, A. DPMS: Data-Driven Promotional Management System of Universities Using Deep Learning on Social Media. *Applied Sciences* **2023**, *13*, 12300.
47. Kaur, H.; Anand, A. Review and analysis of secure energy efficient resource optimization approaches for virtual machine migration in cloud computing. *Measurement: Sensors* **2022**, *24*, 100504.