Journal of Al-Azhar University Engineering Sector

Vol. 20, No. 77, October 2025, 1343 - 1362

AN ADAPTIVE FRAMEWORK FOR MITIGATING JOB FAILURES IN CLOUD COMPUTING VIA MACHINE LEARNING AND DYNAMIC RESOURCE MANAGEMENT

Ahmed ELkaradwy[1,2*], Ayman Elshenawy[1,3], Hany Harb[1]

[1] Systems and Computers Engineering Department, Al-Azhar University, Nasr City, Cairo, Egypt.

[2] National Telecommunications Regulatory Authority (NTRA), Giza, 12577, Egypt.

[3] Software Engineering Department, Faculty of Information Technology, Al-Ahliyya Amman University, Amman, Jordan.

* **Correspondence**: aelkaradawy@ tra.gov.eg

Citation:

A. ELkaradwy, A. Elshenawy, H. Harb" An Adaptive Framework for Mitigating Job Failures in Cloud Computing via Machine Learning and Dynamic Resource Management", Journal of Al-Azhar University Engineering Sector, vol. 20(77), pp. 1343-162, 2025.

Received: 15 June 2025

Revised: 02 August 2025

Accepted: 17 August 2025

Doi: 10.21608/auej.2025.410150.1914

ABSTRACT

Cloud systems are increasingly challenged by job failures caused by fluctuating workloads and unpredictable resource availability. While much of the existing research emphasizes failure prediction, less attention has been given to implementing mitigation strategies post-prediction. This paper introduces an Adaptive Failure Mitigation Framework that combines machine learning-based failure prediction with adaptive resource management techniques. Using historical logs and system metrics, the framework applies the Random Forest algorithm to classify potential failure types. Based on these predictions, a rule-based Cloud Manager dynamically executes appropriate corrective actions tailored to job priority, predicted failure category, and real-time node availability. The corrective strategies include dynamic resource reallocation, job rescheduling, live migration to alternative servers, and replication of critical tasks to ensure continuity and data safety. Evaluations conducted using the Google 2019 dataset show the framework achieves a high classification accuracy of 99.98% in identifying failure types and successfully mitigates 91.2% of predicted job failures.

KEYWORDS: Adaptive Scheduling, Cloud Computing, Failure Mitigation, Reliability, Resource Management

إطار تكيفي للتخفيف من حالات فشل المهام في الحوسبة السحابية باستخدام التعلم الآلي وإدارة الموارد بشكل ديناميكي

أحمد القرضاوي[١،٢*]، أيمن الشناوي[٣،١]، هاني حرب[٢]

[١] قسم هندسة النظم والحاسبات، كلية الهندسة، جامعة الأزهر، القاهرة، ١١٨٨٤، مصر

[٢] الجهاز القومي لتنظيم الاتصالات، الجيزة، ١٢٥٧٧، مصر

[٣] قسم هندسة البرمجيات، كلية تكنولوجيا المعلومات، جامعة عمان الأهلية، عمّان، الأردن

البريد الالكتروني للباحث الرئيسي: aelkaradawy@ tra.gov.eg

الملخص

تواجه أنظمة الحوسبة السحابية تحديات متزايدة ناتجة عن فشل تنفيذ بعض مهام المستخدمين (Cloud Jobs)، ويرجع ذلك غالبًا إلى عدم توفر الموارد اللازمة في الوقت المناسب، نتيجة لتقلبات الطلب وزيادة أعباء العمل بشكل غير متوقع، مما يؤدي إلى انخفاض مستوى الأداء ورضا المستخدمين. وعلى الرغم من أن العديد من الأبحاث ركزت على تطوير تقنيات التنبؤ بالفشل، إلا أن هناك قصورًا واضحًا في معالجة ما بعد التنبؤ، أي تنفيذ استراتيجيات فعالة للتعامل مع الفشل المتوقع. في هذا السياق، يقترح هذا البحث إطارًا تكيفيًا لمعالجة حالات الفشل قبل وقوعها، يجمع بين خوارزميات التعلّم الآلي وإدارة الموارد بطريقه ديناميكية. يعتمد الإطار المقترح على تحليل البيانات التشغيلية السابقة وأسلوب إدارة فعال للموارد المتاحة وذلك باستخدام خوارزمية (Random Forest) لتصنيف أنواع الفشل المحتمل حدوثها بدقة عالية. بناءً على نتائج التنبؤ، يقوم العقل المسئول (Cloud manager) بتنفيذ إجراءات تصحيحية بشكل تلقائي وديناميكي، اعتمادًا على قواعد محددة تأخذ في الاعتبار أولوية المهمة، ونوع الفشل المتوقع، ومدى توفر الموارد في الوقت الفعلي. تشمل استراتيجيات المعالجة: إعادة تخصيص الموارد، إعادة جدولة المهام، النقل المباشر للمهام إلى خوادم بديلة، وتكرار المهام الحرجة لضمان استمرارية الخدمة وسلامة البيانات. أظهرت التقييمات التي تم إجراؤها باستخدام مجموعة بيانات Google لعام ٢٠١٩ أن الإطار يحقق دقة تصنيف عالية تبلغ ٩٩,٩٨٪ في تحديد أنواع الفشل، ويُساهم في تقليل نسبة الفشل المتوقع بنسبة ٩١,٢٪، ما يؤكد فعاليته في تحسين استقرار وأداء بيئات الحوسبة السحابية.

الكلمات المفتاحية : التنظيم الذكي للمهام ، الحوسبة السحابية، تخفيف فشل المهام ، الاعتماديه، إدارة الموارد.

1. INTRODUCTION

Cloud computing environments offer scalable and flexible infrastructure to support diverse applications with varying resource demands. As these systems expand in size and complexity, efficient resource management becomes increasingly vital to maintaining reliable service delivery [1,2]. Despite advances in cloud scheduling and provisioning, job failures remain a pressing concern. These failures frequently result from overutilization, underutilization, or sudden changes in resource availability [2-4]. When a job fails, the system suffers not only from service interruptions but also from unnecessary consumption of critical resources, including CPU cycles, memory, and storage space. In current cloud platforms, failed jobs consume substantial resources throughout their execution lifecycle, even when they ultimately do not complete successfully. This leads to degraded performance and lower throughput for cloud service providers (CSPs) clusters. Additionally, high failure rates disrupt workload continuity and reduce the efficiency of system-wide resource utilization [4-5]. Minimizing task or job failure is, therefore, essential to improving cloud reliability and reducing operational waste. Traditional mitigation strategies often rely on job redundancy, where backup instances are pre-deployed to take over in case of failure. While this approach reduces service interruptions, it increases infrastructure and maintenance costs for CSPs and introduces potential service delays for customers [4-7]. Redundancy also leads to inefficient resource allocation, particularly in large-scale and dynamic environments with fluctuating demands. Recent research has focused on predictive analytics as a promising alternative. By identifying failure-prone jobs before execution, cloud platforms can proactively allocate resources, reschedule jobs, or apply alternative strategies to enhance execution success. Predictive models not only reduce the need for redundant infrastructure but also allow for more intelligent scheduling decisions that improve overall system performance [7-8].

However, there is a clear research gap, since existing predictive frameworks face several limitations. Many models use overly simplistic learning techniques that fail to capture dependencies across multiple data dimensions. Some approaches focus exclusively on failure prediction while overlooking the importance of post-prediction actions [7-9]. Others limit their input to resource demand parameters, neglecting real-time resource usage data, which can offer crucial insights into failure behavior [10-12]. This study addresses these limitations by developing a data-driven framework that incorporates machine learning (ML) for failure type prediction and adaptive cloud resource management. The proposed solution examines four critical dimensions: application-level demands, infrastructure capabilities, configuration parameters (such as priority and scheduling policies), and actual resource usage during execution. By analyzing their interrelationships, the framework identifies significant failure patterns and proposes accurate early-stage predictions of job outcomes. In addition to prediction, the framework introduces a set of proactive remedy actions to protect jobs that are likely to fail. These include dynamic resource reallocation, intelligent rescheduling, job migration, and selective replication for high-priority tasks. Since the selection and sequence of these actions significantly impact performance and resource efficiency, the decision-making process is modeled as a constrained optimization problem. A heuristic strategy is introduced to determine the most effective combination of actions while minimizing response time and reducing overhead. The key contributions of this paper can be summarized as follows:

1) A predictive failure classification model is built based on the Random Forest (RF) algorithm to highly predict multiple job failure types, including EVICT, FAIL, LOST, and KILL, and identify failure risks.

2) An adaptive remediation engine is added to the framework by a rule-based decision process to evaluate each job's failure probability and urgency level and select the most suitable remedy from a set of predefined strategies such as replication, live migration, resource scaling, or rescheduling.

3) The Adaptive Failure Mitigation Framework (AFMF) boosts cloud performance, optimizes resource utilization, and strengthens resilience. Adaptation to workload demands and system constraints ensures efficiency and scalability for proactive fault management in large-scale environments.

This paper follows a structured flow. Section 2 reviews existing literature on cloud failure prediction and resource scheduling. Section 3 introduces the mathematical modeling of failure type classification using the Random Forest algorithm, along with the heuristic-based remedy strategy and a comprehensive description of the dataset. Section 4 presents the proposed framework in detail, explaining both the classification model and the rule-driven remediation techniques, in addition to analyzing trace data to support the design. Section 5 outlines the experimental setup, reports the obtained results, and evaluates the overall performance of the system. Finally, Section 6 concludes the study and discusses potential directions for the future.

2. RELATED WORK

The primary goal of this section is to review recent studies relevant to the proposed research. Numerous researchers have applied statistical, machine learning, and deep learning (DL) techniques to large-scale datasets, especially Google Cluster trace, to predict job or task failures. While many solutions forecast failure probabilities with high accuracy, few use those predictions to guide targeted corrective actions [10]. This work advances the field by integrating precise failure type classification with real-time decision-making for dynamic resource remediation [13,14].

A critical challenge in cloud computing has been addressed: the frequent occurrence of task failures caused by the complex and dynamic nature of these systems [15]. They found that traditional fault-tolerance methods could not handle such challenges effectively. To solve this, they developed an ML framework that predicts failures in advance. They used CatBoost with a genetic algorithm to select important features and applied a tuned multilayer perceptron for prediction. Their model, tested on the Google 2019 dataset, reached 98.38% precision, 99.62% recall, and 99.31% accuracy. These results confirm the model's strength in early fault prediction. The AI-based Cloud Failure Prevention Algorithm (ACFP), which utilizes multiple ML and DL models to analyze traces from the Google 2019 cluster has been introduced [16]. ACFP aims to predict job failures using advanced techniques such as CatBoost, LightGBM, Gradient Boost, and RF. Each of these models delivered perfect scores in accuracy, F1 score, and precision during evaluation. Once evaluated, the algorithm automatically selects the most accurate model based on its predictive performance. It then relies on critical parameters, such as resource utilization, event type, and scheduler behavior, to generate actionable recommendations for preventing failures. Experimental outcomes confirmed that ACFP significantly improved prediction accuracy, reducing the failure rate from 22.76% to just 2.67%.

A proactive task scheduling framework tailored for cloud environments, with a specific focus on managing task failures has been presented [17]. The authors analyzed three large-scale datasets from Google, Alibaba, and Trinity to gain a deeper understanding of failure patterns. Based on this analysis, they developed a failure-aware scheduling model that predicts task termination in real time and applies appropriate corrective measures. Their results were compelling: using the Google Cluster 2011 dataset, ANN and CNN models achieved prediction accuracy of 94% and 92%, respectively. Additionally, by applying corrective actions to tasks likely to fail, particularly in the Alibaba dataset, the framework preserved up to 40% of those tasks and reduced resource waste, including CPU and RAM. An advanced approach to resource scheduling in edge-cloud environments, aiming to optimize performance and enhance service quality has been explored [18]. To meet these goals, the authors combined Deep Reinforcement Learning (DRL) for workflow preprocessing with multi-objective optimization, enabling Pareto-optimal scheduling decisions. To

address the growing complexity of resource coordination in modern cloud infrastructures, a machine learning-based framework aimed at improving workload distribution and boosting overall operational efficiency has been proposed [19]. To address this, the authors built a neural network model capable of processing real-time sensor data across the infrastructure. This model adapts to changing environmental conditions and enhances both resource utilization and task execution across virtual machines.

The issue of fault tolerance in dynamic cloud environments, where changes in workload, system states, and configurations often lead to unpredictable failures has been tackled [20]. These variations make accurate failure prediction difficult, especially in datasets where failed jobs are rare. To overcome this, the authors proposed using an ensemble ML technique. They tested their approach using the Google Cluster 2019 dataset. Results showed an impressive accuracy of 98.92%, confirming their strength in predicting failures in dynamic and imbalanced cloud settings. A thorough statistical analysis of resource usage patterns in distributed computing environments has been carried out [21]. Their investigation revealed clear behavioral distinctions between successful and failed tasks. Based on this observation, they utilized the XGBoost classifier to predict task failures. The model delivered strong results, with a precision of 92% and a recall of 94.8%, confirming its reliability and practical potential. In a related effort, a predictive resource management framework designed for both cloud and volunteer computing systems has been introduced [22]. The framework features a dual-mode auto-scaling mechanism that integrates both reactive and proactive strategies. Central to the proactive mechanism is a Hidden Markov Model (HMM) that anticipates future resource behavior. Tuli et al. [23] presented CILP, a co-simulation-driven imitation learning platform designed to address the challenges of virtual machine provisioning. Their framework divides the provisioning process into two core components: predicting workload demands and optimizing provisioning actions. Experimental results showed that CILP enhanced resource utilization by up to 22%, improved QoS by 14%, and reduced execution costs by 44% when compared to conventional methods.

Building on the need for balanced cloud resource management, DCOTS, a scheduling algorithm that simultaneously considers user deadlines and budget limitations while optimizing overall provider performance has been developed [24]. The approach integrates cost and time constraints into a unified decision-making process, creating an equitable solution for both clients and service providers. The serious consequences of task failures in cloud computing, including service interruptions and poor user experience have been addressed [25]. To mitigate these risks, they proposed a DL-based approach using a Bidirectional Long Short-Term Memory (Bi-LSTM) model. This architecture leverages the sequential nature of task execution data to better capture dependencies. A comprehensive evaluation of the Google 2019 dataset has been performed [26]. They focused on terminated jobs and job-level workload characteristics in large-scale clusters. The authors emphasized the importance of understanding workload behaviors to boost system throughput and optimize resource allocation. Traditional ML models for failure prediction using Google's Borg cluster traces from 2019 h has been investigated [27]. Given the complexity of modern cloud systems, the authors recognized the urgent need to reduce downtime and associated costs. They tested 4 ML algorithms, Decision Tree (DT), RF, Gradient Boosting (GB), and Logistic Regression (LR), to develop failure classifiers. Their DT model delivered the best results, achieving 98.79% accuracy.

Early task failure prediction using a clustering-based strategy has been [28]. They grouped jobs with similar characteristics using Fuzzy C-Means (FCM), treating each cluster as a distinct dataset for model training and evaluation. They compared the performance of three algorithms: Extreme Learning Machine (ELM), Support Vector Machine (SVM), and LR. ELM and SVM performed best, each reaching an accuracy of 89.96%, while LR followed with 80.93%. A multi-layer Bi-LSTM network to predict both job and task failures has been employed [29]. They used a combination of static and dynamic job features during training. The model achieved up to 93% accuracy in predicting task failures and up to 87% for job failures, confirming its effectiveness across different failure types. The job failures by examining both workload patterns and system attributes has been analyzed [30]. They created three DL models to classify job termination states. Their models achieved 94.4% accuracy for job classification and 76.8% for task classification, indicating strong performance with room for refinement at the task level. The features associated with application failures in cloud environments has been identified [31]. They developed an LSTM-based model that predicted application termination status with an accuracy of up to 87%. The use

of static offline models for job termination prediction and introduced an Online Sequential Extreme Learning Machine (OS-ELM) has been challenged [32]. This model updates continuously as it receives new data. It achieved 93% accuracy, showing the promise of real-time learning methods in dynamic cloud settings.

The features that most strongly correlate with job and task failures in cloud environments has been studied [33]. They examined resource usage logs and job event data. Using a Recurrent Neural Network (RNN) model, they reached an accuracy of around 84%, highlighting the value of sequence modeling in failure prediction. **Table 1** presents an overview of the key studies addressing failure prediction and resource management within cloud clusters. These contributions have significantly shaped the state of the art, yet many of them approach predictive modeling and adaptive recovery as distinct processes. In several cases, solutions rely heavily on redundancy-based strategies, while others limit themselves to a single type of remedial action. Such limitations reveal a consistent gap in the literature, a lack of comprehensive frameworks that unite accurate failure forecasting with flexible, context-aware decision-making for resource control. In response to this challenge, the present work proposes an integrated approach that combines an RF classification model with a rule-guided remediation mechanism.

Table 1: Key Research Contributions on Managing Failures and Resources in Cloud Environments

Models used	Dataset	Modeling Insights	Test Env.	Corrective actions applied?	Accuracy	Ref.
Tuned multilayer perceptron	Google	• Applied a Single model to classify jobs as failed or successful.	-	No	99.31%	[15] (2022)
ANN and CNN	Trinity Alibaba Google	• Used two separate models to detect failures, and no failure classification was implemented	Local server	Yes, saved 40% of failed tasks	ANN = 94% CNN = 92%	[17] (2021)
XGBoost, AdaBoost, GD, And Light GBM	Google	• Tested multiple algorithms individually • Failure classification was not applied	-	No	XGBoost (Best) = 98.92%	[20] (2024)
SVM, LSTM, and Bi-LSTM	Google	• Developed a dynamic task migration strategy that activates after failure detection. • ML & DL models are applied.	CloudSim	Yes	Bi-LSTM (Best) = 93%	[25] (2024)
DT, RF, GBoost, and LR	Google	• Applied 4 ML algorithms to predict failures, and only one model is used at a time • No type of failure classification applied	-	No	DT (Best) = 98.79%, for task failures	[27] (2023)
Bi-LSTM	Google	• Applied DL algorithm to predict job and task failure • Single model used. • No failure classification applied.	-	No	93%, for task failures and 87%, for job failures	[29] (2020)
LR and KNN	Bit Brains dataset	• Tested 2 ML Algorithms individually • No failure classification applied	Google Colab Pro+	No	KNN= 99%, LR = 95%.	[34] (2022)
SVM, LDA, CART, RF, KNN	NERSC	• Provided failure specifics for I/O related systems and components • Tested the models individually	Local server	No	SVM(Best)= 90.76%	[35] (2019)
RF, DT, NB, and QDA	Google Mustang Trinity	• Applied ML models separately. • No detection of failure types	Google Colab Pro+	No	DT (Best) on Google dataset = 99%.	[36] (2022)
Ensemble Model, ANN, and KNN	Google	• Used an ensemble technique. • No detection of failure types. • ML & DL models are applied.	AWS SageMaker	No	Ensemble = 99.13%.	[37] (2023)

3. METHODOLOGY

3.1. Mathematical Model for Failure Type Classification Using RF

Reliable classification of cloud job failures supports smarter resource allocation and faster recovery. A mathematical model built on the RF algorithm enables precise identification of failure types based on historical patterns. The model relies on structured input features that capture key attributes of each job, forming the foundation for accurate predictions.

- **Input Representation**

Each cloud job ***j*** is represented by a feature vector:

$$x_j = [x_{j1}, x_{j2}, \dots, x_{jd}] \in \mathbb{R}^d$$

Where:

- d is the number of features describing the job.
- x_{jk} denotes the *k-th* feature of job j.
- The feature vector captures historical metrics such as CPU usage, memory consumption, scheduling class, and prior failure history.

- **Failure Type Classification Objective**

Let:

- ✓ $F = \{\text{FAIL, KILL, LOST, EVICT}\}$ is the set of possible failure classes
- ✓ $f_j \in F$: the actual failure class for job j
- ✓ $\hat{f}_j \in F$: the predicted failure class for job *j*, produced by the Random Forest classifier.

- **RF Prediction Process**

The RF classifier C consists of an ensemble of M decision trees $\{T_1, T_2, \dots, T_M\}$. Each decision tree T_m independently predicts a class label for the input job:

$$T_m(x_j) \in F, \quad \text{for } m = 1,2,\dots,M$$

To determine the final predicted class $\hat{f}_j$, the model uses majority voting across all trees, as shown in Equation 1:

$$\hat{f}_j = \arg\max_{f \in F} \sum_{m=1}^{M} 1(T_m(x_j) = f) \qquad (1)$$

Where $1(\cdot)$ is the indicator function that counts votes:

$$1(T_m(x_j) = f) = \begin{cases} 1, & \text{if tree m predicts class f} \\ 0, & \text{Otherwise} \end{cases}$$

This model aggregates the decisions of multiple trees to robustly classify the failure type of a given job. By leveraging the diversity of the ensemble, the RF mitigates overfitting and increases prediction accuracy, particularly in large-scale, dynamic cloud environments.

3.2. Heuristic Remedy Strategy and Mathematical Modeling

The remedy framework employs a heuristic-based strategy that selects corrective actions using conditional logic tied to job attributes and system resource states.

- **Critical jobs** are handled through replication. The system replicates the job across multiple nodes *n*, where $n \geq 2$, ensuring redundancy, as defined in Equation 2:

$$\text{Replication Nodes} = \{\text{Top n nodes by availability}\} \quad (2)$$

- **High-priority jobs** are migrated to better-suited nodes. The selection criterion ensures availability as described in Equation 3:

$$\text{Available CPU} \geq \text{CPU}_{\text{job}} \wedge \text{Available RAM} \geq \text{RAM}_{\text{job}} \quad (3)$$

- **Medium-priority jobs** use dynamic scaling. The system calculates a resource scaling factor according to Equation 4:

$$\text{Scale Factor} = \min\left(\frac{\text{CPU}_{\text{node}}}{\text{CPU}_{\text{job}}}, \frac{\text{RAM}_{\text{node}}}{\text{RAM}_{\text{job}}}\right) \quad (4)$$

The job's demands are scaled as:

$$\text{CPU}_{\text{adjusted}} = \text{CPU}_{\text{job}} \times \text{Scale Factor}$$

$$\text{RAM}_{\text{adjusted}} = \text{RAM}_{\text{job}} \times \text{Scale Factor}$$

- **Low-priority jobs** are rescheduled. They are deferred until resource conditions improve, as determined by Equation 5:

$$\text{Reschedule Time} = t + \Delta t \quad (5)$$

Where Δt is a future slot based on system load.

3.3. Dataset Description

The Google Cluster dataset offers a detailed view of resource usage across eight cloud clusters, each containing approximately 12,000 machines as of May 2019 [4]. Released in early 2020, the dataset spans around 2.4 terabytes and captures jobs and tasks submitted throughout May 2019 [38,39]. It focuses specifically on resource requests and consumption patterns, as summarized in **Table 2** [2]. Each data entry represents a group of machines within a single cluster [40-42]. A typical Google cluster consists of numerous servers, physically arranged in racks and interconnected via high-bandwidth networking. Inside a cluster, machines are further grouped into what's known as a "cell." This cell operates under a unified management system that handles the scheduling and allocation of computing tasks. In this environment, resource requests come in two main forms: jobs and alloc sets [43-45]. A job includes one or more tasks and defines the computational work a user wants to perform [4]. Each task acts as a Linux program that runs on a single machine and may involve several subprocesses depending on its complexity [46,47].

Table 2: Details of Google 2019 Clusters.

Date	May 2019
Dataset Duration	31 days
Cloud Clusters	8
Total No. of Machines	96.4k
No. of Machines per Cluster	12.0k
Hardware platforms	7
Machine shapes	21
Priority values	0–450
Alloc sets	Y
Job dependencies	Y
Batch queueing	Y
Vertical scaling	Y
Format	BigQuery tables

When a job is submitted in the Google Cloud system, it may optionally request an alloc set for its execution. In such cases, each task within that job draws its required resources from one of the alloc instances that belong to the specified alloc set. If no alloc set is provided, the system allows tasks to consume resources directly from the machine where they are deployed [4]. This process introduces multiple execution scenarios, all of which are visually summarized in **Fig. 1** [48,49]. The dataset documentation categorizes jobs and alloc sets as "collections" while tasks and alloc instances are referred to as "instances". Given the massive scale of the full dataset, a representative sample was selected to simplify processing and analysis [48-50]. During the lifecycle of a collection or instance, termination can occur through one of five event types:

- **Finish**: The jobs are successfully executed and completed as expected [4].
- **Evict**: The system forcibly removes the task for several reasons, including hardware malfunctions, mandatory operating system updates, preemption by higher-priority tasks, or overuse of system resources, where the actual demand exceeds the available capacity, prompting the scheduler to free up space [17,38].
- **Kill**: The user manually cancels the task [4]. This can also happen indirectly if a parent task is terminated, resulting in the automatic cancellation of its child tasks [49,51].

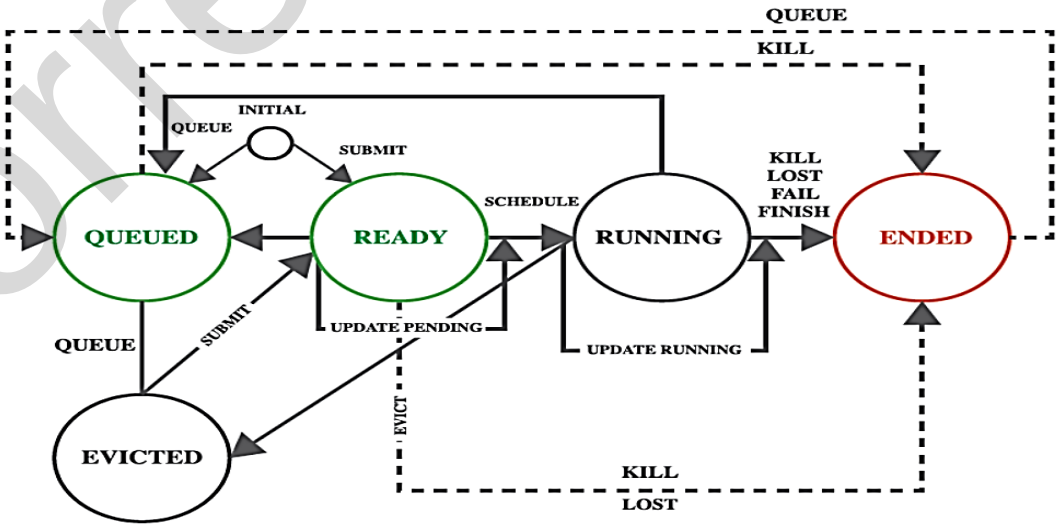

Fig. 1 : Job lifecycle [4]

- **Fail:** The task terminates unexpectedly due to internal errors, such as segmentation faults or exceeding resource limits. These failures often result from memory leaks, configuration errors, or other runtime issues [4,49].
- **LOST:** The system expected a termination event to occur, but the relevant log data was missing or corrupted, making it unclear how or when the task ended [4,38].

3.4. Google Priority Tier Classification

Google's cloud infrastructure categorizes jobs into distinct priority tiers, each with specific service guarantees and associated costs. The scheduler manages jobs based on job priority, which is grouped into structured tiers. Each tier reflects the job's importance and influences how resources are allocated and protected under varying load conditions [4,52]. At the lowest level is the Free Tier, which includes jobs with a priority value of 99 or below. These jobs are not billed and come with no Service Level Objectives (SLOs), meaning they receive minimal scheduling guarantees. Next is the Best-effort Batch (BEB) Tier, covering jobs with priorities ranging from 110 to 115. These jobs are managed by the batch scheduler, incur minimal charges, and, like the free tier, are not tied to any formal SLOs. The Mid-tier sits just above, including jobs with priorities from 116 to 119 [4,53]. These tasks benefit from weaker SLOs compared to higher-tier jobs and are charged less than production workloads. The Production Tier, which spans priorities from 120 to 359, consists of high-availability jobs that are critical to user-facing services. These jobs are billed at the standard rate and enjoy stronger guarantees [4]. To meet their SLOs, the system may preempt lower-priority jobs if necessary [52-54]. At the top of the priority scale lies the Monitoring Tier, which includes jobs essential for the stability and observability of the cloud infrastructure. These tasks often monitor other services and have the highest priority, marked by values of 360 or greater in the 2019 trace [55,56]. **Table 3** below is a summary table that presents the job priority tiers.

Table 3: A summary of the job priority tiers

Tier Name	Priority Range	Description	SLOs	Charges
Free Tier	≤ 99	Lowest-priority jobs with no service guarantees.	None	None
Best-effort Batch	110–115	Batch-scheduled with minimal charges.	None	Low
Mid-tier	116–119	Medium-priority with reduced internal cost.	Weak	Moderate
Production Tier	120–359	High-availability jobs.	Strong	Full
Monitoring Tier	≥ 360	Very Critical or sensitive jobs.	Highest	Highest

4. PROPOSED ADAPTIVE FAILURE MITIGATION FRAMEWORK

In this study, we present the AFMF, specifically designed to address failure events in dynamic cloud computing environments. The framework is designed to accurately identify the specific failure type for each job, categorizing it as one of the following: Evict, Fail, Lost, or Kill. This classification enables the system to apply the most appropriate mitigation strategy for each failure case, improving operational efficiency and fault tolerance. As illustrated in **Fig. 2**, the framework comprises three main modules: data preprocessing, failure type classification, and remedy decision-making.

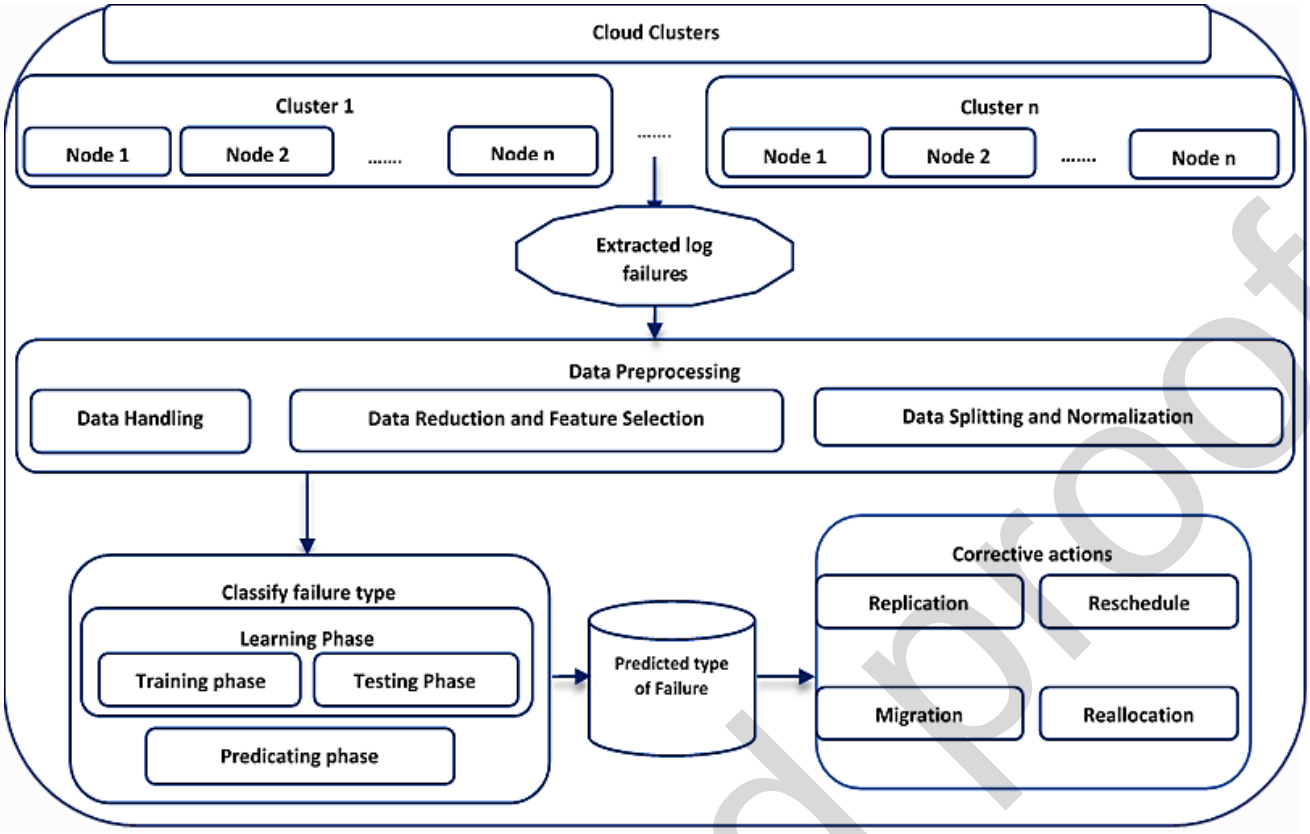

Fig. 2: Adaptive Failure Mitigation framework

We utilize the RF algorithm to predict failure based on historical job execution data. Using these predictions, a rule-based Cloud Manager (CM) selects and applies the most suitable recovery action. These adaptive strategies help maintain workload continuity, optimize resource utilization, and enhance system reliability. Notably, AFMF remains cloud-platform agnostic, allowing it to be adapted to any dataset that includes relevant failure-related information. However, minimal configuration adjustments may be necessary to tailor it to the unique characteristics of specific cloud centers.

4.1 Data Preprocessing:

Data preprocessing is a critical step in preparing raw information for ML applications. The process began by importing the dataset using the Pandas library, with the first column set as the index to facilitate cleaner referencing and organization. One of the most important preprocessing steps focused on the *resource_request* column, which stored CPU and memory values in a semi-structured string format, such as {type: REQUEST, cpu: 2.0, memory: 512}. Since this format is not suitable for direct numerical analysis, a custom parsing function was developed to extract the CPU and memory values. The function verified the structure of each entry, retrieved the relevant values, and replaced any invalid or malformed records with NaN. To maintain data consistency, missing values across all numerical columns were then addressed. Instead of discarding incomplete rows, which could reduce valuable training data, the missing entries were replaced with the meaning of their respective columns. This approach preserved the dataset's statistical distribution while ensuring that no gaps disrupted the model's learning process. Following this, the index was reset to ensure structural consistency across all records, and a quick validation confirmed that no missing values remained.

The data cleaning phase involved eliminating columns that added little or no predictive value. Text-heavy fields such as user, scheduler_name, and high-cardinality attributes like machine_id were removed due to their limited relevance and the unnecessary complexity they introduced. Additional columns, including tail_cpu_usage_distribution, event, and collection_logical_name, were also excluded because they were either deeply nested or redundant. After cleaning the data, the focus shifted to engineering features that could better represent the underlying system behavior. Resource usage information was extracted from the resource_request

and maximum_usage columns. Using the earlier parsing function, memory and CPU usage values were pulled and placed into new columns. These metrics offered a clearer and more structured representation of each job's resource footprint, which is crucial for identifying failure patterns. To better understand relationships between variables, a correlation matrix was generated and visualized.

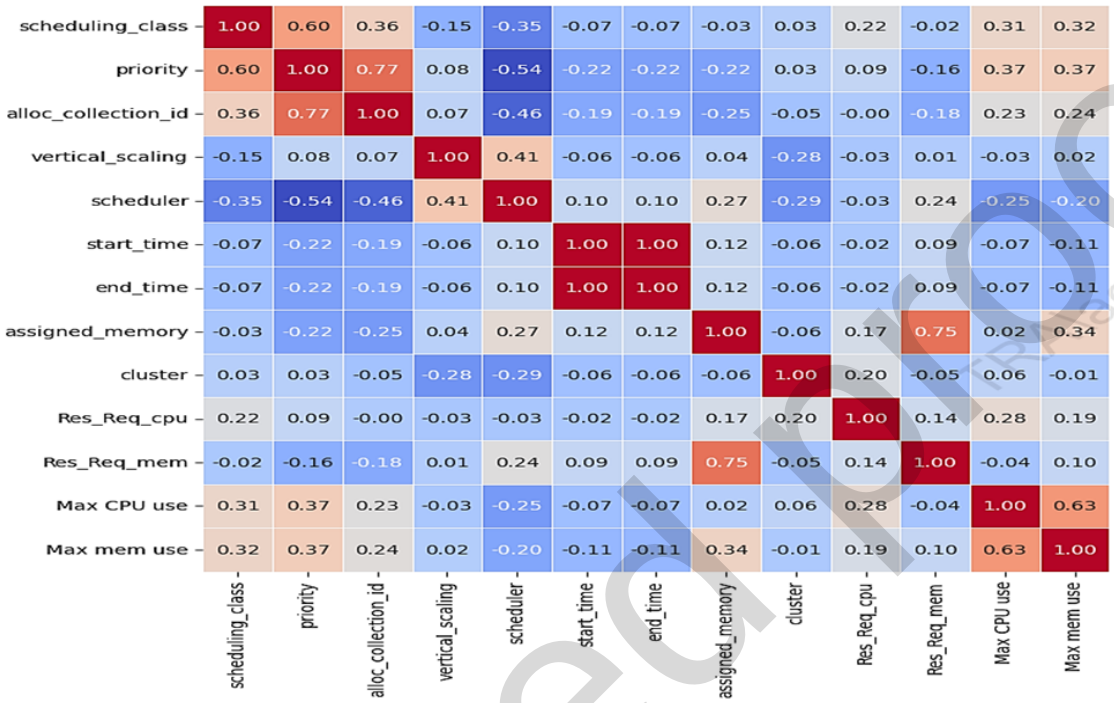

Fig. 3: Correlation between Features

Fig. 3 presents a heatmap that captures the correlation patterns among various features in the dataset. This visualization helps clarify the relationships and interactions between those attributes. A clear and strong positive correlation appears among memory-related features, including assigned_memory, requested_memory, and maximum_memory_usage. This finding implies that memory allocation strategies are closely aligned with actual usage levels, reflecting consistent planning across jobs. The same trend is evident in CPU-related attributes. Both requested_CPU and maximum_CPU_usage are highly correlated, which suggests that resource provisioning for computational tasks generally matches the system's operational demands. Such alignment indicates that CPU planning is effectively synchronized with workload requirements. The correlation analysis also reveals a notable positive association between priority and alloc_collection_id. This relationship suggests that higher priority jobs often follow predefined allocation patterns, pointing to the presence of structured or policy-based scheduling mechanisms. In contrast, scheduling_class shows a negative correlation with both priority and scheduler. This means that jobs assigned to lower scheduling classes are more likely to receive higher priority, possibly due to system rules that prioritize specific job types. These jobs may be managed differently to ensure performance or meet predefined constraints. Some features, including start_time, end_time, and vertical_scaling, display weak correlations with other variables. Their limited associations suggest that these attributes have little influence on resource requests or actual usage, making them less significant from a predictive modeling perspective.

Once the new features were constructed, the dataset was narrowed down to retain only the most relevant predictors. All features except for the failure label were used to build the input matrix, while the failed column was designated as the prediction target. For effective model evaluation, the dataset was split into two parts: 70% for training and 30% for testing. This division ensured that the model could learn from a significant portion of the data while reserving enough examples for reliable performance evaluation.

Fig. 4: Data preprocessing steps

Before feeding the data into the model, feature scaling was applied. Using standard normalization, each feature was transformed to fall within a similar range. The scale was fitted on the training data and then applied to the test data to maintain consistency between the two subsets. **Fig. 4** summarizes the steps involved in the data preprocessing process.

4.2 Type of Failure Prediction Method

Classifying failure types is essential for identifying the causes of job interruptions in cloud computing. A predictive model was developed to assign each failure to one of four categories: EVICT, FAIL, LOST, or KILL, enabling service providers to apply precise corrective actions that improve reliability and resolution speed. This is especially beneficial for large-scale providers such as Alibaba, Amazon, and Microsoft, where early intervention enhances stability and user experience. After preprocessing, the model was trained using an RF classifier with 100 DTs, chosen for its strength in handling structured, high-dimensional data and its reliability in classification tasks. Historical execution patterns were used to build a strong predictive base, with a fixed random seed ensuring consistency. The trained model was evaluated on unseen data to test its generalization ability, using a confusion matrix to measure classification accuracy and a detailed report on precision, recall, and F1-score for each category.

4.3 Dynamic Remediation Techniques for Predicted Job Failures in Cloud Environments

To address predicted failures in cloud-based workloads, the simulation framework incorporates a module called CM, designed specifically for dynamic remediation. This class orchestrates intelligent decision-making by evaluating both job urgency and system resource availability. The core logic begins by sorting all jobs based on two primary factors: their assigned priority (ranging from Low to Critical) and their estimated probability of failure. This dual-ranking strategy ensures that the most critical and failure-prone jobs are addressed before others. At the same time, compute nodes are evaluated and ranked according to their current availability and capacity, enabling optimal job placement. Once prioritization is complete, the CM applies targeted remediation strategies tailored to the priority level of each job. Critical jobs activate a replication strategy, creating backup instances across the two most suitable nodes to maximize fault tolerance. High-priority jobs trigger migration; the system searches for a more capable node with matching resource availability and transfers the job accordingly. For jobs labeled as medium priority, the framework dynamically scales the allocated CPU and RAM, aligning resource demands with what the node can realistically provide. Low-priority jobs are not immediately executed. Instead, they are deferred and rescheduled, allowing the system to manage them when resources become more favorable.

Each remediation decision is recorded in a centralized action log, which serves as both a diagnostic tool and a performance tracker. The CM tracks how often each remediation strategy is applied. It also maintains a running count of jobs that are completed after receiving a remedy. Those that fail despite intervention are flagged separately, providing transparency in the decision-making process and allowing for more informed evaluation. After applying these strategies, the system moves to evaluate its effectiveness. This is achieved by invoking the *find_remedy_actions()* method, which processes the complete job list against the available infrastructure and determines an appropriate course of action for each case. The number of jobs that successfully terminate following a remedy is treated as a key performance indicator, reflecting the system's operational efficiency under dynamic conditions. To quantify the system's impact more clearly, the framework calculates the overall remediation success rate. This metric is defined as the percentage of jobs that were successfully handled out of the total workload.

4.4 Traces Analysis

The dataset sample utilized in this study was obtained from Kaggle, a widely recognized platform for data sharing and collaboration [4,46]. It contains job-level logs from large-scale Google Cloud data centers and captures a wide range of operational metrics. The dataset was compiled from eight interconnected tables and includes thirty-three features, covering various dimensions of cloud job execution such as resource allocation, scheduling behavior, and task performance [56-59]. Among the records, a total of 167,900 jobs were labeled as failed. These instances form the core focus of this study. **Table 4** and **Fig. 5** present the distribution of failure categories and highlight an imbalance in the frequency of different failure types [49]. Each label corresponds to a specific failure scenario that may occur during the execution of jobs in the cloud infrastructure. Since every failed job must belong to exactly one of these categories, the classification task is defined as a multiclass problem [60-62]. To address the class imbalance in our dataset, we applied the SMOTE technique, which helped generate synthetic examples for underrepresented failure types. We noticed significant disparity among class distributions, so we balanced them using SMOTE before training the model. By incorporating SMOTE during preprocessing, we ensured the classifier received a more representative and fair dataset to learn from. The model is trained to distinguish between these categories by learning from patterns in the resource metrics and job characteristics [63].

Table 4: Failed Classes distribution

Event	Number of jobs	Class
EVICT	14756	0
FAIL	92678	1
LOST	59515	2
KILL	951	3

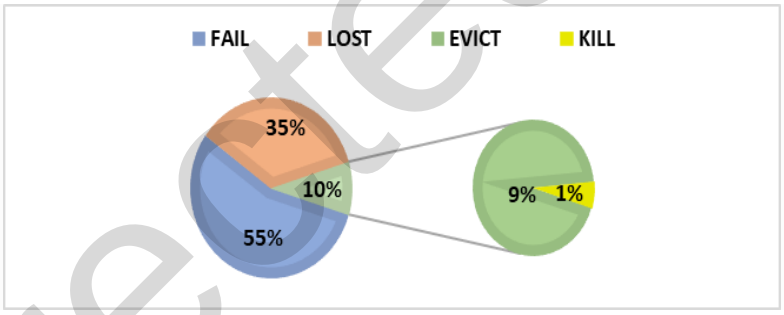

Fig. 5: presents the class distribution of failed jobs in the dataset.

5. EXPERIMENTS

The primary objective of this experimental phase is to assess the performance of the AFMF in predicting failure types and applying adaptive resource management. The RF classifier was trained on the scaled training data and evaluated using the test set. Accuracy served as the core evaluation metric, while additional metrics such as precision, recall, and F1-score were calculated to provide a deeper understanding of the model's classification performance. The confusion matrix offered a clear view of how well each failure type, Evict, Fail, Lost, and Kill, was identified. The effectiveness of the second phase of the framework was evaluated by measuring the number of jobs that were completed following the application of corrective actions. Visualizations were used to illustrate how many jobs were protected under each type of remedy action. Additionally, a comparison between the total number of submitted jobs and those successfully terminated highlighted the framework's ability to maintain operational continuity and reduce failure-related losses across the system.

5.1 platform

The implementation and execution of the proposed framework were conducted using the Kaggle platform, a cloud-based environment that offers flexible computing resources for data science and

machine learning tasks. Kaggle Notebooks provide an interactive interface like Jupyter Notebooks, enabling seamless integration with datasets either from Kaggle's public repository or through user uploads. These notebooks support programming languages such as Python and R, allowing for the development, training, and testing of both ML and DL models in an accessible online setting. In this study, the authors utilized Kaggle's standard resources, which include CPU-based computation, up to 57.6 GB of disk storage, and a maximum of 30 GB of RAM. These specifications were sufficient to support model training and evaluation. Each session also includes temporary local storage that facilitates the handling of datasets, intermediate results, and model outputs throughout the analysis. This setup ensured a smooth and efficient execution process without reliance on external infrastructure.

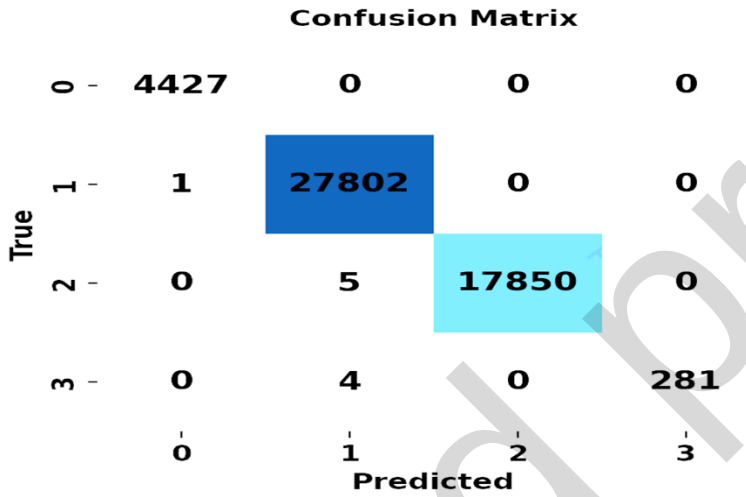

Fig. 6: RF Confusion Matrix

5.2 Evaluating the type of Failure Prediction model

The goal of this section is to evaluate the RF model, which predicts the type of failure. The possible failure outcomes are 'EVICT', 'FAIL', 'LOST', or 'KILL'. During the experimental phase, the RF classifier was configured with two essential parameters. The first, n_estimators, was set to 100, meaning the model constructed a total of one hundred DTs. This number enables the ensemble to draw predictions from a broad set of trees, which enhances generalization and lowers the likelihood of overfitting. Using multiple trees also contributes to greater model reliability, particularly when handling complex or noisy datasets. The second parameter, random_state, was set to 42 to ensure the reproducibility of results. This fixed value controls the randomness associated with tree construction and data sampling. As a result, the model yields consistent outcomes across different runs, which is especially valuable when performing repeated experiments or comparisons. The evaluation metric used shows that the RF model has performed exceptionally well, with most predictions being correct. Misclassifications are minimal, affecting only a few samples, and the overall accuracy is 99.98%.

Fig. 6 presents the confusion matrix for the classification results, where each row corresponds to the actual class and each column to the predicted class. For the EVICT class (labeled as 0), the model demonstrated perfect classification, correctly identifying all 4427 samples without any misclassifications. In the case of the FAIL class (class 1), a total of 27,802 instances were accurately predicted, while only 1 sample was incorrectly classified as EVICT. No samples from this class were misclassified as LOST or KILL. Moving to the LOST class (class 2), the model successfully identified 17,850 samples, with only 5 mistakenly labeled as FAIL and no errors involving the other classes. For the KILL class (class 3), 281 samples were correctly predicted, while 4 were incorrectly classified as FAIL. These results demonstrate a high level of accuracy and minimal confusion across class boundaries, indicating strong model performance in distinguishing between different types of failures.

Although the number of misclassifications is minimal, understanding their operational implications is crucial for effective cloud deployments. For example, classifying a KILL event as a FAIL may lead the system to initiate unnecessary fault-recovery procedures, which waste computing resources and cause delays, even though the termination was intentional. In contrast,

classifying a FAIL as a KILL could result in overlooking an unplanned outage, delaying corrective action, and potentially affecting service availability. Likewise, confusing LOST and FAIL cases can influence the prioritization of resource reallocation, since LOST jobs often require data recovery before rerunning. Although these errors account for less than 0.02% of total predictions, adopting targeted mitigation strategies such as confidence-based alerts or secondary verification for low-probability predictions could further limit their operational impact.

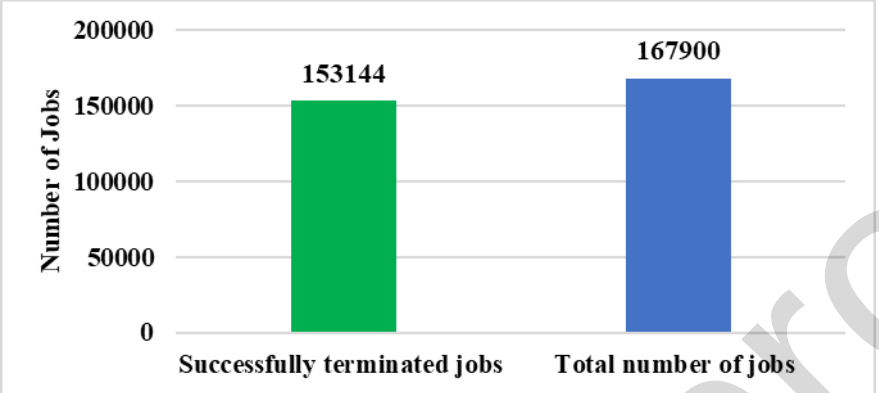

Fig. 7: Total Jobs VS Successfully Completed Jobs

5.3 Evaluation of dynamic remediation techniques for predicted job failures

This rule-based technique makes the system flexible and lightweight, ideal for large-scale cloud management where rapid decisions are crucial. It balances simplicity with effectiveness by applying structured, logic-based remedies under variable operating conditions. The evaluation of the remedy system demonstrates that out of 167900 total jobs, 153144 were completed following the application of targeted corrective actions, as illustrated in **Fig. 7**. These completed jobs were recovered through one of four distinct strategies: replication, migration, dynamic resource scaling, or rescheduling. Among them, replication contributed the highest raw number, accounting for 95006 recoveries. However, since replication involves deploying duplicate instances of the same job across two nodes, this figure was normalized by dividing it in half, resulting in 47503 uniquely protected jobs. Migration facilitated the completion of 32044 jobs by relocating them to more capable nodes. Dynamic resource scaling allowed 57283 jobs to adjust their CPU and RAM requirements based on node availability, while rescheduling helped recover 16314 low-priority tasks by postponing their execution until sufficient resources became available. Together, these tailored remedies formed a cohesive and context-aware framework for handling predicted failures. **Table 5** and **Fig. 8** present a detailed breakdown of the number of jobs recovered by each remedy action.

Table 5: Number of Protected jobs per Remedy Action

Corrective Action	Number of protected Jobs
Replication	95006
Migration	32044
Resource Scaling	57283
Rescheduling	16314

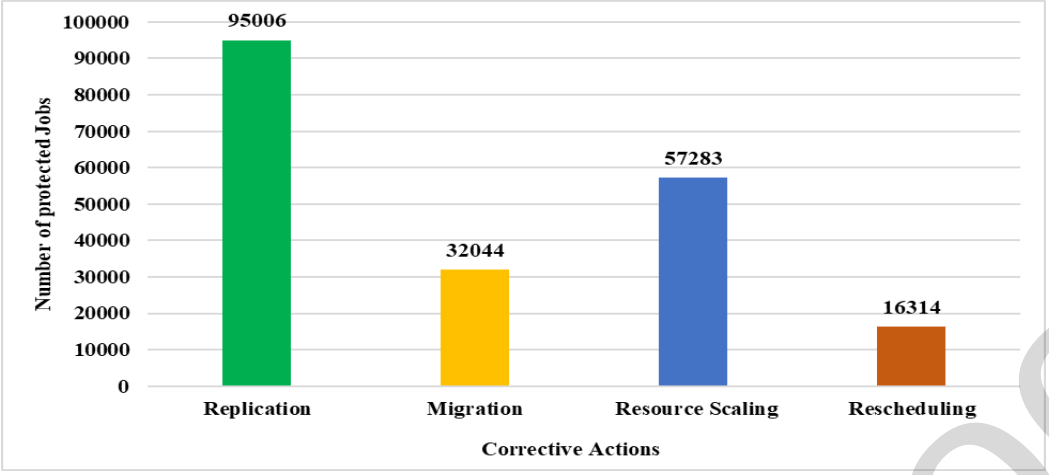

Fig. 8: Effectiveness of Corrective Actions

The overall effectiveness of the system was calculated at 91.21%, underscoring the strength of its adaptive decision-making and scalability. The results affirm the value of this heuristic-based system in enhancing service continuity and optimizing workload execution. By maintaining a high job completion rate and ensuring efficient use of cloud resources, the remedy system establishes itself as a practical and intelligent solution for managing large-scale, failure-prone environments. To place the 91.21% remediation success rate in perspective, we compared the proposed heuristic framework with a baseline redundancy-based method that replicates all predicted risky jobs without any prioritization. Using our dataset of 167,900 jobs, this baseline recovered 47,503 jobs (28.3% success rate after normalizing for duplicate instances) because it could not address failures caused by resource shortages or scheduling limitations. In comparison, our framework dynamically applies replication, migration, scaling, or rescheduling according to job priority and node status, achieving recovery for 153,144 jobs (91.21% success rate). This outcome reflects a 3.2× improvement over the baseline while minimizing unnecessary duplication and resource overhead.

Despite the strengths of the proposed framework, several limitations remain. The model depends heavily on the accuracy of predicted failure probabilities; any deviation in these estimates can lead to incorrect or inefficient remedy choices. It also lacks economic awareness, which may result in unnecessary resource consumption in environments where cost optimization is critical. Additionally, the framework does not consider energy efficiency, which is increasingly important in sustainable cloud operations. Without energy-aware decision-making, the system may overutilize high-power nodes regardless of workload demand.

CONCLUSIONS AND FUTURE WORK

This study presented a comprehensive framework for predicting and mitigating job failures in large-scale cloud environments using ML and heuristic-based techniques. The proposed solution centered on two essential components: an intelligent failure type classification model and an adaptive remedy system capable of responding dynamically to predicted failures in real time. Together, these components offer an integrated approach that improves fault tolerance, enhances resource efficiency, and supports operational stability across distributed systems. The classification model, built using the RF algorithm, achieved a remarkably high level of predictive performance, reaching an overall accuracy of 99.98 %. This result confirms the model's ability to accurately differentiate among distinct failure types. In parallel, the dynamic remedy system applied a heuristic strategy that selected corrective actions based on job priority, predicted failure probability, and current resource availability. By sorting jobs by evaluating node capacities in real time, the system assigned one of four predefined remedies: replication, migration, resource scaling, or rescheduling. This rule-based logic eliminated the need for computationally expensive optimization procedures and enabled swift, context-aware decisions even under heavy workloads.

The remedy system successfully addressed 91.21 % of the jobs predicted to fail. This result demonstrates a notable improvement compared to the performance outcomes reported in previous studies. The proposed framework effectively bridges the gap between early failure detection and proactive corrective action, reinforcing its applicability in real-world cloud scenarios. Future work may explore extending the framework with deep reinforcement learning to allow the system to

learn from past remediation outcomes and refine its decision-making over time. Additionally, incorporating cost-aware policies could improve resource allocation by factoring in operational expenses and economic trade-offs.

ACKNOWLEDGMENT

The authors would like to thank the National Telecommunication Regulatory Authority (NTRA), Egypt, for their support.

REFERENCES

[1] G. Zhou, W. Tian, R. Buyya, R. Xue, and L. Song, “Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions,” arXiv preprint arXiv:2105.04086v2, Nov. 2024.

[2] A. Elkaradawy, A. Elshenawy, and H. Harb, “Enhancing Cloud Job Failure Prediction with a Novel Multilayer Voting-Based Framework,” IEEE Access, early access, Jul. 30, 2025.

[3] D. Saxena, I. Gupta, A. K. Singh and C. N. Lee, “A Fault Tolerant Elastic Resource Management Framework Towards High Availability of Cloud Services”, IEEE Trans. on Network and Service Management, 2022.

[4] F. Haque, T. Islam, T. Shaila, R. Hasan, C. Caicedo “A Deep Dive into the Google Cluster Workload Traces: Analyzing the Application Failure Characteristics and User Behaviors” 10th Int. Conf. on Future Internet of Things and Cloud (FiCloud 2023). August 2023.

[5] B. Ray, A. Saha, S. Khatua, and S. Roy, “Proactive fault-tolerance technique to enhance the reliability of cloud service in cloud federation environment”, IEEE Trans. on Cloud Computing, vol. 8, no. 2, pp. 1-12, Jan. 2020.

[6] J. Dong, Y. Chen, B. Yao, X. Zhang, and N. Zeng, “A neural network boosting regression model based on XGBoost”, Applied Soft Compu-ting, vol. 125, pp. 109067, 2022.

[7] X. Chen, L. Yang, and C. Rong, “Resource Allocation with Workload-Time Windows for Cloud-Based Software Services: A Deep Reinforcement Learning Approach,” IEEE Trans. on Cloud Comput-ing, vol. 11, Apr. 2023.

[8] L. Liang, S. Meng, X. Qiu, and Y. Dai, “Improving failure tolerance in large-scale cloud computing systems”, IEEE Trans. on Reliability, vol. 68, no. 2, pp. 620-632, 2019.

[9] A. Priyadarshini, S. Kumar, S. Ranjan, and D. Nayak, “Enhancing Fault Tolerance and Load Balancing in Cloud Computing for improved e-healthcare Systems Performance”, IEEE Trans., 2nd Int. Conf. on Ambient Intelligence in Health Care (ICAIHC), Nov. 2023.

[10] S. Laha, M. Parhi, S. Pattnaik, and B. Pattanayak, “Issues Challenges & Techniques for Resource Provisioning in Computing Environment”, IEEE Xplore, 2020 2nd Int. Conf. on Applied Machine Learning (ICAML), vol. 2022, 2020.

[11] C. Spyridon and S. Sotiriadis, "An adaptive auto-scaling framework for cloud resource provisioning”, Future Generation Computer Systems, Vol. 148, Nov. 2023, PP. 173-183.

[12] S. Abderraziq, M. Hakem and B. Benmammar, “A Distributed Fault-Tolerant Algorithm for Load Balancing in Cloud Computing Environments”, E3S Web of Conferences, vol. 351, pp. 01012, 2022.

[13] C. Meyyappan and C. S. Ravichandran, “Performance analysis of fault-tolerance techniques towards solar fed cascaded multilevel inverter”, Int. Conf. on Computer Communication and Informatics (ICCCI), vol. 2021, Jan. 2021.

[14] Y. Xiao, L. Xiao, K. Wan, H. Yang, Y. Zhang, Y. Wu, and Y. Zhang, “Reinforcement learning based energy-efficient collaborative inference for mobile edge computing,” IEEE Trans. Commun., vol. 71, pp. 864–876, 2023.

[15] K. Vani and S. Sujatha, “A Machine Learning Framework for Job Failure Prediction in Cloud using Hyper Parameter Tuned MLP,” 2022 Second Int. Conf. on Advanced Technologies in Intelligent Control, Environment, Computing & Communication Engineering (ICATIECE), Dec. 16-17, 2022.

[16] M. Moni, M. Niloy, F. Juboraj, and A. Banik, ‘‘AI-based Cloud Failure Prevention Algorithm leveraging machine learning for enhanced reliability,’’ in Proc. 2023 Innovations Power Adv. Comput. Technol. (i-PACT), Malaysia, 2023.

[17] Y. Alahmad, T. Daradkeh, & A. Agarwal, ‘‘Proactive Failure-Aware Task Scheduling Framework for Cloud Computing,’’. IEEE Access, vol. 9, pp. 123-145, August. 2021.

[18] J. Zhang, Z. Ning, M. Waqas, H. Alasmary, S. Tu, and S. Chen, ‘‘Hybrid Edge-Cloud Collaborator Resource Scheduling Approach Based on Deep Reinforcement Learning and Multiobjective Optimization,’’ IEEE Transactions on Computers, vol. 73, no. 1, pp. 192–205, Jan. 2024.

[19] P. Upadhyay, K. K. Sharma, R. Dwivedi, and P. Jha, ‘‘A Statistical Machine Learning Approach to Optimize Workload in Cloud Data Centre,’’ in Proc. 7th Int. Conf. Comput. Methodologies Commun. (ICCMC), India, 2023.

[20] K. Vani and S. Sujatha, "Enhancing Fault Tolerance System in Cloud Environment Through Failure Prediction Using Ensemble Learning," 2024 2nd Int. Conf. on Intelligent Data Communication Technologies and Internet of Things (IDCIoT), Jan. 04-06, 2024.

[21] J. Shetty, R. Sajjan, and G. Shobha, ‘‘Task resource usage analysis and failure prediction in cloud,’’ in Proc. 9th Int. Conf. Cloud Comput., Data Sci. Eng. (Confluence), Jan. 2019, pp. 342–348.

[22] A. Adel and A. El Mougy, ‘‘Cloud Computing Predictive Resource Management Framework Using Hidden Markov Model,’’ in Proc. 5th Conf. Cloud Internet Things (CIoT), Marrakech, Morocco, Mar. 2022, pp. 1–6, IEEE.

[23] S. Tuli, G. Casale, and N. R. Jennings, ‘‘CILP: Co-Simulation-Based Imitation Learner for Dynamic Resource Provisioning in Cloud Computing Environments,’’ IEEE Trans. Netw. Service Manag., vol. 20, no. 4, Dec. 2023.

[24] S. K. Grewal and N. Mangla, ‘‘Deadline and Cost Optimization Based Task Scheduling (DCOTS) in Cloud Computing Environment,’’ IEEE Trans. in Proc. 2023 4th Int. Conf. Intell. Eng. Manag. (ICIEM), London, U.K.,2023.

[25] V. Salanke and V. Kowshik,"Task Failure Prediction and Migration in Cloud Environment," 2024 1st Int. Conf. on Communications and Computer Science (InCCCS), May 22, 2024.

[26] S. Jomah and S. Aji, "Google Cloud Trace: Characterization of Terminated Jobs," 2024 2nd International Conference on Advancement in Computation & Computer Technologies (InCACCT), May 02-03, 2024, doi: 10.1109/InCACCT61598.2024.10551146.

[27] A. Tuns and A. Spataru, "Cloud Service Failure Prediction on Google’s Borg Cluster Traces Using Traditional Machine Learning," 2023 25th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Sep. 11-14, 2023, doi: 10.1109/SYNASC61333.2023.00029.

[28] C. Liu, L. Dai, Y. Lai, G. Lai, and W. Mao, ‘‘Failure prediction of tasks in the cloud at an earlier stage: A solution based on domain information mining,’’ Computing, vol. 102, no. 9, pp. 2001–2023, Sep. 2020.

[29] J. Gao, H. Wang, and H. Shen, ‘‘Task failure prediction in cloud data centers using DL,’’ IEEE Trans. Services Computing., early access, May 11, 2020, doi: 10.1109/TSC.2020.2993728.

[30] A. Rosa, L. Y. Chen, and W. Binder, ‘‘Failure analysis and prediction for big-data systems,’’ IEEE Trans. Services Computing., vol. 10, no. 6, pp. 984–998, Nov. 2017.

[31] T. Islam and D. Manivannan, ‘‘Predicting application failure in cloud: A ML approach,’’ in Proc. IEEE Int. Conf. Cognit. Comput. (ICCC), Jun. 2017, pp. 24–31.

[32] C. Liu, J. Han, Y. Shang, C. Liu, B. Cheng, and J. Chen, ‘‘Predicting of job failure in compute cloud based on online extreme learning machine: A comparative study,’’ IEEE Access, vol. 5, pp. 9359–9368, 2017.

[33] X. Chen, C.-D. Lu, and K. Pattabiraman, ‘‘Failure prediction of jobs in compute clouds: A Google cluster case study,’’ in Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops, Nov. 2014, pp. 341–346.

[34] F. Bashir and F. Z. Khan, “Analysis of Job Failure Prediction in a Cloud Environment by Applying Machine Learning Techniques,” Int. Journal of Information Science and Technology, vol. 4, no. 4, pp. 1-10, Oct. 2022.

[35] B. Mohammed, I. Awan, H. Ugail, and M. Younas “Failure prediction using ML in a virtualized HPC system and application,” Cluster Computer, vol. 22, pp. 471–485, 2019.

[36] M. S. Jassas and Q. H. Mahmoud, “Analysis of Job Failure and Prediction Model for Cloud Computing Using Machine Learning” Sensors (Basel). 2022.

[37] S. Divesh and Y. Samarawickrama, “Improving fault tolerance of a task in cloud using ensemble approach” Master’s thesis, Dublin, National College of Ireland. Vashi (2023).

[38] M. M. Abualhaj, M. M. Al-Tahrawi, and S. N. Al-Khatib, “Performance evaluation of VoIP systems in cloud computing.” J. Eng. Sci. Technol, vol.14, 1398-1405, 2019.

[39] M. Kolhar, M. M. Abu-Alhaj, and S. M. Abd El-atty, “Cloud data auditing techniques with a focus on privacy and security,” IEEE Security & Privacy, vol. 15, no. 1, pp. 42–51, Jan.–Feb. 2017.

[41] A. Marahatta, Q. Xin, C. Chi, F. Zhang, and Z. Liu, “PEFS: AI-driven prediction-based energy-aware fault-tolerant scheduling scheme for cloud data center,” IEEE Trans. Sustain. Comput., vol. 5, no. 3, pp. 1–14, Aug. 2020.

[42] R. R. Swain, D. S. K. Nayak, and T. Swarnkar, “A comparative analysis of ML models for colon cancer classification,” in 2023 Int. Conf. Adv. Power Signal Inf. Technol. (APSIT), pp. 1–5, Jun. 2023.

[43] M. S. Al-Asaly, M. A. Bencherif, A. Alsanad, and M. M. Hassan, “A deep learning-based resource usage prediction model for resource provisioning in an autonomic cloud computing environment” Neural Comput. Appl., vol. 34, no. 13, pp. 10211–10228, 2022.

[44] M. S. Kumar, A. Choudhary, I. Gupta, and P. K. Jana, “An efficient resource provisioning algorithm for workflow execution in cloud platform,” Cluster Comput., vol. 25, no. 6, pp. 4233–4255, Dec. 2022.

[45] M. Sohani and S. C. Jain, “A predictive priority-based dynamic resource provisioning scheme with load balancing in heterogeneous cloud computing,” IEEE Access, vol. 9, pp. 62653–62664, Dec. 2021.

[46] Google, "Google 2019 Cluster Sample," Kaggle, Accessed: Oct. 5, 2024. [Online]. Available: https://www.kaggle.com/datasets/derrick- mwiti/google-2019-cluster-sample

[47] S. Ouhame, Y. Hadi and A. Ullah, “An efficient forecasting approach for resource utilization in cloud data center using CNN-LSTM model”, Neural Comput. Appl., vol. 33, pp. 10043-10055, Mar. 2021.

[48] X. Gao, R. Liu, A. Kaushik, and H. Zhang, “Dynamic resource allocation for virtual network function placement in satellite edge clouds", IEEE Trans. on Network Science and Engineering, vol. 9, no. 4, pp. 2252-2265, 2022.

[49] M. Tirmazi, A. Barker, N. Deng, M. Haque, Z. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes, “Borg: the next generation,” in Proc. 15th Eur. Conf. Comput. Syst. (EuroSys ’20), Heraklion, Greece, Apr. 2020.

[50] G. Amvrosiadis, J. W. Park, R. G. Ganger, A. G. Gibson, E. Baseman, and N. DeBardeleben, ‘‘On the diversity of cluster workloads and its impact on research results,’’ in Proc. USENIX Annu. Tech. Conf., 2018, pp. 533–546.

[51] M. Soualhia, F. Khomh, and S. Tahar, ‘‘Predicting scheduling failures in the cloud: A case study with Google clusters and Hadoop on Amazon EMR,’’ in Proc. IEEE 17th Int. Conf. High Perform. Computer. Commun., Aug. 2015

[52] T. Islam and D. Manivannan, ‘‘FaCS: Toward a fault-tolerant cloud scheduler leveraging long short-term memory network,’’ in Proc. 6th IEEE Int. Conf. Cyber Secur. Cloud Comput. (CSCloud), 5th IEEE Int. Conf. Edge Comput. Scalable Cloud (EdgeCom), Jun. 2019, pp. 1–6.

[53] Z. Li, L. Liu, and D. Kong, “Virtual Machine Failure Prediction Method Based on AdaBoost-Hidden Markov Model,” 2019 Int. Conf. on Intelligent Transportation, Big Data & Smart City (ICITBS), 2019, pp. 700-703.

[54] D. S. K. Nayak, S. P. Routray, S. Sahooo, S. K. Sahoo and T. Swarnkar, “A Comparative Study using Next Generation Sequencing Data and ML Approach for Crohn's Disease (CD) Identification”, 2022 Int. Conf. on ML Computer Systems and Security (MLCSS), pp. 17-21, August 2022.

[55] D. Yaso Omkari, K. Shaik “An Integrated Two-Layered Voting (TLV) Framework for Coronary Artery Disease Prediction Using ML Classifiers” IEEE Access, vol. 12, April 2024.

[56] X. Xu, R. Mo, F. Dai, W. Lin, S. Wan, and W. Dou, “Dynamic resource provisioning with fault tolerance for data-intensive meteorological workflows in cloud,” IEEE Trans. Ind. Informat., vol. 16, no. 9, pp. 61726181, Sep. 2020.

[57] P. Kumari and K. Parmeet, “A survey of fault tolerance in cloud computing”, Journal of King Saud Univ., Comput. Inf. Sci., vol. 33, no. 10, pp. 1159-1176, 2021.

[58] P. Padmakumari and A. Umamakeswari, ‘‘Task failure prediction using combine bagging ensemble (CBE) classification in cloud workflow,’’ Wireless Pers. Commun., vol. 107, no. 1, pp. 23–40, Jul. 2019.

[59] A. Rosa, L. Y. Chen, and W. Binder, ‘‘Predicting and mitigating jobs failures in big data clusters,’’ in Proc. 15th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput., May 2015, pp. 221–230.

[60] K. Jena, P. K. Das, and M. R. Kabat, “Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment,” J. King Saud Univ., Comput. Inf. Sci., vol. 34, no. 6, pp. 2332-2342, Jun. 2022.

[61] H. Sun, H. Yu, G. Fan, and L. Chen, “QoS-aware task placement with fault-tolerance in the edge-cloud,” IEEE Access, vol. 8, pp. 77987-78003,2020.

[62] M. Ozer, S. Varlioglu, and B. Gonen “Cloud Incident Response: Challenges and Opportunities” 2020 Int. Conf. on Computational Science and Computational Intelligence (CSCI).

[63] I. Kim, W. Wang, and M. Humphrey, “Forecasting Cloud Application Workloads with CloudInsight for Predictive Resource Management,” IEEE Trans. on Cloud Computing, vol. 8, no. 4, pp. 1-13, May 2020.