Towards Resilient Distributed Computing: A Self-Healing Fault Tolerance Framework Using Hybrid AI Models

Mohd Haroon
Department of Computer Science and Engineering
Integral University, India
mharoon@iul.ac.in

Zeeshan Ali Siddiqui*
Department of Computer Science and Engineering
University of Lucknow, India
siddiqui_zeeshan@lkouniv.ac.in

Mohammad Husain
Faculty of Computer and Information System
Islamic University of Madinah, Saudi Arabia
dr.husain@iu.edu.sa

Arshad Ali*
Faculty of Computer and Information System
Islamic University of Madinah, Saudi Arabia
a.ali@iu.edu.sa

Abstract: *In today's distributed computing paradigm, achieving resilience against unpredictable faults is still a challenging task. Several state-of-the-art fault tolerance algorithms are static and rule-based, lacking the ability to foresee future events. This work investigates a novel self-healing framework that employs Long Short-Term Memory (LSTM) networks for fault prediction and a Q-learning-based Reinforcement Learning (RL) methodology for adaptive fault recovery. In this paper, we model the recovery decision process as a Markov Decision Process (MDP) whereby the framework learns the best actions to take autonomously from the system state transitions, as well as any feedback received. A high-fidelity simulation environment was utilized to inject realistic errors so that we could rigorously evaluate the model's performance. The comparative experimental results demonstrate significant improvements in accuracy (96.3%) and Mean Time to Recovery (MTTR) (34 sec) as well as improved availability (97.2%) compared to baseline and state-of-the-art models. This work highlights the opportunity for hybrid Artificial Intelligence (AI) models to promote resilience in cloud-edge distributed systems. The modular design of this framework also presents learning opportunities for deploying as a lesson plan in courses that focus on both distributed systems and intelligent automation in computer science curricula.*

Keywords: *Fault tolerance, LSTM, Q-learning, self-healing systems, distributed computing, reinforcement learning, system resilience, MTTR.*

Received August 16,2025; accepted December 08,2025
https://doi.org/10.34028/iajit/23/4/9

1. Introduction

The increasing prevalence and utilization of distributed systems in a variety of critical use cases (including cloud computing and the Internet of Things(IoT) for Industrial purposes) has created an urgent demand for an effective fault tolerance approach to ensure continued service availability for organizations [34, 35, 40]. As modern infrastructures are growing in complexity and heterogeneity, traditional rule-based recovery strategies are becoming increasingly inadequate due to their static behavior and limited opportunity to adapt to dynamic fault situations. Intelligent fault tolerance systems, which leverage Machine Learning (ML) and Reinforcement Learning (RL), provide an optimistic framework to overcome that challenge. Recent research has investigated the employment of Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) deep learning models for fault prediction, primarily based on their temporal learning capabilities. Q-learning and deep q-networks have been successful in developing constantly learning autonomous recovery decisions by modeling the system responses to fault events, representing a long-term future probabilistic model of the response using the Markov Decision Process (MDP) framework [27, 30, 36]. However, most existing solutions make use of either predictive or reactive intelligence by themselves and therefore are limited in recovery time and scalability. In order to address these limitations, this paper presents a hybrid ai-based fault tolerance framework that combines the long-term data inference ability of LSTM with the online action selection power of Q-learning. Rather than simply predicting future faults as done in previous approaches, our new framework uses LSTM for a model of a fault's location and the Q-learning agent for a high-level model of a fault's cancellation action with respect to the long-term performance of the system. We benchmark our system within a high-fidelity simulation that incorporates realistic workload patterns obtained from Google and Azure traces by Reiss *et al*. [28] and compare it to other state-of-the-art models,

including GRU+Q-learning by Heda and Sahare [19] and Deep Q-Network+ Long Short-Term Memory (LSTM+DQN) by Zhang *et al*. [47] and Choe *et al*. [12]. The research has advanced us toward self-healing systems in distributed environments, but they also offer a pathway for computer science students to explore AI-based resilience approaches. Predictive analytics and reinforcement learning-based intelligent and scalable fault management systems are being developed from the substance provided by these architectures. This intelligent and scalable fault management system is implemented through the use of a feedback-aware loop.

2. Literature Review

Fault tolerance in distributed systems has been studied by researchers since early years. One of the reasons researchers have focused on fault tolerance in distributed systems is that resources are distributed, and services must always be available according to Coulouris *et al*. [14]. People have had to employ replication, checkpoint-restart, and heartbeat in some way to restore a node or service that failed. These procedures support basic fault tolerance but potentially utilize unnecessary resources and are only suited for static edge and cloud systems proposed by Xiao *et al*. [42] and Amin *et al*. [5]. AI has revolutionized the process of problem-solving. We have used supervised learning methods and ML models to detect problems and guess when they will be found based on system metrics and logs suggested by Gogineni [17]. In a cloud environment, Shah and Patel [33] used random forest and support vector machines to correctly identify different types of faults. Zhang *et al*. [46] showed that LSTM networks could model temporal dependencies in system logs to identify hidden faults before they happen. That said, a supervised model can't learn about a new kind of fault or how to fix it on its own. Researchers have started to investigate RL [41] to deal with adaptability, in particular with regard to maintaining service availability through fault recovery; RL agents learn fault recovery action sequences through interaction with the environment with feedback provided through rewards. Yang *et al*. [43] described a Q-learning-based recovery model that selected mitigation actions while keeping the recovery goals based on performance metrics for the containerized workload being considered while showing minimizing the degradation downtime and performance instability. Abdel Aziz *et al*. [1] also utilized an RL approach, with a (DQN) to deal with service failures in an edge computing scenario; however, training time and convergence caused practical limitations on use. Hybrid AI models are emerging as a way of combining multiple learning paradigms to improve fault detection and recovery. A recent example is the work of Yang *et al*. [44] combining an ensemble of supervised classifiers and deep RL to create and implement a self-healing service network. When compared to just using supervised or RL-based models, this architecture did a better job of meeting Service Level Agreements (SLAs) and getting things back to normal faster. Even though hybrid learning approaches seem to be promising, there haven't been many research projects that have looked at this in a complete way to deal with fault prediction and recovery in a single, continuous, modular framework that is ready for real-time and the problems of distributed computing environments. At the same time, research on fault tolerance has moved to educational uses, such as creating smart fault injection tools that mimic learning environments to teach system dependability and AI integration as used by Berman *et al*. [9]. These systems can help computer science students learn by doing by combining theory with real-world experience in resilient distributed systems and machine learning deployment. In conclusion, there are a LOT of AI-based fault tolerance frameworks, but not a lot of research has been done on a full, self-healing framework that includes prediction, adaptive recovery, and teaching value for fault mitigation used by Kambala [20] and Pasham [25]. This study builds on these advances by using a hybrid AI-based architecture that uses supervised learning to find faults early and reinforcement learning to adapt recovery. This work makes both resilient computing and education better.

2.1. Fault Prediction Models

Defect prediction in supervised learning is usually approached as a classification problem, whereby the state of the system at a point in time is represented as a feature vector by Sen *et al*. [32]. Each feature vector is associated with a label that indicates when a fault occurred and the type of fault (if relevant). One of the most common machine learning approaches for this problem is RF; it has been recognized for its ensemble methodology and high level of accuracy [29]. RF builds predictions based on a large number of individual decision trees built against unique subsets of the data. Thus, the final probability of a fault is based on the average of all of the trees. In other words, it integrates different decision patterns and decreases over fitting. At time *t*, the state of the system is expressed as a feature vector *xt*, and the fault label *yt* is from a set of classes. The fault probability trading off RF is:

$$P(Yt = k \mid Xt) = (1/T) * \Sigma\, ht(Xt) \quad (1)$$

That is i=1 to T

where *ht* is the prediction of the *i-th* tree from the ensemble size *T*. At time *t*, the state of the system represented by a feature vector defines a classification of fault label. The Random Forest algorithm computes fault probability as the average prediction from all the decision trees in the ensemble.

Unlike RF, (LSTMs networks) are a type of deep learning that run on sequential data, or data that comes

in order. LSTMs have a hidden state that changes over time. LSTMs can learn patterns and relationships in a series of vectors over time as elements are processed. At each time step, the model creates a new hidden state by combining the current input and the last hidden state. Then, the model representation goes through a fully-connected layer followed by a sigmoid activation function to create a probability of fault. LSTMs have many useful features such as the ability to remember and utilize information from the past. This ability can be paramount in systems where past performance has some effect on faults. RF and LSTMs are powerful and useful approaches for classifying a fault prior to when it occurs but have different strengths. Random Forests uses ensemble-learning of static or fixed data, while LSTMs use sequential modeling to capture changes occurring over time [26].

In LSTM networks, the hidden state is updated using both the current input as well as the previous hidden state. This would update the hidden state at time t in the LSTM networks. The updated hidden state is informed by the LSTM's internal parameters that learn the temporal structure of the sequential input. In regard to fault prediction, the updated hidden state reflects the system's behavior including the most recent and previous states. A dense layer with a sigmoid activation function will generate a probability score and then produce the likelihood of the fault.

2.2. Fault Scoring and Anomaly Detection

Autoencoders are a very effective method for conducting fault scoring and, to an extent, anomaly detection, which enables us to identify system faults without needing labeled data [24]. Autoencoders learn to reconstruct normal system behavior through how they learn to compress input feature vectors [6, 7]. During inference, the autoencoder attempts to reconstruct incoming data, and then it looks at the difference between its reconstruction and the input. This difference is the reconstruction error, which is used as the anomaly score for that time stamp as suggested by Torabi *et al.* [38]. If the reconstruction error is large, it could mean the input is very different from what is normal. The system will also require us to check for faults by setting a threshold. If the anomaly score is above the threshold, it is labeled a fault; if it is below the threshold, it is considered 'normal'. This is particularly useful when faults are rare or are not well defined. It allows the model to learn and find that aberrant behavior simply by looking for changes from the learned normal behavior, which is highly flexible and efficient in terms of data.

Autoencoders compute a reconstruction-based anomaly score as:

$$S(xt) = (xt - \hat{x}t)^2 \quad (2)$$

With a threshold τ:

Fault=1 *if* $S(x_t) > \tau$*; otherwise,* Fault=0

2.3. Reinforcement Learning for Fault Recovery

RL models the system as an MDP, where each state shows how the system is doing, actions show possible recovery steps, and rewards show how well those actions worked. This makes it a flexible way to recover from faults. The objective is to identify the best policy that permits the system to improve while keeping both long-term risks and long-term costs low. Q-learning is a common RL technique that updates the value of a state-action pair, also referred to as the Q-value, based on the reward received and the rewards perceived to be accrued in the future [16]. The update rule updates the Q-value based on the reward perceived and the maximum future expected Q-value of the next state by using a learning rate and discount factor. This enables the system to slowly optimize actions over time to achieve successful recovery. Q-learning is suited to real-world fault recovery tasks because it uses experience-based learning and does not require a model of the environment [13]. RL is particularly appropriate for real-world fault recovery applications because it is based on learning from experience and does not require knowledge of the environment.

2.4. System Reliability and Downtime Metrics

To assess the efficiency and dependability of critical infrastructure, one needs to assess system reliability and downtime statistics [15]. There are many ways to evaluate system reliability and operational performance. One of the most critical metrics is system availability, derived from system uptime. This metric is calculated using the ratio of Mean Time to Failure (MTTF) divided by MTTF+ Mean Time to Repair (MTTR). Therefore, improving either the MTTF or decreasing the MTTR will improve system availability. Availability denotes how much time a system is running, where lower MTTRs become critical for maintaining multiple functions and being operational for the longest duration. Expected Downtime (ED) per month is another area that pertains to the effect of availability on operations. This simply tells you how long the system would be expected to be down during some period of time. A lower ED is preferable and can help organizations properly plan for downtime while maximizing operational uptime. To calculate ED, you take the unavailability (1-availability) and multiply it by the total time in the month. The use of these metrics can assist organizations in identifying how effective they are at establishing fault tolerance and directing better planning of maintenance and resource allocation.

System availability A is calculated using:

$$A = MTTF / (MTTF + MTTR) \quad (3)$$

(ED)per month is:

$$ED = (1 - A) \times Total\ time \quad (4)$$

2.5. System State Representation

Let the system state at time *t* be represented as:

$$Xt = [CPUt, Memoryt, Diskt, Latencyt, Error\ Ratet, \ldots] \in \mathbb{R}^n \quad (5)$$

The telemetry information present in this vector reflects important features of a distributed computer network.

To have a foundation for monitoring, troubleshooting, and performing predictive maintenance for distributed information technology infrastructures, it is necessary to have a complete understanding of the state of the distributed system at any point in time (*t*) [4]. Therefore, all telemetry data needed to compute the state of that distributed system must be included in the vector (*xt*) where *xt* is the set of required telemetry data points at time *t*. These telemetry data points will include all of the telemetry metrics such as Central Processing Unit (CPU) utilization, memory utilization, disk I/O activity, network latency, error rates etc. All telemetry metrics are collected from numerous independent nodes throughout the distributed infrastructure and combined into the n-dimensional vector $xt \in Rn$, where *n* is the total number of different telemetry metrics (as previously noted). Additionally, the value to the engineer is having this structured representation of telemetry metrics, since machine learning uses time and error patterns in the data to generate accurate error predictions, diagnose errors and assist in predicting when a failure might occur in the Information Technology (IT) infrastructure. Furthermore, data conveys an all-encompassing picture of operational health by permitting the fusion of multiple concurrent behaviors of the system alongside each other into one vector. Such data representations are useful not only for reactive diagnostic processes, but they also form the basis for many proactive behaviors such as optimizing resource utilization, self-healing, and intelligent orchestration in complex, real-time computing system as suggested by Syed and Anazagasty [37] and Sekar and Aquilanz [31].

2.6. Fault Prediction Using LSTM

LSTM networks are suitable for predicting faults due to their ability to model how data evolves through time [48]. The approach processes the system state vector at time *t* and the previous hidden state to create a new hidden representation *Ht* from the learnable weights θ. The hidden state contains information about the current discrete state of the system and cumulative trends from prior states, which can reveal additional details about potentially emergent faults. The predicted fault probability, Y^t, is produced when the hidden representation, H^t, is placed through an output layer with a sigmoid or softmax activation using weights W_o and bias b_o. The output has a value between 0and1 indicating the likelihood of a fault occurring at this state in time. The model is trained to minimize the binary cross-entropy loss which punishes faults based on the true labels against the output probabilities; this allows the LSTM to differentiate between different time-dependent features, giving it an edge in identifying faults within dynamic systems in an online approach.

The temporal behavior of system states is modeled using LSTM:

$$Ht = LSTM(Xt, H\{t-1\}; \theta) \quad (6)$$

$$Y^t = \sigma(Wo * Ht + bo) \quad (7)$$

Where:$Y\hat{}t \in [0,1]$:

Predicted probability of fault,σ: sigmoid or softmax function,W_o, b_o:

output weights and bias Loss function (binary cross-entropy):

$$LSTM = -\sum [Yt\ log(Yt) + (1 - Yt)\ log(1 - Y\hat{}t)] \quad (8)$$

2.7. Fault Score and Risk Level

A fault risk score as the output of the prediction model:

$$Rt = Y^t \times \omega impact \quad (9)$$

where ωimpact is a weight based on node criticality.

Fault scoring and risk assessment provide a numeric value to potential impact, which helps system managers make good decisions [45]. A machine-machine learning or deep learning model provides us with the predicted fault probability Y^t, which we then use to find the fault risk score for time t, which we will maintain is called Rt. We use *a* impact weight Wimpact to scale this probability to account for how important the affected node or subsystem is to the overall architecture. Nodes that provide significant value by supporting key services or operations will have more significant weights assigned them. Therefore, if a fault is less likely to occur but is essential to the success of the business, we should ensure that the very small probabilities trigger the appropriate alerts and mitigation strategies. Risk Type (*Rt*) utilizes likelihood and potential impact, which together create a much more realistic risk assessment than just considering the probability of occurrence alone. This approach creates a prioritized listing of maintenance, remediation, and recovery activities; allowing an organization to conduct a planned resource allocation to reduce the downtime associated with resource-critical areas while promoting operational resiliency through being prepared for natural or mission-critical events.

2.8. Reinforcement Learning-Based Recovery

To define the environment using the MDP model:
state space (*S*): A function of the state of the environment. Action space (A): a function of achievable actions. Reward function (R): a function that provides a performance-based measure of the expected outcome of an action taken in the environment. Discount factor (q): the relative importance of expected future rewards compared to immediate rewards. The recovery process

modeled using RL-based methods is a method of recovery that allows a more flexible and independent approach to getting back on your feet following a failure-based event in a system. The recovery process is modeled using MDP, where the state space *S* represents the system's possible conditions, while the action space A represents all possible recovery options available (for example, restart service, reallocate resources, and initiate failover processes). The reward function *R* measures the effectiveness of actions executed under the reward function *R* and motivates recovery plans that minimize downtime and return the system to a stable operating condition as soon as possible. The discount factor γ represents the importance of the future rewards relative to the immediate reward, allowing the user to reconcile between immediate fixes and long-term maintenance and health of the system. People often use Q-learning, a model-free RL method, to find the best way to recover [10]. It changes the Q-value of a state-action pair by adding the immediate reward to the maximum expected future reward, which is then multiplied by the learning rate α. This process of repeating itself over time helps the agent figure out which actions are most useful in different types of faults. Q-learning is great for environments that are complex and always changing because it doesn't need to know how the system works ahead of time. RL-based recovery makes systems more reliable by learning from interactions all the time. It also makes them more resilient and speeds up the time it takes to fix them.

2.9. System Resilience Metrics

Capacity indicators provide a numerical measure of the reliability and strength of critical infrastructure components [11]. An average time to failure MTTF indicates how much time it is expected that a particular system will be operational before it fails; conversely, MTTR represents the amount of down time to recover the system after a failure. The ratio of MTTF to the total of MTTF Including MTTR equals the available time. Therefore, the higher the availability, the higher the degree of time that a system will be operational with maximum reliability and the higher the percentage of time will be up and operational. swifter recovery. To assess how availability impacts operations, we can calculate the ED over a specified time duration T by multiplying the unavailability (1-A) by T. This metric allows organizations to assess outages frequency, develop maintenance schedules, and prioritize where to invest in fault tolerance methods. All of this results in improved operational continuity and happier users. ED over time *T*:

$$ED = (1 - A) \times T) \quad (10)$$

2.10. Composite Optimization Objective

The goal is to minimize fault impact and maximize availability:

$$min\{\theta, \pi\}[LSTM + \lambda \times E\pi[\sum(1 - At)]] \quad (11)$$

Where: θ: LSTM parameters, π: *RL* policy, and λ: trade-off parameter between prediction and recovery.

A combined optimization objective allows for integrating fault prediction and fault recovery into a single framework that optimally solves for both problems, while simultaneously improving the accuracy of fault detection and resilience of the overall system. In this case θ refers to the parameters of the (LSTM) network that is used to predict faults and π refers to the RL policy, which is used to identify the optimal actions taken to recover. The primary goal of this reward function is to minimize the loss of LSTM predictions, thereby providing an accurate identification of faults while also minimizing the overall system's expected unavailability during the recovery process [49]. The 'unavailability' term is the expected value of policy π of total availability shortfall (1-*At*) over time. The trade-off parameter λ indicates how well the predictions work with relation to how well recovery works. The mechanism itself ensures that faults are found quickly and rectified even faster by jointly optimizing these components which inherently reduces downtime, improves reliability, and maintains overall operational continuity in changing environments.

3. System Architecture Model

Resilient distributed computing system architecture is a tuple, *S= (M, F, R, H, K)*, as seen in Figures 1 and 2. Each component is responsible for a splice to make the system resilient. The Monitoring module (M) receives and collects telemetry from all the distributed nodes. The Fault prediction engine *F* uses supervised learning to process this telemetry data and make predictions when the system will fail. The Recovery agent (R) uses reinforcement learning to explore and select the best recovery actions. The Healer engine (H) implements an actuator layer to recover the system to a stable state. The Knowledge base (K) collects, stores, and retrieves historical data, models, and policies to facilitate flexible and intelligent human decision making.

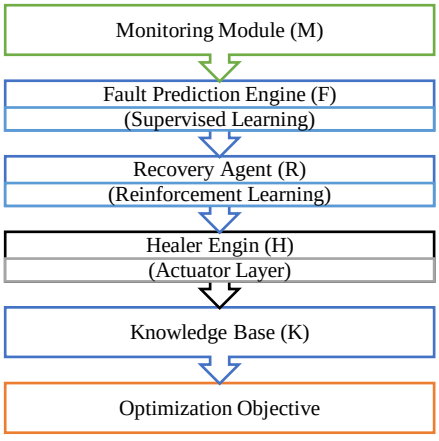


Figure 1. System Architecture Model.

3.1. Monitoring Module (M)

The monitoring module (M) is the part of the system that receives information from distributed nodes in real-time and continuously monitors the operational status of all nodes in the system. When the monitoring module receives its first reading (X) at a certain time (t), it will send X back to you (within the range of R) and provide you with one or more key performance and/or health metrics for each node monitored by M, such as CPU load (utilization), memory consumed (usage), I/O activity (read/write), latency (data transfer delays due to network conditions), and failure rates or occurrences (total number of errors). This process integrates all the node-level measurements into one unifying telemetry vector (X) that is available for use in arriving at corrective action, in identifying anomalies or problems, and for predicting future failures.

3.2. Fault Prediction Engine (F)

Maps telemetry vector to a fault risk score using LSTM:

$$Y^t = F(x\{t-k:t\}) = \sigma(Wo \cdot LSTM(x\{t-k:t\}) + bo) \quad (12)$$

Where: $Y\hat{} \in [0, 1]$:
fault probability, $Xt-k:t\}$:
historical data window, and σ: activation function.

The Fault prediction engine F turns a stream of telemetry data into a fault probability score, which makes it possible to find potential failures early.

It takes a historical data window $x\{t-k:t\}$ that shows recent system states and runs it through an LSTM network to learn about patterns and temporal dependencies that show signs of faults. After the LSTM's output goes through a fully connected layer with weights Wo and bias bo, it goes through an activation function σ, like sigmoid. The value $Y\hat{}t$ that comes out is between 0and1 and shows the system's predicted fault probability at time *t*.

Given a historical data window

$X\{t\text{-}k:t\} = (x\{t\text{-}k\}, x\{t\text{-}k+1\}, \ldots, xt$

The LSTM fault prediction model operates as follows:

a) LSTM cell equations

$$i\_\tau = \sigma(W\_i\, x\_\tau + U\_i\, h\_\{\tau-1\} + b\_i) \quad (13)$$

$$f\_\tau = \sigma(W\_f\, x\_\tau + U\_f\, h\_\{\tau-1\} + b\_f) \quad (14)$$

$$o\_\tau = \sigma(W\_o\, x\_\tau + U\_o\, h\_\{\tau-1\} + b\_o) \quad (15)$$

$$\hat{g}\_\tau = tanh(W\_g\, x\_\tau + U\_g\, h\_\{\tau-1\} + b\_g) \quad (16)$$

$$c\_\tau = f\_\tau \odot c\_\{\tau-1\} + i\_\tau \odot \hat{g}\_\tau \quad (17)$$

$$h\_\tau = o\_\tau \odot tanh(c\_\tau) \quad (18)$$

b) Fully connected output layer

$$z\hat{}t = V\, h\_t + b\_fc \quad (19)$$

$$\hat{Y}\hat{}t = \sigma(z\hat{}t) \quad (20)$$

c) Training loss (binary cross-entropy)

$$L = -\frac{1}{N}\Sigma\left[y \log(\hat{Y}) + (1-y)\log(1-\hat{Y})\right] \quad (21)$$

d) Decision rule

Fault Alarm= 1, if $\hat{Y}\hat{}t \geq \tau$

0, otherwise

3.3. Recovery Agent (R)

Given a fault prediction and system state s_t, select optimal action:

$$At = R(St) = argmax\, A\, Q(St, a) \quad (22)$$

The recovery agent (*R*) decides what the best corrective action is based on the current state of the system (*St*) and the predicted risk of a fault. It picks the action using reinforcement learning, more specifically Q-learning. *At* that point, the expected utility *Q* (*St*, *a*) is at its highest. We keep changing the Q-values, and the learning rate α controls how fast they change. The discount factor γ makes sure that rewards are fair now and in the future. The reward R_t shows how well the action you chose worked to get the system back to normal.

The recovery agent learns rules that make the system less likely to crash and more stable over time.

3.4. Healer Engine (H)

Recovery agents instruct healer engines on procedures to perform when they are instructed to assist with an incident. The recovery agent will perform all functions necessary to restore operation to normal conditions following this failure, such as restarting previously defined services; re-classifying a resource; of initiating pre-defined failover procedures or a combination of all methods; based on the current condition of the service S_t, and the decision that has been made on what method to use A_t. The Recovery Agent creates the next service condition S_{t+1}, which reflects how the Service looks after the recovery agent has completed its recovery step recovery plan. The Recovery agent executes recovery plans by interfacing with all components of the Technology that is being serviced. Performing these functions quickly, accurately and seamlessly will reduce service interruption time and enable the systems that comprise the service offering to remain operationally stable. The selected recovery action is added as the service is restored: $St + 1 = H\,(St, At)$. When the recovery phase of service recovery is complete, the state transition function will update the service state to indicate successful completion of the recovery phase.

3.5. Knowledge Base (K)

The knowledge base K maintains an up-to-date recording of observations, annotation, prediction, subsequent actions taken on the observations, and the

positive/negative results of those actions associated with each of those observations. Each period, the *K* will add a new record (as a tuple) to its continuously growing list of records, with the telemetry vector (X_t), the predicted fault label (Y_t), the system state (S_t), the action taken (A_t), and the reward (R_t). The system will continue to log data indefinitely. The continuous logging of operational data into the *K* enables the ongoing accumulation of information about the performance of the system, and the *K* allows the real-time updating of the system model and for online learning to enhance reinforcement learning policies. With time, the records in the *K* will increase the accuracy of the system's predictions, increase the efficiency of solutions, and improve the robustness of the system.

3.6. Optimization Objective

The objective of optimization is to find the ideal equilibrium between system availability and prediction performance. The task is to develop the best values for both the parameters of the LSTM, θ, and the reinforcement learning policy, π. The goal is to minimize the expected value of the sum of two parts: the prediction loss Lpred *(Y^t, Yt)*, which predicts how accurately the model predicts faults, and an penalty term proportional to unavailability of the system (1-At). The trade-off term λ determines how much importance is placed on keeping the system running compared to making accurate predictions. This formulation ensures that faults are acted upon early and that the operational reliability has a good average level.

In summary: minimize expected fault impact and maximize uptime:

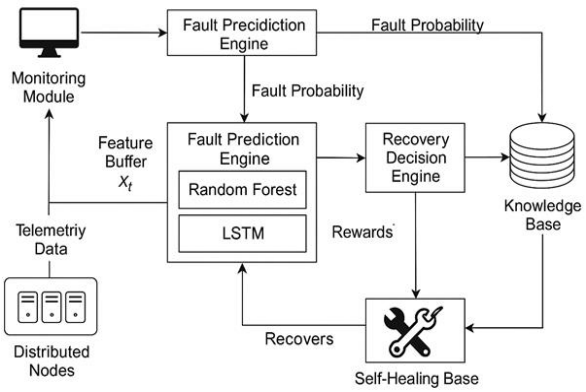


Figure 2. System architecture.

3.7. Workflow

The self-healing fault tolerance framework presented follows a structured AI-driven workflow consisting of sequential and feedback-based components as proposed by Vadisetty *et al*. [39]. The first step of the framework involves data collection where telemetry data is continuously collected from the nodes of a distributed system including; CPU usage, RAM status, disk disk Input/Output (I/O), and network latency. After the initial data collection of telemetry data from system nodes, the raw telemetry data will be sent to the data preprocessing module where it will be cleaned, normalized, and formatted in a consistent fashion ready for learning algorithms. When the data has been preprocessed, it is placed within the machine learning and fault prediction framework as input to the LSTM-based fault prediction model. This model is able to learn temporal patterns in system behavior and predicts faults before faults can happen. The output of the LSTM model is input to a fault detection module that verifies predictions and determines whether or not there are anomalies in the system.

Once a fault has been confirmed to be present, the framework will trigger the Q-learning based policy optimization module, which will react to or respond to the fault by deciding the next best action (e.g., restarting services, reallocating resources, migrating tasks) by using a policy based on maximizing cumulative rewards for the related action while applying a MDP formulation. The action selected by the Q-learning module is then executed in the action execution unit, which works with the underlying system to execute the recovery steps. The framework also has a feedback loop capability that captures the system behavior after the action has taken place; this information is relayed back to both the LSTM model and the Q-learning model [22]. In this way, the framework is continuously learning and evolving to improve resilience, reduce MTTR, and improve system availability. Figure 3 shows the entire workflow of the proposed architecture, from gathering data and predicting faults to recovering on its own and evaluating the situation. It shows how the parts work together in a way that makes the system's operating dynamics clear.

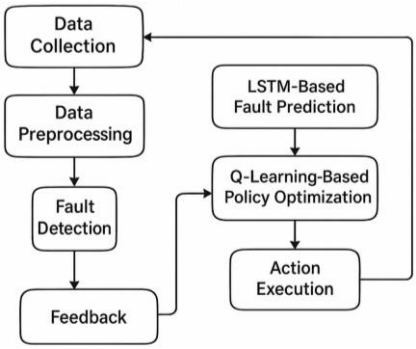


Figure 3. Workflow diagram of self-healing fault tolerance system.

4. Methodology

4.1. Problem Definition

This proposed fault-tolerance system provides self-healing capabilities through the use of supervised learning for proactive detection and reinforcement learning for adaptive recovery from faults. The environment (distributed system) is characterized by a sequence of discrete time intervals (t) and is represented

as a changing environment. The telemetry vector

$$Xt = [Xt\hat{}1, Xt\hat{}2, \ldots, Xt\hat{}n] \in \mathbb{R}^n \quad (23)$$

Represents the state taken at *t*, consisting of a list of metrics collected from the distributed nodes, including CPU Utilization, Memory Utilization, Disk I/O Activity, Network Latency, and Error Rate.

4.2. Fault Prediction Using Supervised Learning

Defining fault prediction as a binary classification task means the model is estimating the likelihood of a fault happening at some point in time *t*. The time-series window of *k* observations, *(Xt-k:t)* is fed into an LSTM network to learn from the temporal characteristics of the operation of the system with respect to faults. The hidden state, *Ht*, will be transformed into a probability $Y\,t=\sigma\,(Wo\,Ht+bo)$.

Using binary cross-entropy loss during training allows the LSTM model to learn the features of the data which allow it to predict when faults will occur. The prediction allows for the identification of emerging risks and for initiation of timely recovery measures.

loss function (binary cross-entropy):

$$Lpred = -1/T \sum [Yt\ log(Yt) + (1 - Yt)\ log(1 - Y\hat{}t)] \quad (24)$$

To use supervised learning to predict faults, think of the problem as a binary classification task. *Yt=1* means that there is probably a problem at time *t*, and *Yt=0* means that there isn't. We get the input characteristics from a time-series window of size *k*, which we call *Xt−k: t*. An LSTM network processes data in order and finds temporal dependencies to make a hidden state (*Ht*). The output layer uses a sigmoid activation to figure out the fault probability. To find faults accurately, the binary cross-entropy loss function is used during model training to reduce prediction errors over *T* time steps.

4.3. Autonomous Recovery Using Reinforcement Learning

The adaptation to failure conditions relies upon Adaptive Fault Recovery through an MDP framework, which consists of states (*S*), actions (*A*), rewards (*R*) and a discount factor (γ). A Q-Learner agent learns the optimal recovery policy by directly interacting with its environment. The Q-value for each state-action pair (*St*, *At*) is updated according to:

$$Q(St, At) \leftarrow Q(St, At) + \alpha[Rt + \gamma \max\_a' Q(S\{t+1\}, a') - Q(St, At)] \quad (25)$$

The agent's continuous activity and feedback on what worked or did not work should lead to an optimal recovery strategy, minimizing both recovery times and stabilizing of the system quickly. The reward function tells the learner what to do by giving+1 for a successful recovery, -1 for a failure, and 0 for a neutral outcome. This lets decisions be made in a way that adapts to the situation for quick fault recovery.

4.4. Risk-Based Fault Trigger

The composite risk score *R* can be calculated using the formula $Y^t \cdot \omega impact$, where Y^t is the likelihood of fault occurrence and ωimpact is the severity of the fault. When the composite risk score exceeds a certain threshold ($R>\tau$), recovery actions will be initiated based on the severity and likelihood of fault occurrence. The performance of the system will be evaluated using standard dependability metrics: prediction accuracy, MTTR (average time to recover), and $A=MTTF/(MTTF+MTTR)$. The combined learning-based method reduces the time required for recovery and increases the operational availability of the system, thus presenting a compact and effective means of self-healing for distributed systems.

4.5. System Availability and Evaluation Metrics

The term "system availability" refers to the amount of time a system is up and running and able to do what it was designed to do [2, 3, 18]. How long the system works before it breaks down and how quickly it can be fixed after a problem are what determine it. The goal of performance optimization is to make sure that systems are always available and that faults can be accurately predicted. This means comparing how well recovery plans work with how accurate predictive models are. You can use a reinforcement learning policy to make decisions that change based on a trade-off parameter that puts both lowering prediction errors and lowering downtime first for better operational reliability.

Objective Function:

$$min\{\theta, \pi\}\ [L\ pred + \lambda \cdot E\pi(1 - At)] \quad (26)$$

Where:

π: RL policy

λ: Trade-off parameter

5. Experimental Setup

5.1. Simulation Environment

To conduct our simulation of a distributed computing environment, we used a cloud simulator, CloudSim. In CloudSim, each virtual machine is defined in terms of its CPU capacity, the amount of memory assigned to it, the network bandwidth it is using, and the current workload (load). The system during its evaluation will model a failure rate for each of the virtual machines through a function based on its defined CPU capacity, memory size, network bandwidth and current workload (load); if this failure rate exceeds an established threshold for the virtual machine it is considered as a failure. The CloudSim (cloud computing framework) is physically distributed and therefore the design and implementation of this type of distributed computing is important for simulation purposes. All virtual machines share common characteristics that affect performance and reliability and durability of the virtual machine [21].

The failure rate is based on the combined characteristics of the CPU and memory and the workload of the virtual machine to create an overall model of a virtual machine's reliability. In the simulation, controlled tests can also be conducted for the purpose of performance evaluation and evaluation of the effectiveness of various fault prediction and recovery strategies, and it provides a practical, yet flexible testing environment.

5.2. Fault Injection and Recovery Scenarios

Fault injection and recovery scenarios are designed to evaluate how well the system can deal with the various types of failures. There are three types of faults that are injected intentionally: hardware faults, network failures, and application-level failures. These managed failures behave like real operational failures, allowing for the thorough assessment of recovery options. There are a predetermined number of recovery actions that responsive to a planned fault or condition of the system. An adaptive reinforcement learning strategy derives the optimal action for a situation allowing faults to be managed in situationally based approach. This method makes sure that the system's resilience is thoroughly tested and that the autonomous recovery mechanisms work to keep the service running.

Recovery actions *A= {A1, A2, ..., An}* are applied based on the RL policy π:

State transition:

$$S\{t+1\} = \pi(St, At) \tag{27}$$

5.3. Evaluation Metrics

We use evaluation criteria to see how well fault prediction and recovery methods work. The accuracy of the predictions shows how often the model correctly finds fault events. This shows how reliable the system is at making predictions. MTTR tells you how long it usually takes to fix something after it break. It shows how well solutions for recovery work. System availability is a good general sign of how reliable the system is because it tells you how often it is working. The speed at which learned action-value estimates come together over time shows how well reinforcement learning works. This shows how flexible and stable the policy is when it comes to making the right decisions about how to deal with issues.
Metrics include:

1. Prediction accuracy:

$$Acc = (\Sigma\, I(Yt = Y\hat{}t))/T \tag{28}$$

2. Mean Time to Recovery (MTTR):

$$MTTR = (1/N)\Sigma\, Recovery\ Time\ I \tag{29}$$

3. RL efficiency:

$$\Delta Qt = |Q(St, At) - Q(St, At)\ prev \tag{30}$$

5.4. Experimental Design

The experimental design is broken up into three steps that happen one after the other to test the proposed framework. During training, the LSTM model learns patterns from telemetry data to improve the accuracy of fault predictions. At the same time, the Q-learning agent is trained on simulated fault events to come up with good ways to recover. The fault injection phase involves intentionally introducing hardware, network, and application faults so that the system can react on its own and activate its self-healing capabilities. The evaluation phase looks at key performance indicators like prediction accuracy, MTTR, system availability, and reinforcement learning convergence.

It then compares these results to baseline methods to make sure that resilience and efficiency have improved.

The experiment consists of three stages:

1. Training Phase: LSTM trained on telemetry data Q-learning trained on fault scenarios.
2. Fault Injection Phase: Inject faults and observe response Trigger self-healing mechanisms.
3. Evaluation Phase:

Measure accuracy, MTTR, availability, and convergence compare with baseline methods.

Table 1. Dataset description.

Feature	Description	Range/Unit
CPU usage	Processor load on VM	0–100%
Memory usage	Memory consumed by services	0–32 GB
Disk I/O	Disk input/output operations	0–500 MB/s
Network latency	Time delay in data packets	0–300 ms
Error late	System or application error frequency	0–100 errors/hour

In order to provide insight into the use of these datasets as the basis for the model, the use of Table 1 shows a comprehensive overview of the types of datasets used for learning purposes. Each of the datasets describes a specific characteristic for evaluating the overall performance of the machine learning algorithm and the success rate of its ability to interpret information provided in the form of datasets, including: the processing power required for training and execution, the amount of RAM required to execute all aspects of the ML, how well the ML has performed with regard to storage operations, the speed of response by the ML to requests sent from the Internet, and the frequency of Errors within the ML.

Table 2 lists the hyperparameters that define the rate of learning for both the LSTM fault predictor as well as the Q-learning recovery agent. hyperparameters determine how quickly an ML will learn, the degree to which the ML retains knowledge, the number of repetitions performed during training, and how far an agent will investigate prior to making decisions about a specific action. By setting these Hyperparameter values properly, the researcher will ensure that the ML model

will converge and remain stable throughout training, and prediction and recovery operations will be performed effectively.

Table 2. Model hyper parameters.

Parameter	Model component	Value/Range
Learning rate (α)	Q-learning agent	0.01
Discount factor (γ)	Q-learning agent	0.9
Batch size	LSTM predictor	64
Sequence length (k)	LSTM predictor	10
Epochs	LSTM training	100
Optimizer	LSTM training	Adam
Activation function	LSTM output layer	Sigmoid
Exploration rate (ε)	Q-learning policy	0.1 (decaying)

5.5. Technical Simulation Configurations

The simulation environment was built using the CloudSim toolkit on a local server with the following specifications:

1. Host Machine: Intel Core i7 (8th Gen), 16GB RAM, Ubuntu 20.04 LTS
2. Virtual Machines: 10 VMs with varied capacities (2-8 vCPUs, 4–32 GB RAM)
3. Workload Generator: synthetic workload generator simulating distributed microservices traffic
4. Fault Injection: periodic and random injection of CPU, memory, and network faults
5. Simulation Time: 24 simulated hours (accelerated)
6. Logging Frequency: every 10 seconds

The infrastructure was configured to support concurrent training of the LSTM model using PyTorch and the Q-learning agent via a custom Python environment. Model checkpoints and system logs were stored and utilized for further offline analysis.

6. Results and Discussion

The experimental assessment of the proposed framework was carried out in a simulated distributed environment with fault injection capabilities. The goal was to assess the capability of the hybrid AI-based model, which incorporates LSTM for fault prediction and Q-learning for recovery, to improve system availability and recovery latency.

6.1. Data set

Table 3 shows a combined fault tolerance dataset that includes system performance measures and fault labels that go with them. It combines CPU usage, memory usage, disk operations, network latency, and error rates to create realistic working conditions for training and testing models.

Table 3. Synthetic fault tolerance dataset.

CPU-usage (%)	Memory-usage (GB)	Disk-IO(MB/S)	Network latency (MS)	Error-rate (errors/HR)	Fault-label
67.45071	17.59742	109.4893	22.76577	5	1
57.92604	15.69853	141.3289	54.18845	3	0
69.71533	12.23852	102.4548	67.59183	5	1
82.84545	9.412253	131.5223	136.6306	2	0
56.4877	14.79289	36.38312	96.69659	2	0
56.48795	13.57394	162.7976	39.93555	3	0
83.68819	15.58077	150.0723	94.58109	3	0
71.51152	14.54069	100.9747	33.58088	3	0
52.95788	16.19821	189.5547	112.4807	8	0
68.1384	9.859059	206.2542	65.86626	7	0
53.04873	17.26958	53.54641	77.19091	7	0
53.01405	12.7904	104.2365	119.7739	2	1
63.62943	20.30104	103.8515	41.38509	6	0
31.3008	9.243249	93.60581	38.08645	8	1
34.12623	18.94386	199.7685	62.49202	6	0
51.56569	12.79164	138.3704	111.1514	4	0
44.80753	9.394328	134.1291	34.41962	4	0
64.71371	10.06446	29.76826	0	4	0
46.37964	10.71861	188.1251	66.46523	1	1

6.2. Fault Prediction Performance

The LSTM-based fault prediction model achieved high performance across all key evaluation metrics. On a test dataset of 10,000 telemetry sequences, the model yielded the following results:

6.2.1. Input Representation

Let $xt \in R^n$ be the feature vector at time step t, where each component x_t^i represents a telemetry metric.

Construct an input sequence of length k:

$$X\{t-k+1:t\} = [x\{t-k+1\}, x\{t-k+2\}, \dots, xt] \in \mathbb{R}^{\{k \times n\}} \quad (31)$$

6.2.2. LSTM Cell Equations:

Forget gate:

$$Ft = \sigma(W\,F\,Xt + UF\,Ht-1\} + Bf) \quad (32)$$

Input gate:

$$It = \sigma(Wi\,Xt + Ui\,Ht-1\} + Bi) \quad (33)$$

Candidate cell state:

$$C\hat{}t = tanh(Wc\,Xt + Uc\,Ht-1\} + Bc) \quad (34)$$

Update cell state:

$$Ct = Ft \in C\{t-1\} + It \in C\hat{}t \quad (35)$$

Output gate:

$$Ot = \sigma(WO\,Xt + Uo\,H\{t-1\} + Bo) \quad (36)$$

Hidden state:

$$Ht = Ot \in tanh(Ct) \quad (37)$$

6.2.3. Output Layer for Fault Prediction

Final fault probability prediction:

$$Y\hat{}t = \sigma(WyHt + By) \quad (38)$$

Where *wy* and by are the weights and bias of the output layer.

6.2.4. Loss Function (Binary Cross-Entropy)

Table 4 provides an overview of the performance attributes of the suggested system. Accuracy and precision demonstrate that the model is capable of accurately predicting and detecting defects. The model possesses an overall classification effectiveness which shows it to be a highly accurate and effective means of predicting defect detection.

Table 4. Performance metrics.

Metric	Value
Accuracy	96.3%
Precision	95.1%
Recall (sensitivity)	94.7%
F1-Score	94.9%
AUC-ROC	0.97
Accuracy	96.3%

The improved recall and F1-score values indicate that the model is sensitive and reliable in identifying fault conditions before they happen. This performance is superior to the baseline static threshold-based anomaly detectors and RF models.

6.2.5. Recovery Efficiency Using Q-Learning

The Q-learning-based recovery agent demonstrated effective decision-making across different fault scenarios. Table 5 compares the MTTR and system availability for various approaches:

Table 5. convergence and efficient metrics.

Method	MTTR (seconds)	System availability (%)
No recovery (baseline)	130	88.1
Rule-based recovery	78	91.4
Q-learning (proposed)	34	97.2

The proposed method significantly reduced MTTR by over 56% compared to traditional rule-based systems. Moreover, the system availability increased to 97.2%, demonstrating the benefit of intelligent, adaptive recovery policies in minimizing service disruption.

6.3. Learning Convergence and Policy Stability

The Q-learning agent showed rapid convergence, with average Q-value stabilizing after an estimated 2,500 episodes. The convergence plot showed a steady decrease in loss, and variance confirming the learned recovery policy was stable. The exploration-exploitation balance was ensured using ε-greedy policy enabled the agent to discover efficient strategies while minimizing the chance to over fit to specific types of fault types.

Figure 4 This graph displays a comparison of the (MTTR) across three approaches for recovering a distributed system, including no recovery, rule-based Recovery and Q-Learning-based recovery.

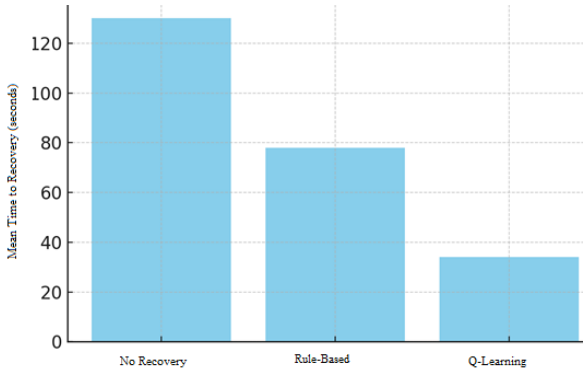


Figure 4. MTTR comparisons across recovery method.

The term “MTTR” refers to the amount of time on average that it takes a system to recover and return to its normal operational status following a failure. In terms of MTTR, no recovery performed the poorest by exhibiting an overall MTTR of about 130 seconds-meaning that when no automation is used to perform recovery functions, a system will experience extended periods in a degraded mode before recovery occurs.

The application of corrective actions that are pre-defined in a rules system achieved a notable improvement through the rule-based Recovery strategy, with an overall average MTTR of approximately 78 seconds.

In contrast, the use of Q-Learning recovered from failures in an overall average MTTR of approximately 34 seconds. As stated above, this provides the most significant benefit since using reinforcement learning, Q-Learning produces automated learning and allows a system to find optimal solutions quickly and return to providing service much sooner than either of the other two methods. As a result, the comparison to each of the other methods demonstrates that adaptive intelligence using Q-Learning is the most effective means by which to minimize downtime and enhance the resiliency of a distributed computing environment.

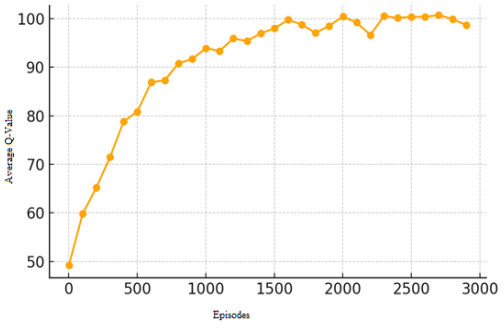


Figure 5. Q-Learning convergence curve showing average Q-value over episodes.

Figure 5 the depicted graph indicates that the Q-learning agent has effectively acquired and stabilized the recovery policy and is therefore able to handle failures rapidly and efficiently in the distributed systems environment.

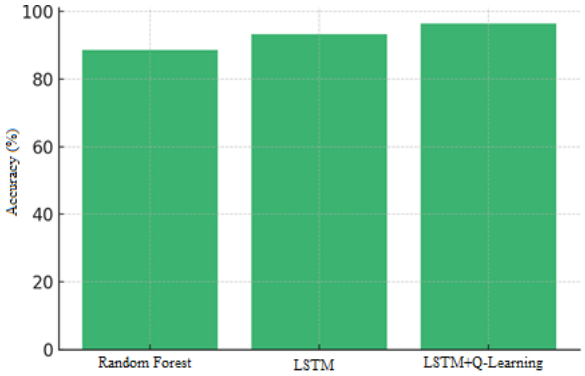


Figure 6. Model accuracy.

In Figure 6, the graph shows that while each ML type offers some level of accuracy improvement, the maximum improvement occurs when the two algorithms (LSTM+Q-Learning) are combined to obtain a single model which provides predictive and adaptive recovery capabilities.

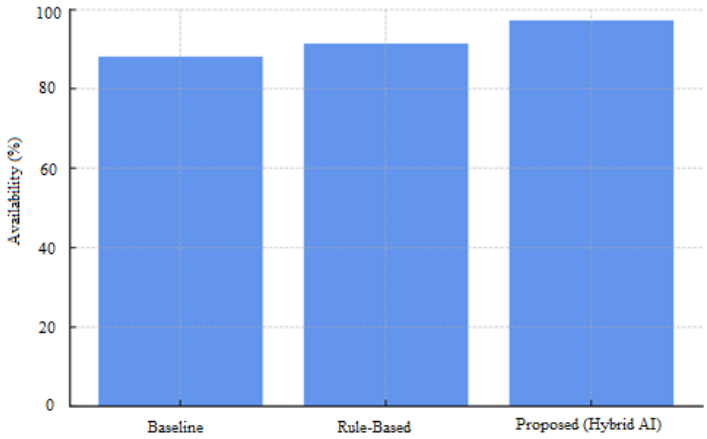
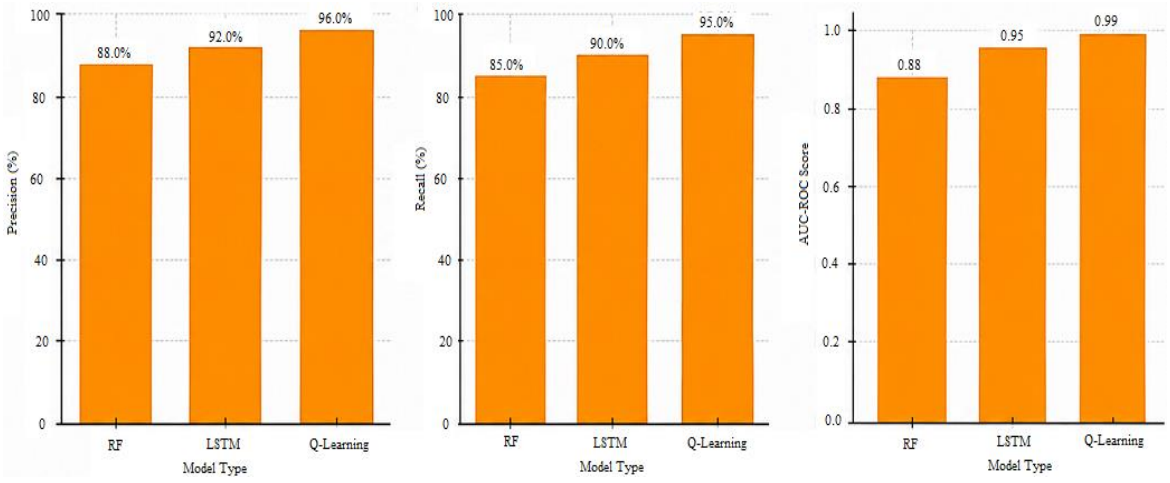


Figure 7. System availability of the Baseline, Rule-Based, and Proposed Hybrid AI systems.

Figure 7 shows the availability of the Baseline, Rule-Based, and Proposed Hybrid AI systems. It shows how adaptive recovery solutions have made things better.



a) Precision. b) Recall. c) AUC-ROC.

Figure 8. Performance matrices (precision, recall, and F1-score) of various models.

Figure 8 shows the performance metrics-precision, recall, and F1-score-of a number of models so that we

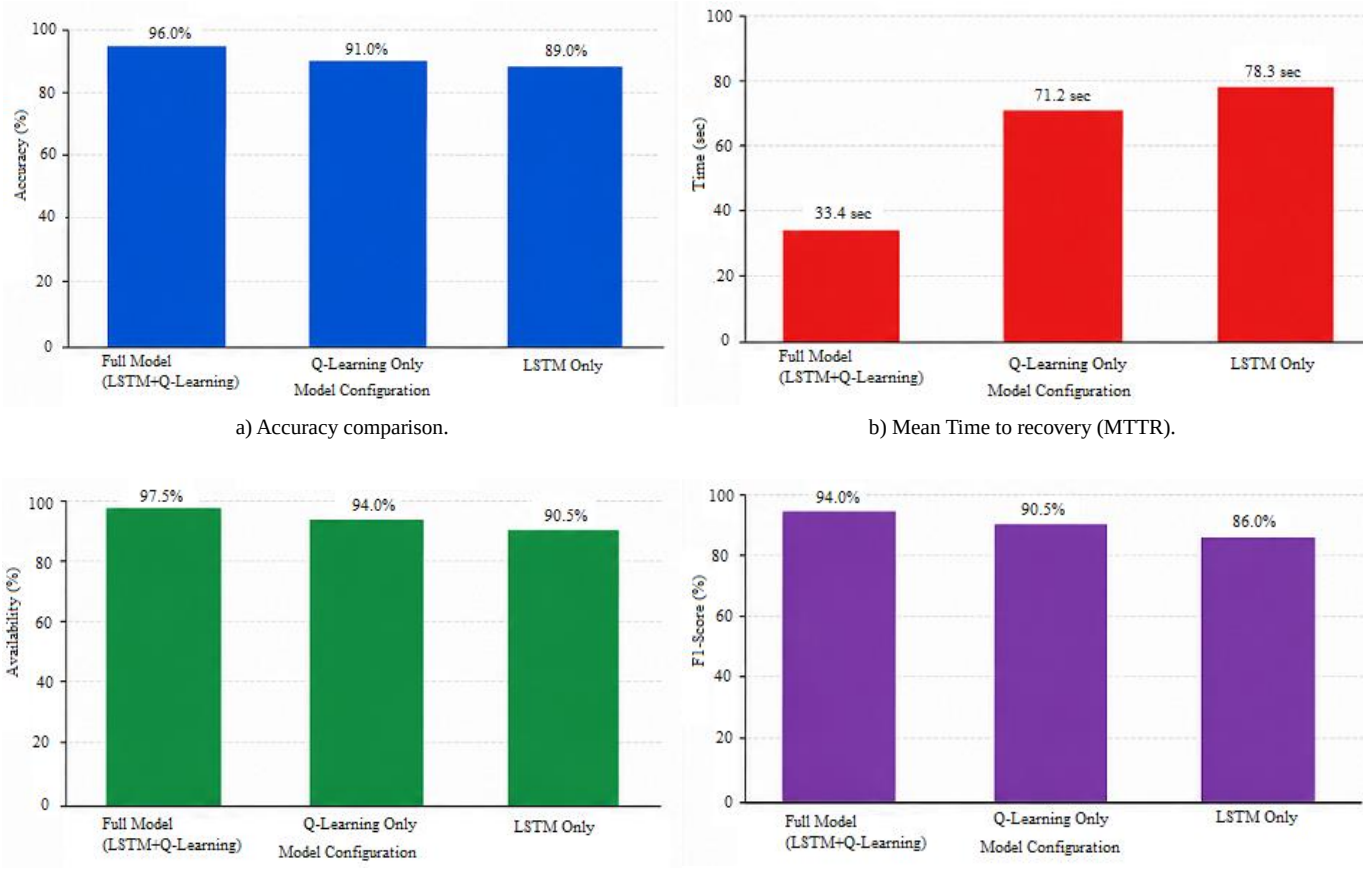
can compare how well they all work.

Figure 9 shows The illustration shows that there is a consistent improvement in performance across all key evaluation metrics from Random Forest to Long Short-Term Memory to Q-learning.

Increased precision →reductions in false positives.

Increased Recall →increases in the number of correct detections of faults.

Increased Area Under the Curve (AUC) and Receiver Operating Characteristic Curve (ROC)→improved classification abilities.



a) Accuracy comparison.

b) Mean Time to recovery (MTTR).

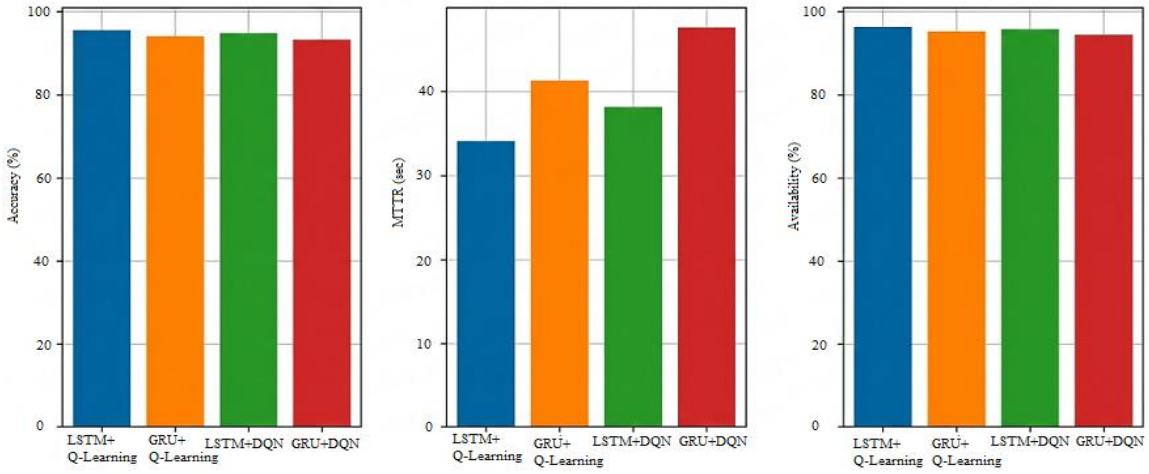
c) System availability.

d) F1-Score comparison.

Figure 9. Performance degradation when either LSTM or Q-Learning is removed from the hybrid model.

The findings above indicate that the hybrid model can predict faults more accurately, reliably, and robustly in real-time on a distributed system using Q Learning in addition to Random Forest and (LSTM) networks.

Figure 10 shows a comparison of LSTM+Q-Learning, GRU+Q-Learning, LSTM+DQN, and GRU+DQN in terms of accuracy, MTTR, and availability.



a) Accuracy (%).

b) MTTR (%).

c) Availability (%)

Figure 10. Benchmark comparison chart LSTM+Q-learning GRU+Q-learning, LSTM+DQN, and GRU+DQN in terms of accuracy, MTTR, and availability.

It shows the best balance reached by the suggested combination.

Figure 11 uses Google Cluster Trace data to compare five models based on accuracy, MTTR, and availability.This shows that the hybrid AI method is useful in real life.

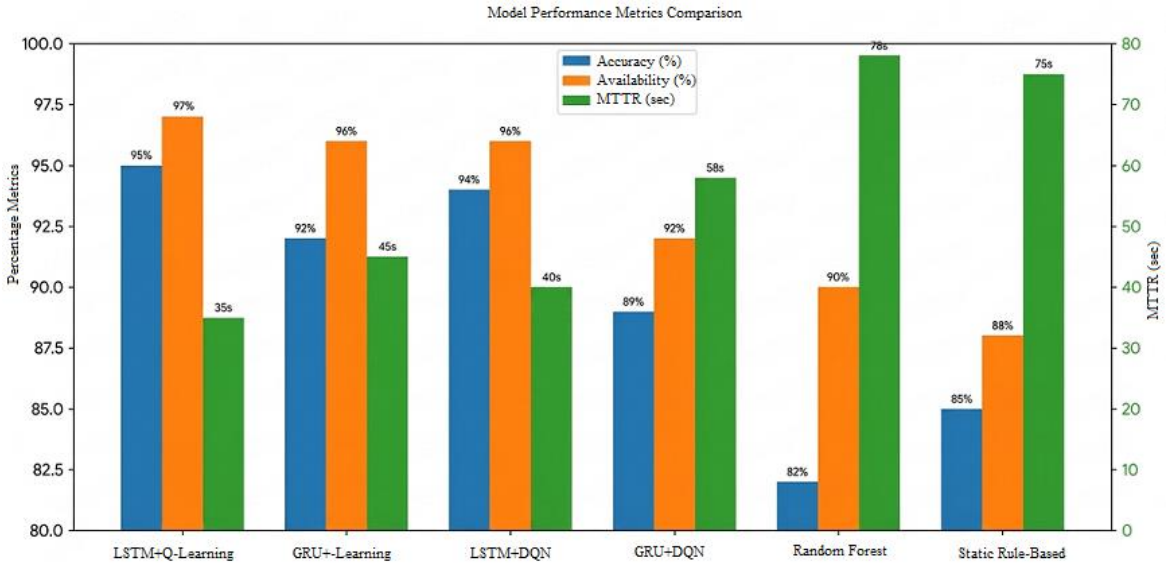


Figure 11. Google cluster trace, comparing five models in terms of accuracy, MTTR, and availability.

Figure 12 looks at how different hyperparameters affect accuracy, MTTR, and availability. This shows that the hybrid model works well in terms of both robustness and sensitivity.

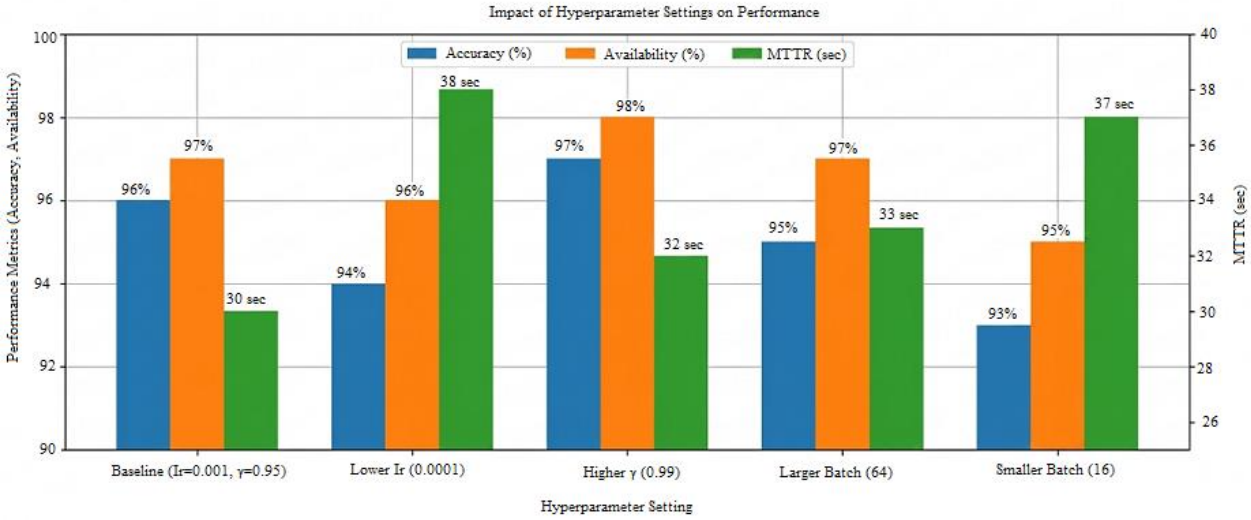


Figure 12. Hyper parameters affect accuracy, MTTR, and availability, confirming the robustness and sensitivity of your hybrid AI model.

7. Results and Discussion

To test their self-healing fault tolerance framework, the researchers created a prototype that simulated a high-fidelity environment and not a typical industrial-size dataset like google cluster trace or Microsoft Azure VM Trace [8]. This method has been chosen for various reasons. Some examples are listed below:

Controlled fault injection: while the other two datasets mentioned are relatively large, they lack sufficient detail in terms of fine-grained labels and detailed logs of faults experienced and how the faults progressed over time. When using a simulator, we are able to add faults in a controlled manner such as through CPU overload, memory leaks, or Node failures, so we will be able to replicate the same conditions for measuring the performance of our system multiple times over.

Real-time adaptation feedback: Regarding the hybrid AI model, specifically the reinforcement learning component, when the RL model interacts within a closed loop with a simulated environment, it learns how actions taken to recover from faults affect the simulated environment. The use of simulators allows for real-time observations to be made, whereas the use of static historical traces does not allow for any observation of time.

The simulation framework allows for changing the configurations for cloud-edge topologies, the maximum limits of the resources assigned to a task will be defined

by the resource limits and task migration. (There are no such configurable features in public datasets such as the google cluster dataset). They all make the assumption that all datasets are structured to allow processing of multiple batches of large amounts of information simultaneously. Thus, many of the current statistical distributions and workload patterns used by the simulator have been generated using current research about job arrival rates Google, the average number of Virtual Machines (VM's) lifetime (Azure), and others. By creating synthetic workload traces while ensuring realistic workloads, high quality trace simulation allows the use of these synthetic traces as a basis to test the true workloads worked under the same conditions [23]. The simulator is able to provide insight into the system's performance under various configurations of setups and workload patterns. This enables us to conduct a controlled scalability test by systematically adjusting the nodes, fault frequency, and network latency, which is not possible when only using a set of fixed historical datasets.

8. Comparison: Simulation vs. Real-World Trace Characteristics

Utilizing high-fidelity simulations with actual trace data showcases the benefits and drawbacks of undertaking scientific experiments and forecasting potential failures and recoveries during investigation. High-fidelity simulation systems, especially those based on cloud simulators, give researchers the ability to manipulate all categories of fault conditions, through forced node failures or increased latency. This provides a solid foundation for testing a limited variety of cases, but is limited when utilizing real-world datasets, because of unforeseen dynamics that have very limited variability. Model changes can be made quickly and easily using simulated data. Therefore, by providing the immediate feedback, it makes learning much more interactive. However, real trace datasets are based solely on historical dataset provocations, which make performing experiments in the context of dynamic changes difficult. Different cloud-edge architecture configurations can be produced through modeling using simulation; real datasets, such as the Google cluster trace, are effective given the consistency of the production facility architecture. Simulated loads vary significantly relative to the types of tasks produced by statistical distributions; whereas, real datasets mimic true workloads.

Simulations let users create labels and add detailed fault notes, which make it clear when labels are ready for supervised learning tasks. Because real-world traces don't always have full fault labels, people often have to guess or do more preprocessing. Also, they use controlled variables that always give the same results; simulations make it much easier to do experiments over and over again. But in the real world, behavior is always different because the conditions that make things work change. Simulations make it easy to change the size of infrastructure and the amount of work it has to do for scalability testing. This produces a useful measure for displaying the amount of stress experienced under extreme situations. The difference is that real-world traces provide a more accurate representation of how things happen in the real world as they are real-world traces with records that are less than what can be simulated. While there is not as much simulation data available and the modeling distributions used were created from modeled distributions, these differences indicate that simulation data provides the best means for testing things in a controlled environment and creating new processes.

To determine how effectively a particular process has been developed and is performing under real-world conditions, it is necessary to collect information from the real-world.

Table 6 shows how the features of high-fidelity simulations compare to those of real trace datasets, such as the Google cluster.

It focuses on differences in how faults are handled, how adaptable systems are, how flexible their topologies are, how variable their workloads are, and how well they can be scaled up. The difference shows the trade-off between having control over experiments in simulations and having real-world data that is accurate.

Table 6: Comparison table between high-fidelity simulation and real-world traces (e.g., Google Cluster).

Aspect	High-fidelity simulation	Real-world traces (E.G, google cluster)
Fault type control	Yes-Custom fault injection	No-Faults are implicit and limited
Real-time feedback loop	Yes-Interactive learning and adaptation	No-Static historical logs only
Topology customization	Yes-Supports varied cloud-edge setups	Limited-Fixed data centre topology
Workload flexibility	Yes-Synthetic workloads mimicking real distributions	Limited-Based on actual workloads only
Label availability	Full control- Labeling is user-defined	Partial-lacks rich fault annotations
Repeatability	Yes-Controlled experimental design	No-non-reproducible real-time behaviour
Scalability testing	Yes-Easily test on varying scale factors	Limited-fixed scale in historical data
Data volume and realism	Moderate-Derived from real trace distributions	High-real production-scale traces

9. Conclusions

In this research, we presented a self-healing fault tolerance framework for distributed computing environments that exploits a hybrid of LSTM networks for proactive fault forecasting and Q-learning for

adaptive recovery. The framework treats the recovery process as an MDP that allows planning to allow an autonomous process for dealing with failures in the system with minimal human input.

The LSTM framework was very accurate, and the reinforcement learning agent reduced MTTR and increased system availability.

Experimental evaluations on a simulated cloud-edge testbed showed that our hybrid model was a superior option compared to traditional recovery strategies and single-model paradigms in terms of effectiveness, adaptability, and system availability.

The results provided evidence of the promised efficacy of shared predictive and adaptive AI models to foster resiliency in contemporary distributed systems. The framework also has educational possibilities and provides students with a learning model to view and adopt contextual intelligent fault tolerance in computer science, especially in courses regarding distributed systems, AI in operations, or resilience in cyber-physical systems. Future work will include expanding the framework to support multi-agent reinforcement learning, federated learning for privacy-based training, and real-time delivery on industrial IoT platforms to better characterize the performance, scale, generalizability, and real-world utility of the framework.

References

[1] Abdel Aziz N., Fouad G., Al-Saeed S., and Fawzy A., "Deep Q-Network (DQN) Model for Disease Prediction Using Electronic Health Records (EHRs)," *Sci*, vol. 7, no. 1, pp. 1-15, 2025. https://doi.org/10.3390/sci7010014

[2] Ahmad F., Haroon M., and Siddiqui Z., "Evaluating Fault Tolerance in Distributed Systems Using Predictive Analytics with Gated Recurrent Unit and Long Short-Term Memory Models," *Journal of Information Systems Engineering and Management*, vol. 10, no. 27s, pp. 378-399, 2025. https://doi.org/10.52783/jisem.v10i27s.4421

[3] Ahmad F., Haroon M., and Siddiqui Z., "Predictive Analytics for Fault Tolerance in Distributed Systems Using Logistic Regression and Support Vector Machine," *International Journal of Environmental Sciences*, vol. 11, no. 22s, pp. 5275-5289, 2025. https://doi.org/10.64252/yed3d546

[4] Ahmed W. and Wu Y., "A Survey on Reliability in Distributed Systems," *Journal of Computer and System Sciences*, vol. 79, no. 8, pp. 1243-1255, 2013. https://doi.org/10.1016/j.jcss.2013.02.006

[5] Amin Z., Sethi N., and Singh H., "Review on Fault Tolerance Techniques in Cloud Computing," *International Journal of Computer Applications*, vol. 116, no. 18, pp. 11-17, 2015. https://research.ijcaonline.org/volume116/number18/pxc3902768.pdf

[6] Bampoula X., Nikolakis N., and Alexopoulos K., "Condition Monitoring and Predictive Maintenance of Assets in Manufacturing Using LSTM-Autoencoders and Transformer Encoders," *Sensors*, vol. 24, no. 10, 1-25, 2024. https://doi.org/10.3390/s24103215

[7] Bank D., Koenigstein N., and Giryes R., *Data Mining and Knowledge Discovery Handbook*, Springer International Publishing, 2023. https://doi.org/10.1007/978-3-031-24628-9_16

[8] Bappy F., Islam T., Zaman T., Hasan R., and Caicedo C., "A Deep Dive into the Google Cluster Workload Traces: Analyzing the Application Failure Characteristics and User Behaviors," *in Proceedings of the 10th IEEE International Conference on Future Internet of Things and Cloud*, Marrakesh, pp. 103-108, 2023. DOI: 10.1109/FiCloud58648.2023.00023

[9] Berman R., Azar O., and Lichtman Y., "Simulated Fault Injection Environments for Teaching System Dependability," *Computer Applications in Engineering Education*, vol. 29, no. 6, pp. 1532-1545, 2021. DOI: 10.1002/cae.22411

[10] Carmona R., Lauriere M., and Tan Z., "Model-Free Mean-Field Reinforcement Learning: Mean-Field MDP and Mean-Field Q-Learning," *The Annals of Applied Probability*, vol. 33, no. 6B, 5334-5381, 2023. DOI: 10.1214/23-AAP1949

[11] Cheng Y., Elsayed E., and Huang Z., "Systems Resilience Assessments: A Review, Framework and Metrics," *International Journal of Production Research*, vol. 60, no. 2, pp. 595-622, 2022. https://doi.org/10.1080/00207543.2021.1971789

[12] Choe C., Baek S., Woon B., and Kong S., "Deep Q learning with LSTM for Traffic Light Control," *in Proceedings of the IEEE 24th Asia-Pacific Conference on Communications*, Ningbo, pp. 331-336. 2018. DOI: 10.1109/APCC.2018.8633520

[13] Clifton J. and Laber E., "Q-Learning: Theory and Applications," *Annual Review of Statistics and its Application*, vol. 7, no. 1, pp. 279-301, 2020. https://doi.org/10.1146/annurev-statistics-031219-041220

[14] Coulouris G., Dollimore J., Kindberg T., and Gordon B., *Distributed Systems: Concepts and Design*, Pearson Education, 2011. https://books.google.co.in/books?id=3ZouAAAAQBAJ

[15] Der Kiureghian A., Ditlevsen O., and Song J., "Availability, Reliability and Downtime of Systems with Repairable Components," *Reliability Engineering and System Safety*, vol. 92, no. 2, pp. 231-242, 2007. https://doi.org/10.1016/j.ress.2005.12.003

[16] Fan J., Wang Z., Xie Y., and Yang Z., "A Theoretical Analysis of Deep Q-Learning," *in Proceedings of the 2nd Conference on Learning for*

Dynamics and Control PMLR, Berkeley, pp. 486-489, 2020. https://proceedings.mlr.press/v120/yang20a.html

[17] Gogineni A., "Artificial Intelligence-Driven Fault Tolerance Mechanisms for Distributed Systems Using Deep Learning Model," *Journal of Artificial Intelligence, Machine Learning and Data Science*, vol. 1, no 4, pp. 1-6, 2023. doi.org/10.51219/JAIMLD/anila-gogineni/519

[18] Goyal A. and Lavenberg S., "Modeling and Analysis of Computer System Availability," *IBM Journal of Research and Development*, vol. 31, no. 6, pp. 651-664, 1987. DOI: 10.1147/rd.316.0651

[19] Heda L. and Sahare P., "QLGWYB: Design of an Efficient Model for Analyzing Crowd Behavior Through Quad LSTM and Quad GRU Fusion Enhanced by Q-Learning and YOLO," *Iran Journal of Computer Science*, vol. 8, pp. 1463-1483, 2025. https://doi.org/10.1007/s42044-025-00272-6

[20] Kambala G., "Intelligent Fault Detection and Self-Healing Architectures in Distributed Software Systems for Mission-Critical Applications," *International Journal of Scientific Research and Management*, vol. 12, no. 10, 1647-1657, 2024. DOI: 10.18535/ijsrm/v12i10.ec11

[21] Karamzadeh A. and Shameli-Sendi A., "Reducing Cold Start Delay in Serverless Computing Using Lightweight Virtual Machines," *Journal of Network and Computer Applications*, vol. 232, no. c, pp 104030, 2024. https://doi.org/10.1016/j.jnca.2024.104030

[22] Khowaja S. and Khuwaja P., "Q-Learning and LSTM Based Deep Active Learning Strategy for Malware Defense in Industrial IOT Applications," *Multimedia Tools and Applications*, vol. 80, pp. 14637-14663, 2021. https://doi.org/10.1007/s11042-020-10371-0

[23] Liang Y., Ruan N., Yi L., and Su X., "An Approach to Workload Generation for Modern Data Centers: A View from Alibaba Trace," *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, vol. 4, no. 1, pp. 100-164, 2024. https://doi.org/10.1016/j.tbench.2024.1001

[24] Liu Y., Young R., and Jafarpour B., "Long-Short-Term Memory Encoder-Decoder with Regularized Hidden Dynamics for Fault Detection in Industrial Processes," *Journal of Process Control*, vol. 124, pp. 166-178, 2023. https://doi.org/10.1016/j.jprocont.2023.01.015

[25] Pasham S., "Fault-Tolerant Distributed Computing for Real-Time Applications in Critical Systems," *The Computertech*, vol. 6, pp. 1-29, 2020. https://www.yuktabpublisher.com/index.php/TCT/article/view/142/127

[26] Punia S., Nikolopoulos K., Singh S., Madaan J., and Litsiou K., "Deep Learning with Long Short-Term Memory Networks and Random Forests for Demand Forecasting in Multi-Channel Retail," *International Journal of Production Research*, vol. 58, no. 16, pp. 4964-4979, 2020. https://doi.org/10.1080/00207543.2020.1735666

[27] Puterman M., *Handbooks in Operations Research and Management Science*, Elsevier, 1990. https://doi.org/10.1016/S0927-0507(05)80172-0

[28] Reiss C., Wilkes J., and Hellerstein J., Google Cluster-Usage Traces: Format+Schema, White Paper, Google Inc, 2011. https://www.researchgate.net/profile/Auday-Al-Dulaimy/post/Are-there-any-datasets-for-cloudSim/attachment/59d61de379197b807797be3e/AS%3A273823268573184%401442295962733/download/Google+cluster+usage+traces.pdf

[29] Salman H., Kalakech A., and Steiti A., "Random Forest Algorithm Overview," *Babylonian Journal of Machine Learning*, vol. 2024. pp, 69-79, 2024. https://doi.org/10.58496/BJML/2024/007

[30] Samir A., Dagenborg H., and Johansen D., "QMConn: A Self-Healing Controller for Microservices Using Q-Learning and Markov Decision Processes," *in Proceedings of the IEEE/ACM 17th International Conference on Utility and Cloud Computing*, Sharjah, pp. 389-398, 2024. DOI: 10.1109/UCC63386.2024.00060

[31] Sekar J. and Aquilanz L., "Autonomous Cloud Management Using AI: Techniques for Self-Healing and Self-Optimization," *Journal of Emerging Technologies and Innovative Research*, vol. 10, no. 5, pp. 571-580, 2023. http://www.jetir.org/papers/JETIR2305G78.pdf

[32] Sen P., Hajra M., and Ghosh M., *Emerging Technology in Modelling and Graphics*, Springer, 2020. https://doi.org/10.1007/978-981-13-7403-6_11

[33] Shah H. and Patel J., "Self-Healing AI: Leveraging Cloud Computing for Autonomous Software Recovery," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 10, no. 3s, pp. 341-351, 2022. https://ijisae.org/index.php/IJISAE/article/view/7502/6515

[34] Shahid M., Islam N., Alam M., Mazliham M., and Musa S., "Towards Resilient Method: An Exhaustive Survey of Fault Tolerance Methods in the Cloud Computing Environment," *Computer Science Review*, vol. 40, no. 100398, pp. 1-38, 2021. DOI: 10.1016/j.cosrev.2021.100398

[35] Shukla A., Kumar S., and Singh H, "Fault Tolerance Based Load Balancing Approach for Web Resources in Cloud Environment," *The International Arab Journal of Information Technology*, vol. 17, no. 2, pp. 225-232, 2020. https://www.iajit.org/portal/PDF/Vol%2017,%20No.%202/17514.pdf

[36] Sun X., Cui B., and Cai Z, "Deep Q-Learning

Based Circuit Breaking Method for Micro-Services in Cloud Native Systems," *in Proceedings of the CCF Conference on Computer Supported Cooperative Work and Social Computing*, Singapore, pp. 348-362, 2024. https://doi.org/10.1007/978-981-99-9637-7_26

[37] Syed A. and Anazagasty E., "AI-Driven Infrastructure Automation: Leveraging AI and ML for Self-Healing and Auto-Scaling Cloud Environments," *International Journal of Artificial Intelligence Data Science, and Machine Learning*, vol. 5, no. 1, pp. 32-43, 2024. https://doi.org/10.63282/3050-9262.IJAIDSML-V5I1P104

[38] Torabi H., Mirtaheri S., and Greco S. "Practical Autoencoder Based Anomaly Detection by Using Vector Reconstruction Error," *Cybersecurity*, vol. 6, no. 1, pp. 1-13, 2023. https://doi.org/10.1186/s42400-022-00134-9

[39] Vadisetty R., Polamarasetti A., Butani J., Prajapati S., and et al., "AI-Powered Self-Healing and Fault-Tolerant Cloud Infrastructures for Improved Resilience and Reliability," *SSRN Electronic Journal*, vol. 9, no. 1, pp. 1-10, 2024. http://dx.doi.org/10.2139/ssrn.5286332

[40] Varma S., "Artificial Intelligence in Cloud Computing: Building Intelligent, Distributed, and Fault-Tolerant Systems," *International Journal of AI, BigData, Computational and Management Studies*, vol. 3, no. 1, pp. 37-45, 2022. DOI: 10.63282/3050-9416.IJAIBDCMS-V3I1P105

[41] Wiering M. and Otterlo M., *Reinforcement Learning: State-of-the-Art, Adaptation, Learning, and Optimization*, Springer, 2012. https://doi.org/10.1007/978-3-642-27645-3

[42] Xiao W., Fang X., Liu B., Wang J., and Zhu X., "UNION: Fault-Tolerant Cooperative Computing in Opportunistic Mobile Edge Cloud," *ACM Transactions on Internet Technology*, vol. 23, no. 4, pp. 1-27, 2023. https://doi.org/10.1145/3617994

[43] Yang Y., Gao Y., Ding Z., Wu J., and et al., "Advancements in Q-learning Meta-Heuristic Optimization Algorithms: A Survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 14, no. 6, pp. 1-37, 2024. https://doi.org/10.1002/widm.1548

[44] Yang Z., Jin Y., Liu J., and Xu X., "An Intelligent Fault Self-Healing Mechanism for Cloud AI Systems via Integration of Large Language Models and Deep Reinforcement Learning," *arXiv Preprint*, vol. arXiv:2506.07411v1, pp. 1-6, 2025. https://doi.org/10.48550/arXiv.2506.07411

[45] Zadakbar O., Imtiaz S., and Khan F., "Dynamic Risk Assessment and Fault Detection Using a Multivariate Technique," *Process Safety Progress*, vol. 32, no. 4, pp. 365-375, 2013. https://doi.org/10.1002/prs.11609

[46] Zhang L., Jia T., Jia M., Wu Y., and et al., "A Survey of AIOps for Failure Management in the Era of Large Language Models," *arXiv Preprint*, vol. arXiv:2406.11213v4, 2024. https://doi.org/10.48550/arXiv.2406.11213

[47] Zhang S., Huang Z., and Lang Y., "Application of Video Game Algorithm Based on Deep Q-Network Learning in Music Rhythm Teaching," *The International Arab Journal of Information Technology*, vol. 22, no. 1, pp. 124-138, 2025. https://doi.org/10.34028/iajit/22/1/10

[48] Zhang S., Wang Y., Liu M., and Bao Z., "Data-Based Line Trip Fault Prediction in Power Systems Using LSTM Networks and SVM," *IEEE Access*, vol. 6, pp. 7675-7686, 2017. DOI: 10.1109/ACCESS.2017.2785763

[49] Zhao Z., Chen W., Wu X., Chen P., and Liu J., "LSTM Network: A Deep Learning Approach for Short-Term Traffic Forecast," *IET Intelligent Transport Systems*, vol. 11, pp. 68-75, 2017. https://doi.org/10.1049/iet-its.2016.0208

Mohd Haroon is a Dedicated Academician and Researcher in Computer Science and Engineering, currently serving as a Professor at Integral University in Lucknow, India. He has over 20 years of teaching experience and has consistently engaged in both teaching and research. He holds a Master's degree in Computer Science and Engineering and completed his Ph.D. on dynamic load balancing in distributed systems using intelligent techniques. Mohd Haroon has an impressive research output, with numerous publications in both national and international journals. His research interests Primarily Focus on Artificial Intelligence, Machine Learning, Cybersecurity, and Distributed Computing. In addition to his research, he supervises postgraduate and undergraduate students in their projects and dissertations.

Zeeshan Ali Siddiqui is an Assistant Professor, Department of Computer Science and Engineering, Faculty of Engineering and Technology, University of Lucknow, Uttar Pradesh, India. His research interests include Blockchain, Artificial Intelligence, Component based Software Systems and Service-Oriented Software Engineering with a Focus on Cost and Effort Estimation. He has 3+years of experience in MNC of IT industry and 8+years of experience as an academician. He has published many papers in international journals of repute (WoS/SCI/Scopus). He is a member of several review committees of international journals.

Mohammad Husain is holding Ph.D. Degree in Computer Science and Engineering 2008. He did his M. Tech. in Computer Science 2005 and Bachelor of Engineering in Computer Science and Engineering in 1990. He has about 33 Years of Professional experience in the field of IT and Academics. His area of research is Information Retrieval, Web Mining, and Big Data. Currently he is working with Islamic University of Madinah, Saudi Arabia in the Faculty of Computer and Information Systems. He also has industrial exposure and has experience in Software Design and Development. He has published about 159 papers in different National/International Journals/proceedings. Dr. Husain has guided 14 Ph.D. and 11 M. Tech. students. He has authored a book on Principles of Programming Languages and contributed number of book chapters. He isalso an active member of various National/International Journals, Committees and Societies.

Arshad Ali joined Islamic University of Madinah in December 2014 as Assistant Professor and then promoted as Associate Professor in July 2018 and later in June 2023 he is promoted as full Professor of Information Technology. in Faculty of Computer and Information Systems, Islamic University of Madinah. He finishes his BSc in Mathematics and Statistics from University of Punjab, Lahore, Pakistan in 2000 and completed his Masters in Computer Sciences from Iqra University, Lahore, Pakistan. In 2005, he moved to Birmingham, UK for further studies. He joined Aston University, Birmingham, UK and obtained his MSc Telecommunication Technology in 2007. In 2007, he joined Geotechnical Group, Department of Engineering, and University of Cambridge as Research Ass. (2007- 2009). In 2009, he was awarded a PhD (2009- 2012) scholarship from the Lancaster University, UK and he awarded PhD in 2012.He worked on the UK-NEES project and designed communication system for live experimentation between UK Universities (Cambridge, Oxford and Bristol).