IAES International Journal of Artificial Intelligence (IJ-AI)
Vol. 15, No. 6, December 2026, pp. 5912~5924
ISSN: 2252-8938, DOI: 10.11591/ijai.v15.i6.pp5912-5924

Deep reinforcement learning for task semantic adaptive fault tolerance in fog network

Indumathi Srinivasa[1,2], Anitha Narasimha Murthy[1]
[1]Department of Computer Science and Engineering, BNM Institute of Technology Bangalore affiliated to Visvesvaraya Technological University, Belagavi, India
[2]Department of Artificial Intelligence and Machine Learning, BMS Institute of Technology and Management Bangalore affiliated to Visvesvaraya Technological University, Belagavi, India

Article Info

Article history:

Received Jul 29, 2025
Revised Aug 6, 2026
Accepted Aug 31, 2026

Keywords:

Deep reinforcement learning
Fault tolerance
Fog computing
Reliability
Task scheduling

ABSTRACT

Managing the faults effectively is very important in fog networks as it influences the reliability and quality of service (QoS) of the network. The existing fault management techniques for fog networks were not adaptive to task semantics. In earlier work, a deep learning fault management solution with a two stage process was proposed to solve this problem. In the first stage, faults were predicted using hosts state information. In the second stage, the fault prediction information was used for improving the multi criteria QoS based scheduling. But this two-stage process lacked management of faults after scheduling till the next schedule. Re-scheduling in batches affected the task response time. This work proposes a deep reinforcement learning fault management (DRL-FM) solution to address this problem. The proposed solution handles task migration till next batch of scheduling based on optimization of response time and fault occurrence. With this adaptive fault management, the proposed solution increases task reliability by 2%, fault detection accuracy 3%, reduces task response time by 8.1%, and task migration overhead by 29% compared to most recent works.

This is an open access article under the CC BY-SA license.



Corresponding Author:

Indumathi Srinivasa
Department of Computer Science and Engineering, BNM Institute of Technology Bangalore affiliated to Visvesvaraya Technological University
Belagavi, Karnataka, India
Email: indusrinivasa@gmail.com

1. INTRODUCTION

Internet of things (IoT) is making life smarter with sensors, applications becoming part of digital coexistence. Various innovative applications like smart healthcare, smart cities, and smart homes are revolutionizing the way of living. These applications are realized by processing large volumes of data generated by the IoT sensors. The computing resources necessary for processing large volumes of data was provided by clouds. Though the cloud model of processing data provides resources with high elasticity, it also introduces latency and congestion. Higher latency is a problem for many real time IoT based applications. Fog computing solves this problem by shifting the computing resources to the edge [1]. Fog's nature of distributed computing and close proximal resource allocation allows for real time decision making in IoT environment [2]. Fog computing is not elastic and thus have various challenges in terms of load and fault handling [3]. Ineffectiveness in managing the load and faults can trigger further faults and distort the quality of service (QoS) in the fog network. Various proactive fault tolerance techniques have been proposed to predict faults and schedule tasks in fog networks [4]. Faults were predicted based on resource consumption metrics, past failure rates and task execution metrics in these approaches. But these techniques did not

consider analyzing the correlation between various events arising from multiple fog layers. The inefficiency in fault prediction affects task scheduling/migration decisions significantly, creating poor QoS. Considering it, a deep learning based fault prediction based on fine grained events collected at various layers was proposed in earlier work [5]. Metrics collected from various fog layers are processed by deep learning long short term memory (LSTM) classifier to predict the fault probability of a node. Multi criteria task scheduling was done to schedule tasks to nodes based on predicted fault probability. Thus, there were two stages of fault prediction and task scheduling in that solution. But an important problem in that solution is that scheduling was done in batches. Once a task schedule was done, the faults occurring during the execution of tasks were not managed, leading to reduced reliability. The response time of waiting tasks till batch is scheduled increases. This also increased resource underutilization and migration overhead.

The main reason for these problems is a lack of fault management between batch schedules. This work proposes a deep reinforcement learning fault management (DRL-FM) solution to solve this problem. As part of the solution, two important contributions are made: i) a novel ensemble fault prediction technique is proposed which considers both multi-metrics collected at various fog layers and the history of faults to increase the accuracy of fault probability prediction; and ii) a DRL-FM is proposed to manage the tasks to improve the response time and increase the reliability of tasks. DRL-FM mitigates the impact of faults through effective task migration decisions. These decisions are found automatically through the reward-loss guidance function of DRL-FM. Due to effective migration, the reliability of tasks is improved without much impact on response time.

The rest of the paper is organized as follows. Section 2 presents a survey on existing fault management techniques in a fog computing environment. Section 3 presents the proposed DRL-FM technique to manage task executions in a fog environment. Section 4 presents the results of the DRL-FM solution in terms of reliability and QoS metrics. Finally, section 5 presents the concluding remarks and future work scope.

2. RELATED WORK

Choppara and Mangalampalli [6] used reinforcement learning for task scheduling. Though the reward function included reliability metrics, the approach was reactive and did not consider node wise fault risk projections. Though deep reinforcement improved the response time, the fault tolerance was reactive. Also, reliability metrics considered in task scheduling decision was only the number of fault tolerance. Metrics like mean time to failure (MTTF) and mean time to repair (MTTR) were not considered. Zhang *et al.* [7] predicted fog node failure using Markov chain steady state probabilities. Failure was predicted as whole without any fine-grained resolution. Failure prediction was based only on past failure history. Task scheduling was done to nodes with least probability of failure on two criteria of execution time and cost minimization. The influence of scheduling on further failure and refinement of schedule was not considered in this work. Park *et al.* [8] trained LSTM network to predict application faults in terms of application data. Though application faults are predicted, fault resiliency was not considered in this work. Kwon and Kim [9] predicted application faults. Authors proposed an iterative feature selection approach to select features with higher correlation to fault. Random forest classifier was then used to predict the faults. Faults were predicted but handling faults was not considered in this work. Mohammadi *et al.* [10] used past central processing unit (CPU) resource consumption data to train a Markov model. The model predicted fault probability of nodes. Tasks running in higher fault probability nodes are replicated in hot standby mode to provide fault resilience. But approach is resource intensive. Also, tasks were replicated without considering task semantics.

Tuli *et al.* [11] trained deep neural network to predict faults based on resource overconsumption. Deep neural network is trained with past interval resource consumption metrics to predict overconsumption. Temporal nature of consumption was not considered in this work. Task to node schedule matrix was provided by deep neural network. The approach was not scalable and impact of scheduling on further occurrence of fault is not considered. Theodoropoulos *et al.* [12] modeled faults as resource over consumption. Resource overconsumption is predicted for next interval based on past history of usage using recurrent neural networks. Tasks are migrated from nodes predicted overconsumption to other nodes. By this way proactive fault tolerance is done. But the problem in this approach is that even transient resource overconsumption is detected as failure and migration is done. This increases the migration cost. Huang [13] pre-allocated resources to each tasks using deep learning. The pre-allocation was done to reduce any resource contentions and failures due to it. But the system utilization is very less in this work. Also, the resource pre-allocation decision did not consider the node fault risks. Abawajy *et al.* [14] used task metrics and node energy consumption to predict faults. Tasks were migrated to lower fault probability nodes based on its deadline adherence ratio. When task is predicted to violate its deadline, it is migrated to lower resource consumption hosts to boost its response time. The scheduling decisions are modified frequently and the approach is not scalable for bigger networks.

Martinez *et al.* [15] proposed a proactive fault management technique using hot standby design. Optimal hot standby matrix is generated using mixed integer programming with objective of maximizing node utilization. Task semantics and fault occurrence rate were not considered in fault management decisions. The approach is highly resource intensive. Siyadatzadeh *et al.* [16] proposed a hot standby mode based proactive fault tolerance technique. Reinforcement learning is used for identification of tasks to be back up based on its workload and communication delay. Resource consumption impeding faults and task semantics were considered in backup decision making. Oma *et al.* [17] proposed a proactive fault management strategy based on hot standby provisioning. Locating hot standby nodes were made fast with hierarchical organization of nodes and selection of node based on least energy consumption. Tasks are backed up irrespective of task semantics and fault probability resulting in higher resource consumption.

Baskar and Periasamy [18] proposed a proactive fault management strategy by organizing the fog nodes based on residual energy. Tasks with higher priority are allocated to nodes with higher residual energy and tasks with lower priority are allocated to nodes with lower residual energy. Task QoS and effective of scheduling tasks on further residual energy drop were not considered in the scheduling decision. Tuli *et al.* [19] used generative adversarial network (GAN) to generate a fault schedule backing up task. Node fault was predicted based on resource consumption metrics. Task semantics and occurrence of further faults were not considered in the task scheduling decision. Mudassar *et al.* [20] proposed a hot standby based proactive fault management technique. Fault is modeled as resource over consumption at nodes and when it is detected, the tasks are proactively migrated to another node. Temporal characteristics of resource consumption were not considered in task migration decision. This will result in frequent task migrations. Task semantics was also not considered in scheduling decision.

Most of existing fault tolerance techniques predicts faults either based on resource consumption or on history of faults. Approaches using various diverse metrics across fog layers, task metrics and history of faults to increase the fault prediction accuracy were not considered in existing works. For proactive fault tolerance, tasks were scheduled to nodes only based on node's resource metrics without considering task semantics and fault prediction probabilities of the node. Considering this problem, Indumathi and Anitha [5] proposed a deep learning fault management technique. Fault prediction considered more fog layer parameters in addition to resource consumption metrics. This increased the fault prediction accuracy. Tasks were scheduled based on multi criteria of task semantics, resource consumption at hosts, and service level agreements. But the solution scheduled tasks in batches. Though fault was considered as criteria in forming a task schedule, the impact of faults during execution of schedule was not considered. This work rectifies this problem by applying deep reinforcement learning (DRL) [21].

3. METHOD

The architecture of the proposed DRL-FM is given in Figure 1 and steps in the proposed solution are given in Figure 2. The DRL-FM extends [5] in two functional ways: i) fault prediction and ii) DRL based task maintenance and retains the multi criteria optimized task scheduling. On arrival of batch of jobs, multi criteria optimized task scheduling is invoked to find the task schedule matrix [5]. The tasks are scheduled to fog nodes as per the task schedule matrix. Before arrival of necessary tasks for a new batch, DRL task maintenance checks the fault probabilities and task execution metrics. Based on it, invokes task migration decisions with goal of maximizing reliability, improving task QoS and reducing the fault probabilities. Differing from [5], faults are predicted using ensemble of temporal resource characteristics and fault occurrence history. Fault management during execution of task batch is handled by DRL with suitable task migration decisions done based on reward function. The details of each of the two functions of fault prediction and DRL based task rescheduling are given as follows.

3.1. Fault prediction

A fog node can be in two states of normal (0) and faulty (1). The state transition matrix P for a fog node over next time interval (Δt) can be given as (1).

$$P(\Delta t) = \begin{pmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{pmatrix} \quad (1)$$

The state transition probability of node being normal from previous state of normal is given as (2).

$$p_{00}(\Delta t) = P\{X(t+\Delta t) = 0 \,|X(t) = 0\} = 1 - \lim_{\Delta t \to 0} \left(\int_0^{\Delta t} \lambda e^{-\lambda t} dt \right) = 1 - \lambda \Delta t \quad (2)$$

Where λ is death rate of fog node. The probability of transition from faulty state to normal state is given as (3).

$$p_{10}(\Delta t) = 1 - \mu\Delta t \tag{3}$$

Where μ is the repair rate. In terms of p_{00}, the other state transition probabilities are given as (4) and (5).

$$p_{01} = 1 - p_{00} \tag{4}$$

$$p_{11} = 1 - p_{10} \tag{5}$$

The (1) can be rewritten as (6).

$$P(\Delta t) = \begin{pmatrix} 1 - \lambda\Delta t & \lambda\Delta t \\ \mu\Delta t & 1 - \mu\Delta t \end{pmatrix} \tag{6}$$

The state transition probabilities of fog node is flatted to feature vector and passed to LSTM to predict the fault probability of node. The fault is predicted using the resource / task metrics as in [5]. From the fault prediction probability got from resource/task metrics (Fr) and state transition probabilities (Fs), ensemble is done to get final fault prediction probability (Fp) as (7).

$$Fp = w_1 Fr + w_2 Fs \tag{7}$$

Where w_1, w_2 are set to value in such a way that $w_1 + w_2 = 1$.

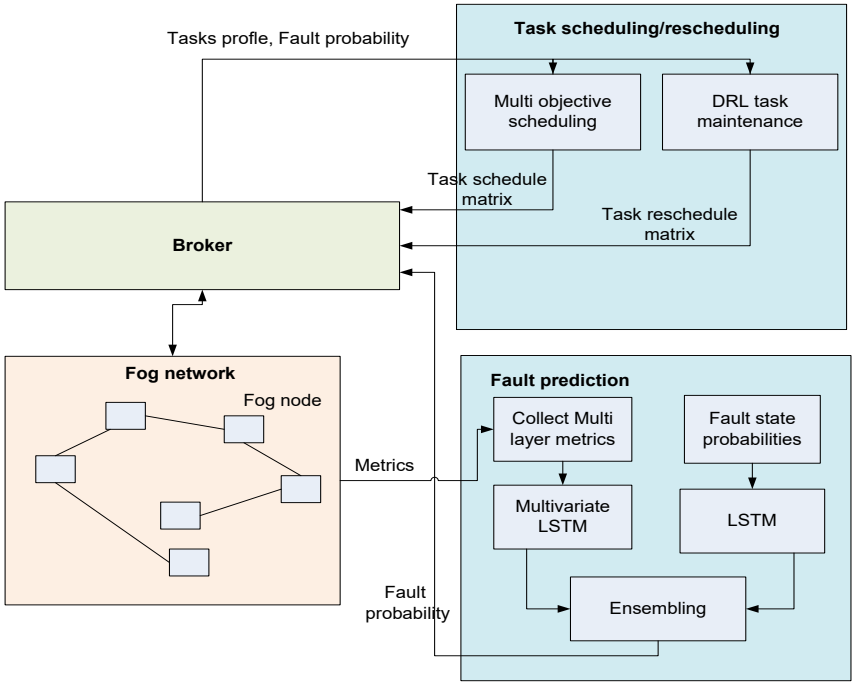

Figure 1. Architecture of DRL-FM

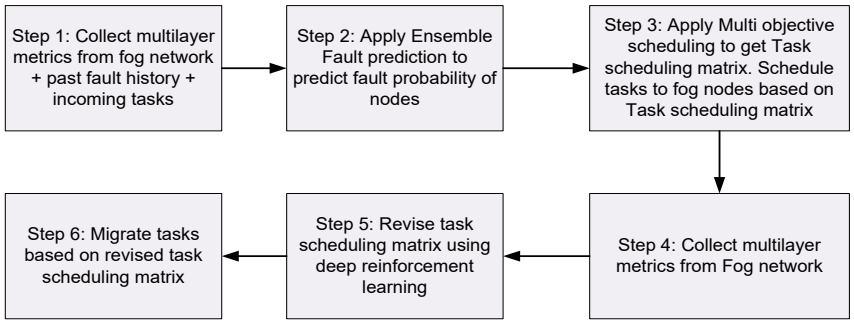

Figure 2. Structure of DRL FM policy network

3.2. Deep reinforcement learning task maintenance

The multi criteria optimized task scheduling proposed in [5] is executed on arrival of task batch to schedule the tasks to the node. Once the task batch is scheduled DRL task maintenance takes over. The task migration decisions are based on DRL with goal of improving reliability, task QoS and reducing further fault probabilities.

DRL is selected for this task due to its effectiveness in dealing with Markov decision process [22] over large state and action space. DRL finds the optimal policy Π^* which maximizes the long term cumulative reward. In DRL, an agent (scheduler) makes continuous interaction with environment (fog network) through sequence of decisions. At a scheduling step n the long term cumulative reward function V under the policy Π is calculated as (8).

$$V_{\Pi} = E_{\Pi}(r^n + \Upsilon r^{n+1} + \Upsilon^2 r^{n+1} + \cdots] \tag{8}$$

Where r is the reward achieved at each time step, E is the expectation operator, $\Upsilon\ \varepsilon[0,1]$ is the weight factor showing importance of future rewards to the cumulative return.

On each interaction, the agent (scheduler) makes action. This action is based on the current observed state s^n. The action updates the state to a new state and provides a reward r. The action-state value also called as Q function for a state action pair is represented as (9).

$$Q_{\Pi}(s,a;\theta) = E_{\Pi}[\sum_{k=n}^{\infty} \Upsilon^{k-n} r^k | s^n = s, a^n = a] \tag{9}$$

In (9) θ is the deep neural network parameter. In terms of Q function, the reward and the optimal policy can be rewritten as (10) and (11).

$$V_*(s) = max_{\Pi} Q_{\Pi}(s,a) \tag{10}$$

$$\Pi_* = arg_{\Pi} max Q_{\Pi}(s,a) \tag{11}$$

DRL attempts to search for optimal policy which maximizes the rewards. There are two strategies to achieve optimal policy. Value and policy gradient based. In value-based function, the Q learn function is learnt and the actions which maximizes $arg_{\Pi} max_{a \in A}(s,a)$. In policy gradient function, the best action is selected from the distribution of action probabilities obtained by softmax function as (12).

$$\Pi_{\theta}(s,a) = \frac{e^{\phi(s,a)^T \theta}}{\sum_{k=1}^{K} e^{\phi(s,a_k)^T \theta'}} \tag{12}$$

Where $\phi(s,a)$ is the input feature vector. Policy gradient is adopted in this work due to its effectiveness in higher dimensional action space and its stochastic policy representation.

Before applying DRL to the problem of task migration, the state space, action space and state transitions are identified. The state space has 4 parts: the task schedule matrix (TS), the fault probability vector (FP), the node occupancy vector (NO), and the task response time vector (TR). The state of system S is represented as (13).

$$S = \{s | s = (TS, FP, NO, TR)\} \tag{13}$$

The task schedule matrix is the initial task schedule provided [5]. It is a binary matrix with 1 in position i,j representing task i is allocated to node j. The node occupancy vector provides the resource utilization percentage of each node. The task response time vector represents the expected response time for each task. The action space includes two actions: valid and invalid. Valid action represents the migration of task from a high fault probability node to a low fault probability node. Invalid action represents no task is selected for migration. At each scheduling step n, the action is represented as (14).

$$A = \{A_e | A_e = (T_m, N_p)|^{m \in \{-1,1,2,\ldots M\}}_{p \in \{-1,1,2,\ldots P\}}\} \tag{14}$$

A_{ψ} is the invalid action represented as (-1,-1) indicating no task is selected for migration. A_{φ} is the valid action indicating task m is migrated to node p. At each scheduling step, DRL makes a decision based on system state. If valid action A_{φ} is selected, the suitable task (from node having higher fault probability) is

selected for migration to another node (node having lower fault probability). If invalid action A_ψ is selected, time step proceeds to next scheduling interval. The state moves to next based on its past and the action (s, a). The aim of DRL scheduler is to find optimal policy $\Pi = p(a, s)$. This movement to optimal policy by guided by the reward (r) as in (15).

$$r = \begin{cases} mcr, & a \epsilon A_\varphi \\ 0, & a \epsilon A_\psi \end{cases} \quad (15)$$

Where mcr is the multi criteria reward calculated based on overall task satisfaction (ts) and fog network reliability (fr). Task satisfaction is calculated in terms of average of ratio of response time to the expected latency requirement of each task (16). Fog network reliability [23] is calculated in terms of ratio of number of fog nodes with fault probability less than average fault probability to the total number of fog nodes.

$$mcr = ts + fr \quad (16)$$

DRL-FM uses Monte Carlo policy gradient approach to update policy parameters θ based on expected return over the trajectories of state, action and reward $\tau = (s^0, a^0, r^1, s^1, a^1, r^2, \ldots s^T, a^T, r^{T+1}\}$. Policy gradient at step t in the trajectory is transformed to (17).

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log \pi_\theta(s^n, a^n) G^n] \quad (17)$$

Based on the gradient descent in (17), the parameter θ is updated as (18).

$$\theta^{n+1} = \theta^n + \alpha G^n \nabla_\theta log \pi_\theta(s^n, a^n) \quad (18)$$

Where α is the learning rate. The log probability of selecting the action a^n in the state s^n is increased when G^n is positive, else it is reduced. By this way, gradient descent guide towards optimal policy. A DRL-FM policy network (Figure 2) is created, which takes the system state $s^n = (TS, FP, NO, TR)$ and provides the action probability as output. This DRL-FM policy network is called at every time step with updated state to get the action probability. If the action probability is (-1,-1), no migration is done and no reward is given. In case of valid action, the task from high fault probability node (above threshold say 0.7) is moved to low fault probability node. The state, action and rewards are used to compute the cumulative reward for further training. The DRL-FM policy network is optimized for best network parameters θ using a loss function as in (19).

$$L_\theta = \frac{1}{T} \sum_{n=1}^{T} G^n C^n \quad (19)$$

Where T is steps so far, G^n is the expected cumulative reward, and C^n is the cross entropy. The cross entropy is calculated as difference between the predicted action $\left(\pi_\theta(a, s)\right)$ and target action as in (20).

$$C^n = -y_a \log\left(\pi_\theta(s^n, a^n)\right) \quad (20)$$

DRL-FM policy network is trained to select the best actions which improve reliability, task QoS and reducing further fault probabilities. The trained DRL-FM policy network is then used for task migration decision till the next batch of task scheduling is initiated [5].

4. RESULTS AND DISCUSSION

4.1. Results

The fault tolerant effectiveness of the proposed solution was tested through simulation in yet another fog simulator (YAFS) [24] environment. YAFS is the light weight simulator based on discrete even simulation model for the fog computing applications. Two different datasets were used in this work. For simulating the tasks and task profiles, DeFog fog computing benchmark [25] was used. The simulation was conducted for 200 minutes with 100 epoch intervals.

The simulation configuration for reproducibility of the results is summarized in Table 1. Historical data collected by running gradient based optimization strategy using back-propagation of gradients with

respect to input (GOBI) approach [26] is used for testing the fault detection performance. For training of steady state fault probabilities based fault prediction Backblaze dataset [27] was used. The performance was compared in terms of fault detection effectiveness, task scheduling effectiveness, resource consumption effectiveness and fault handling effectiveness with metrics given Table 2.

Table 1. Simulation environment and experimental parameters

Category	Parameter	Value
Simulator	Simulation platform	YAFS
	Programming language	Python 3.10
	Operating system	Ubuntu 22.04 LTS (64-bit)
	Deep learning framework	TensorFlow 2.15/Keras
	Simulator model	Discrete-event fog computing Simulation
	Task dataset	DeFog Benchmark
	Number of fog nodes	100 (default); scalability tested from 20–200 nodes
	Simulation duration	200 minutes
	Scheduling interval	100 epochs
	Task arrival model	Poisson arrival process
	Task categories	High-priority and low-priority tasks
	CPU utilization threshold	80%
	Fault probability threshold	0.70
	Node state model	Two-state Markov model (healthy/faulty)
LSTM configuration	Hidden units	128
	Layers	2
	Sequence length	20 time steps
	Batch size	64
	Optimizer	Adam
	Learning rate	0.001
	Epochs	100
	Loss function	Binary cross-entropy
	LSTM weight (α)	0.60
	Markov weight (β)	0.40
Reinforcement configuration	RL algorithm	Policy Gradient (REINFORCE)
	Policy network hidden layers	128 → 64 neurons
	Activation function	ReLU
	Output activation	Softmax
	Discount factor (γ)	0.99
	Policy learning rate	0.0003
	Reward function	Weighted combination of task satisfaction and fog reliability
Migration policy	Fault probability threshold	0.70
	Migration action	High-risk node → Low-risk node
	Reward weights	Task Satisfaction = 0.5, Reliability = 0.5
Training configuration	Random seed	42
	Training/validation/test split	70%/15%/15%
	Number of independent runs	10

Table 2. Metrics used for comparison

Fault detection effectiveness	Accuracy, precision, recall, and F1-score
Task scheduling effectiveness	Response time and task reliability
Resource consumption effectiveness	Migration overhead, resource utilization
Fault handling effectiveness	MTTF, MTTR, sensitivity analysis

The performance of fault detection in the proposed approach is compared against the DeepFT approach by Tuli *et al.* [11], the Markov chain based approach proposed by Zhang *et al.* [7], a Markov-based approach proposed by Mohammadi *et al.* [10], and earlier work deep learning based fault tolerance [5]. The fault detection effectiveness was measured across the solutions. The results are given in Table 3.

Table 3. Comparison of fault detection performance

Solution	Accuracy	Precision	Recall	F1-score
DRL-FM	0.98	0.93	0.94	0.93
Deep learning based	0.96	0.89	0.92	0.91
Tuli *et al.* [11]	0.94	0.86	0.90	0.88
Zhang *et al.* [7]	0.87	0.78	0.84	0.81
Mohammadi *et al.* [10]	0.92	0.85	0.89	0.87

The accuracy of DRL-FM is higher by 2% compared to our earlier deep learning based work, 4% higher compared to Tuli *et al.* [11], 11% higher compared to Zhang *et al.* [7], and 6% higher compared to Mohammadi *et al.* [10]. The fault detection performance has increased in DRL-FM due to ensemble of resource/task metrics based fault detection with steady state fault probability based detection. These two approaches complimented each other in fault detected probabilities. While resource/task metrics based fault detection covered all software induced faults and application faults, the steady state fault probabilities covered hardware faults. Due to this wide fault coverage, the accuracy of fault detection increased and the false positives too reduced (as indicated by higher recall value).

The average response time for two task profile of high and low priorities is measured across the solutions and the result is given in Figure 3. The response time decreased by at least 4% for higher priority tasks and 6% for low priority tasks in the proposed DRL-FM. The response time reduction was due to its consideration in both task scheduling and task migration decision in the DRL-FM. The response time improvement increased the reward in DRL-FM due to which the policy decisions of DRL-FM policy network was optimized to reduce the response time.

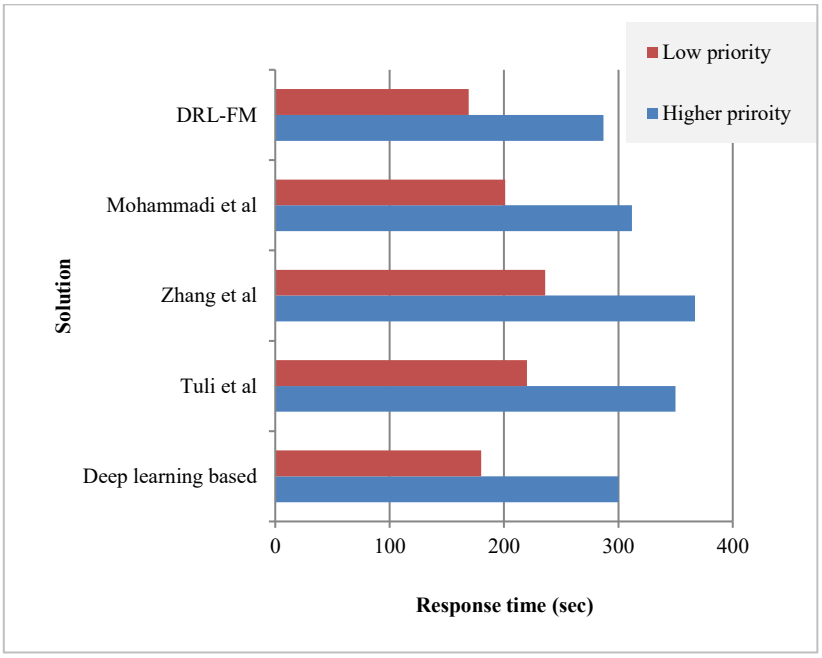

Figure 3. Comparison of response time

Three parameters of MTTF, MTTR, and task reliability ratio (TRR) are measured and given in Table 4. These parameters are indicators of fault tolerance performance. While parameters MTTR and MTTF are direct, the TRR is calculated as (21).

$$\text{TRR} = 1 - \frac{\sum \text{task failed}}{\sum \text{task scheduled}} \tag{21}$$

Table 4. Comparison of fault tolerance parameters

Solution	MTTF	MTTR	TRR
DRL-FM	16.1	6.1	0.97
Deep learning based	15.3	6.2	0.95
Tuli *et al.* [11]	13.45	7.7	0.93
Zhang *et al.* [7]	9.5	8.3	0.90
Mohammadi *et al.* [10]	11.2	8.9	0.91

An effective fault tolerance strategy should have higher MTTF, lower MTTR, and higher TRR. Though there is not much difference in MTTR, the proposed DRL-FM had better MTTF, and TRR compared to existing works. MTTF and TRR have increased in proposed DRL-FM due to avoidance of loading

nodes with higher fault probability and shifting load from the node when its fault probability increases. The reliability (in terms of TRR) was measured by increasing the task arrival rate to check the scalability of the solution and the results are given in Figure 4.

The average reliability in proposed DRL-FM is at least 2% higher compared to existing works. Even when task rate is higher, the reliability is fairly above 94%. The increased reliability is due to effective task scheduling and DRL based task migration in the proposed DRL-FM. Overhead analysis is conducted in terms of average decision time, number of migrations per 1,000 tasks and resource utilization improvement. The proposed DRL-FM is compared against the deep learning based solution and the results are given in Table 5.

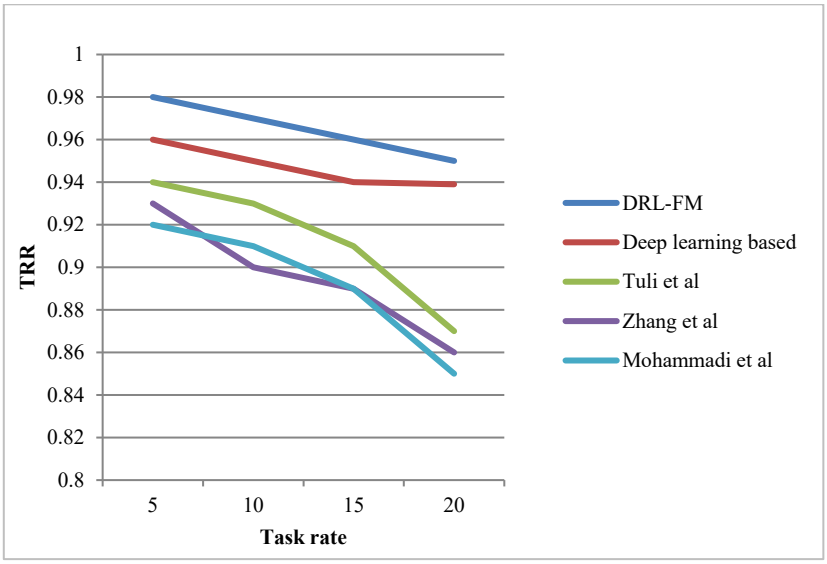

Figure 4. Reliability vs. task rate

Table 5. Overhead comparison

Metric	Deep learning based	DRL-FM
Average decision time (ms)	12	18
Task migrations/1000 tasks	45	32
Average node utilization	74	81

The DRL-FM takes slightly longer (18 ms) than the previous deep learning–based solution (12 ms), because it performs both the policy network inference and ensemble probability computation. Frequent task migrations increases communication delay and temporary resource wastage. Therefore, it is essential to check whether DRL-FM causes unnecessary migrations. DRL-FM learns an optimal migration policy that triggers migration only when the fault probability exceeds a threshold (0.7) and the expected improvement in reliability outweighs the migration cost. Better fault-aware scheduling and fewer unnecessary migrations help DRL-FM maintain higher average node utilization. This metric reflects how effectively computing resources are used without overloading any single node. This improvement shows that DRL-FM balances the load more effectively across nodes, avoiding both underutilization and overloading. To ensure fair comparison, all models were simulated under the same environment using the YAFS simulator with identical task profiles and node configurations. The results were measured using three core performance indicators: TRR, fault detection accuracy, and average task response time. The results are given in Table 6.

Table 6. Comparison with other RL works

Solution	Fault detection accuracy	TRR	Average response time (s)
ReLIEF [16]	0.96	0.95	7.0
PreGAN [19]	0.95	0.94	7.4
DRL-FM	0.98	0.97	6.8

From Table 5, it can be seen that the DRL-FM outperforms both ReLIEF and PreGAN across all three evaluation parameters. The improvement in fault detection accuracy (2–3%) is primarily due to the ensemble learning approach that combines temporal resource metrics with steady-state fault probabilities. This hybrid modeling captures a wider spectrum of both transient and persistent faults. The improvement in task reliability (TRR) is due to adaptive migration in DRL-FM. Different from PreGAN, which triggers migrations solely based on predicted failures, DRL-FM considers multiple QoS-driven factors such as response time, fault probability, and task semantics in its reward function.

Additionally, the average response time is reduced by nearly 9% compared to ReLIEF and 12% compared to PreGAN, highlighting that DRL-FM not only improves fault resilience but also enhances runtime performance efficiency. The performance improvement shows that the proposed policy gradient–based DRL strategy generalizes better under varying fog loads and fault intensities compared to adversarial or value-based RL strategies. An ablation study was conducted to analyze the contribution of each component of proposed DRL-FM. Three model variants were developed and evaluated under identical fog simulation conditions using the YAFS environment and the same task arrival distributions: the model variations for the ablation study are given in Table 7. Each model was evaluated using three metrics: fault detection accuracy, TRR, and average response time for the overall workload. The results are given in Table 8.

Table 7. Ablation cases

Model	Description
A	This variant uses only resource/task metrics–based fault prediction (similar to the earlier deep learning model [5]) without combining it with steady-state fault probability estimation.
B	This version performs ensemble-based fault prediction but does not use reinforcement learning for task migration. The tasks are scheduled once in each batch with no mid-batch migration
C	Includes both ensemble fault prediction and DRL-based task maintenance, representing the complete architecture proposed in this work.

Table 8. Ablation study results

Model variant	Accuracy	TRR	Avg response time(s)
A	0.95	0.93	7.8
B	0.96	0.94	7.2
C	0.98	0.97	6.8

Comparing model A and model B, it can be seen that the ensemble of LSTM-based temporal features and Markov-based steady-state fault probabilities enhance detection accuracy and increase the task reliability by 1%. The ensemble helps capture transient software faults and persistent hardware degradation, thereby reducing false negatives. Comparing model B and model C shows that the inclusion of the DRL policy network significantly improves performance. Accuracy increases by 3% and response time drops by 0.4 s. This is because the DRL agent dynamically migrates tasks away from nodes exhibiting rising fault probabilities, maintaining higher reliability without batch-level waiting. The combination of ensemble fault prediction and DRL-based migration in the full DRL-FM framework yields the best results, with improvements of up to 3% increase in reliability and 13% reduction in average response time compared to partial variants. These findings show that fault prediction provides early warning, while DRL ensures adaptive response to runtime variations. The ablation results demonstrate that the superior performance of DRL-FM is not due to a single enhancement but rather to the joint integration of predictive and adaptive mechanisms. The ensemble predictor ensures accurate estimation of node health, and the reinforcement learning component effectively translates these predictions into proactive scheduling decisions. In the proposed DRL-FM framework, task migration is initiated from a node when the predicted fault probability of a fog node exceeds a predefined threshold. To analyze the sensitivity of system performance to this threshold, the threshold is varied from 0.5 to 0.8 and three metrics of TRR, average response time, and number of migrations per 1000 tasks was measured.

From the results in Table 9, it can be seen that the threshold value of 0.7 achieves the best trade-off between reliability and overhead, yielding the highest TRR with a moderate number of migrations and the lowest average response time. The scalability of the proposed DRL-FM was evaluated by varying the number of fog nodes from 20 to 200 but still keeping the task arrival distribution unchanged. The results are given in Table 10.

Table 9. Sensitivity analysis

Fault probability threshold	TRR	Average response time(s)	Task migrations/1,000 tasks
0.5	0.98	7.1	54
0.6	0.97	6.9	41
0.7	0.97	6.8	32
0.8	0.95	7.3	18

Table 10. Scalability analysis

Number of fog nodes	TRR	Average response time (s)
20	0.95	7.4
50	0.96	7
100	0.97	6.8
150	0.97	6.7
200	0.96	6.9

As the number of fog nodes increases from 20 to 100, task reliability increases due to higher availability of alternative low-fault-probability nodes for task migration. The TRR remains consistently above 95% even for large-scale deployments. It shows that the learned policy generalizes well to larger state and action spaces.

4.2. Discussion

The results demonstrate that the proposed DRL-FM model significantly enhances both fault detection accuracy and fault tolerance compared to existing Fog computing strategies. The improvements achieved in the proposed solution are summarized in Table 11. The improvement is due to the ensemble-based fault prediction, which integrates both temporal resource/task metrics and steady-state failure probabilities. The complementary nature of these two components helps the model detect both short-term and long-term faults more precisely, reducing false positives and false negatives. The proposed framework also shows substantial improvement in QoS-related metrics, particularly response time and reliability. The improvement is due to adaptive task migration during execution, instead of relying solely on static, batch-level scheduling. This enables the model to react immediately to variations in node reliability, preventing task delays caused by resource degradation. The higher MTTF further confirms the model's resilience. Scalability and resource efficiency are additional strengths of the DRL-FM model. The overhead analysis (Table 5) shows that even though the DRL-FM introduces a small computational delay, it significantly reduces migration overhead by 29% and improves node utilization by 7%. This indicates that the proposed policy gradient–based DRL formulation generalizes more effectively under varying fault conditions than adversarial or value-based models. Together, these results validate that DRL-FM achieves a robust balance between accuracy, reliability, and computational efficiency, making it a scalable and practical solution for real-world fog computing environments.

Table 11. Improvements achieved in proposed solution

Effectiveness category	Metric	Best existing method (value)	Proposed DRL-FM value (% improvement)
Fault detection effectiveness	Accuracy	Deep learning [5] (0.96)	0.98 (2.1%)
	Precision	Deep learning [5] (0.89)	0.93 (4.5%)
	Recall	Deep learning [5] (0.92)	0.94 (2.2%)
	F1-score	Deep learning [5] (0.91)	0.93 (2.2%)
Task scheduling effectiveness	Average response time	PreGAN [19] (7.4 s)	6.8 s (8.1%)
	TRR	ReLIEF [16] (0.95)	0.97 (2.1%)
Resource consumption effectiveness	Migration overhead	Deep learning [5] (45 /1000 tasks)	32 /1000 tasks (29%)
	Average node utilization	Deep learning [5] (74%)	81% (9.5%)
Fault handling effectiveness	MTTF	Deep learning [5] (15.3)	16.1 (5.2%)
	MTTR	Deep learning [5] (6.2)	6.1 (1.6%)
	Reliability under high load	0.94	0.96 (2%)

5. CONCLUSION

This work proposed a DRL-based fault tolerance solution called DRL-FM. The solution improved the earlier work in terms of two functionalities of fault prediction and task migration. An ensemble of resource/task migration metrics-based fault prediction with steady state fault probabilities-based fault

prediction is done to increase fault prediction accuracy. The proposed DRL-FM increased the fault prediction accuracy to 98%, which is at least 2% higher compared to existing works. DRL-FM fine-tuned its policy network for task migration with goal of improving task response, reliability, and fault occurrences. Due to this the response time dropped by 4%, reliability increased by 2%, migration overhead reduced by 29%, and MTTF dropped by 1% in the proposed solution compared to existing works.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Indumathi Srinivasa	✓	✓	✓	✓	✓	✓		✓	✓	✓			✓	
Anitha Narasimha Murthy	✓	✓		✓		✓		✓	✓	✓	✓	✓		

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

The data that support the findings of this study are available on request from the corresponding author, [IS].

REFERENCES

[1] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *2015 Workshop on Mobile Big Data*, Jun. 2015, pp. 37–42, doi: 10.1145/2757384.2757397.
[2] P. Habibi, M. Farhoudi, S. Kazemian, S. Khorsandi, and A. L. -Garcia, "Fog computing: a comprehensive architectural survey," *IEEE Access*, vol. 8, pp. 69105–69133, 2020, doi: 10.1109/ACCESS.2020.2983253.
[3] A. Sharif, M. Nickray, and A. Shahidinejad, "Fault-tolerant with load balancing scheduling in a fog-based IoT application," *IET Communications*, vol. 14, no. 16, pp. 2646–2657, Oct. 2020, doi: 10.1049/iet-com.2020.0080.
[4] A. Reyana, S. Kautish, K. A. Alnowibet, H. M. Zawbaa, and A. W. Mohamed, "Opportunities of IoT in fog computing for high fault tolerance and sustainable energy optimization," *Sustainability*, vol. 15, no. 11, May 2023, doi: 10.3390/su15118702.
[5] S. Indumathi and N. Anitha, "Deep learning based fault tolerance adaptive to task semantics in fog computing network," *International Journal of Intelligent Engineering and Systems*, vol. 17, no. 6, pp. 905–914, Dec. 2024, doi: 10.22266/ijies2024.1231.68.
[6] P. Choppara and S. S. Mangalampalli, "Resource adaptive automated task scheduling using deep deterministic policy gradient in fog computing," *IEEE Access*, vol. 13, pp. 25969–25994, 2025, doi: 10.1109/ACCESS.2025.3539606.
[7] P. Zhang *et al.*, "A fault-tolerant model for performance optimization of a fog computing system," *IEEE Internet of Things Journal*, vol. 9, no. 3, pp. 1725–1736, Feb. 2022, doi: 10.1109/JIOT.2021.3088417.
[8] D. Park, S. Kim, Y. An, and J.-Y. Jung, "LiReD: a light-weight real-time fault detection system for edge computing using LSTM recurrent neural networks," *Sensors*, vol. 18, no. 7, Jun. 2018, doi: 10.3390/s18072110.
[9] J.-H. Kwon and E.-J. Kim, "Failure prediction model using iterative feature selection for industrial internet of things," *Symmetry*, vol. 12, no. 3, p. 454, Mar. 2020, doi: 10.3390/sym12030454.
[10] V. Mohammadi, A. M. Rahmani, A. Darwesh, and A. Sahafi, "Fault tolerance in fog-based social internet of things," *Knowledge-Based Systems*, vol. 265, Apr. 2023, doi: 10.1016/j.knosys.2023.110376.
[11] S. Tuli, G. Casale, L. Cherkasova, and N. R. Jennings, "DeepFT: fault-tolerant edge computing using a self-supervised deep surrogate model," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, May 2023, pp. 1–10, doi: 10.1109/INFOCOM53939.2023.10229049.
[12] T. Theodoropoulos, J. Violos, S. Tsanakas, A. Leivadeas, K. Tserpes, and T. Varvarigou, "Intelligent proactive fault tolerance at the edge through resource usage prediction," *ITU Journal on Future and Evolving Technologies*, vol. 3, no. 3, pp. 761–778, Dec. 2022, doi: 10.52953/EHJP3291.

[13] Z. Huang, "Task scheduling in fog computing systems using deep learning," *Egyptian Informatics Journal*, vol. 31, Sep. 2025, doi: 10.1016/j.eij.2025.100757.
[14] J. Abawajy, S. Ghanavati, and D. Izadi, "Enhancing dependability of fog computing using learning-based task scheduling," *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 6, pp. 5849–5860, Nov. 2025, doi: 10.1109/TDSC.2025.3566395.
[15] I. Martinez, A. S. Hafid, and M. Gendreau, "Robust and fault-tolerant fog design and dimensioning for reliable operation," *IEEE Internet of Things Journal*, vol. 9, no. 19, pp. 18280–18292, Oct. 2022, doi: 10.1109/JIOT.2022.3157557.
[16] R. Siyadatzadeh *et al.*, "ReLIEF: A reinforcement-learning-based real-time task assignment strategy in emerging fault-tolerant fog computing," *IEEE Internet of Things Journal*, vol. 10, no. 12, pp. 10752–10763, Jun. 2023, doi: 10.1109/JIOT.2023.3240007.
[17] R. Oma, S. Nakamura, T. Enokido, and M. Takizawa, "Fault-tolerant strategies in the tree-based fog computing model," *International Journal of Distributed Systems and Technologies*, vol. 11, no. 4, pp. 72–91, Oct. 2020, doi: 10.4018/IJDST.2020100105.
[18] P. Baskar and P. Periasamy, "Minimum-cost-based neighbour node discovery scheme for fault tolerance under IoT-fog networks," *Future Internet*, vol. 16, no. 4, Apr. 2024, doi: 10.3390/fi16040123.
[19] S. Tuli, G. Casale, and N. R. Jennings, "PreGAN: Preemptive migration prediction network for proactive fault-tolerant edge computing," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, May 2022, pp. 670–679, doi: 10.1109/INFOCOM48880.2022.9796778.
[20] M. Mudassar, Y. Zhai, L. Liao, and J. Shen, "A decentralized latency-aware task allocation and group formation approach with fault tolerance for IoT applications," *IEEE Access*, vol. 8, pp. 49212–49223, 2020, doi: 10.1109/ACCESS.2020.2979939.
[21] M. Krichen, "Deep reinforcement learning," in *2023 14th International Conference on Computing Communication and Networking Technologies*, Jul. 2023, pp. 1–7, doi: 10.1109/ICCCNT56998.2023.10306453.
[22] D. T. Hoang, N. V. Huynh, D. N. Nguyen, E. Hossain, and D. Niyato, "Markov decision process and reinforcement learning," in *Deep Reinforcement Learning for Wireless Communications and Networking*, Hoboken, United States: Wiley, 2023, pp. 25–36, doi: 10.1002/9781119873747.ch2.
[23] J. Liu, S. Wang, A. Zhou, S. A. P. Kumar, F. Yang, and R. Buyya, "Using proactive fault-tolerance approach to enhance cloud service reliability," *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 1191–1202, Oct. 2018, doi: 10.1109/TCC.2016.2567392.
[24] I. Lera, C. Guerrero, and C. Juiz, "YAFS: a simulator for IoT scenarios in fog computing," *IEEE Access*, vol. 7, pp. 91745–91758, 2019, doi: 10.1109/ACCESS.2019.2927895.
[25] J. McChesney, N. Wang, A. Tanwer, E. D. Lara, and B. Varghese, "DeFog: fog computing benchmarks," in *4th ACM/IEEE Symposium on Edge Computing*, Nov. 2019, pp. 47–58, doi: 10.1145/3318216.3363299.
[26] S. Tuli, S. R. Poojara, S. N. Srirama, G. Casale, and N. R. Jennings, "COSCO: container orchestration using co-simulation and gradient-based optimization for fog computing environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 1, pp. 101–116, Jan. 2022, doi: 10.1109/TPDS.2021.3087349.
[27] Backblaze, "Backblaze drive stats: hard drive reliability test data," *backblaze.com*. 2014. Accessed: Oct 30, 2025. [Online]. Available: https://www.backblaze.com/cloud-storage/resources/hard-drive-test-data

BIOGRAPHIES OF AUTHORS



Indumathi Srinivasa is a part-time research scholar in the Department of Computer Science and Engineering, BNM Institute of Technology affiliated to Visvesvaraya Technological University, Belagavi, India. She is currently working as an assistant professor in the Department of Artificial Intelligence and Machine Learning, BMS Institute of Technology and Management, Bengaluru. She completed her B.E. in Information Science and Engineering and M.Tech. in Computer Science and Engineering from Visvesvaraya Technological University. She is currently pursuing her Ph.D. from Visvesvaraya Technological University. She has published around 5 research papers in conferences and journals. Her areas of research interest include cloud computing, fog computing, and edge computing. She can be contacted at email: indusrinivasa@gmail.com.



Anitha Narasimha Murthy currently working as a professor in Computer Science and Engineering, BNM Institute of Technology, Bengaluru. She completed her B.E. in Computer Science and Engineering from Bangalore University, M.Tech. in Information Technology from Bangalore University, and Ph.D. from Visvesvaraya Technological University. She has published around 30 research papers, and her area of research interest are artificial intelligence, machine learning, big data analytics, and data mining. She can be contacted at email: anitha.mhp@gmail.com.