Chapter 5

Energy-Aware and QoS-Enhanced Routing with Node Fault Prediction for Consumer IoT Networks Using ML Frameworks

Over the years, IoT has expanded its reach across various industries, from healthcare and manufacturing to transportation and agriculture [129, 15]. A notable area of growth is consumer IoT (CIoT), which focuses on enhancing daily life through connected devices such as smart home appliances, wearables, and personal assistants. Chapter 3 focused on static IoT networks where device positions and network topology remain fixed over time. In contrast, this chapter addresses dynamic, time-varying IoT networks, where device mobility causes frequent link changes. The proposed approach extends the fault prediction framework to mobile scenarios and introduces an adaptive routing strategy combining Q-learning, deep learning, and fuzzy logic to maintain energy efficiency and QoS under mobility-induced challenges. As shown in Figure 5.1, the primary goal of CIoT is to enable these devices to autonomously collect and transmit data, often

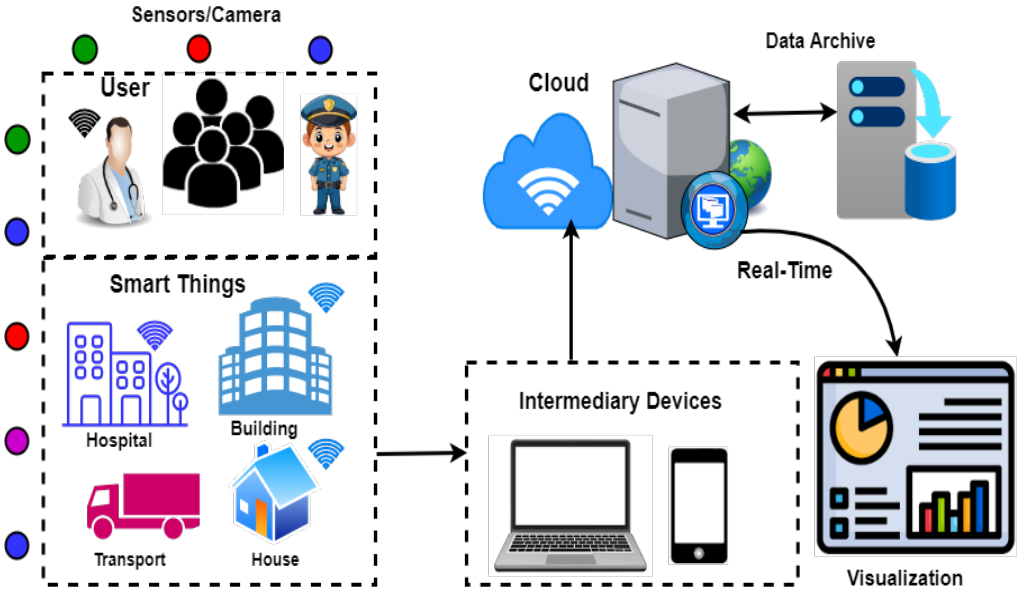


Figure 5.1: IoT ecosystem: connecting devices for diverse consumer applications

in real-time, to support automation, monitoring, control, and data-driven decision-making. The rapid growth of CIoT has reshaped consumer experiences by streamlining and transforming interactions through the seamless connectivity of smart devices [130].

In real-world IoT applications, for example, in smart healthcare systems, wearable devices continuously monitor patient vitals and transmit data to healthcare providers. If a sensor node fails, proactive fault prediction ensures seamless data rerouting, preventing data loss and enabling timely medical interventions. Similarly, in autonomous vehicles and intelligent transportation systems, real-time data exchange is vital for vehicle-to-everything (V2X) communication. Predicting network failures and dynamically rerouting data can prevent navigation errors, improve collision avoidance, and enhance overall traffic coordination. Integrating node fault prediction with adaptive data routing is essential for ensuring reliable and efficient communication in various real-world IoT applications. It ensures that even if some sensor nodes fail, critical alerts are rerouted effectively, allowing emergency responders to act promptly.

Notably, the existing studies in the literature have primarily focused either on providing optimal data routing solutions or on tackling the challenges of node fault detection within dynamic IoT networks. Traditional fault prediction methods [49] identify

faulty nodes but do not provide an immediate mechanism to reroute data, leading to potential data loss, increased latency, and degraded network performance. Similarly, conventional routing methods [35] optimize data paths but do not consider real-time node failures, which can result in unreliable transmissions, congestion, or inefficient energy consumption. By integrating fault prediction with adaptive data routing, the network can proactively avoid faulty nodes before failures impact communication. This ensures continuous data transmission, reduced network disruptions, and enhanced energy efficiency. To bridge this gap, this chapter introduces a novel approach that integrates both aspects, leading to optimize network performance in terms of energy efficiency and QoS. By optimizing energy efficiency, the proposed approach contributes to reducing the overall power consumption of IoT devices, which is crucial for minimizing carbon footprints [131] and promoting environmentally friendly smart infrastructures. This is especially relevant in large-scale deployments such as smart cities, where efficient IoT networks can lead to sustainable resource management and reduced energy waste. Moreover, low-latency data transmission enabled by QoS-aware routing makes the approach highly suitable for critical applications like telemedicine, where real-time data exchange is essential for remote patient monitoring and emergency healthcare services. The primary contributions of this chapter are as follows:

- An innovative approach for predicting node faults is proposed using the autoencoder-based ML algorithm, which reduces the risk of data loss caused by faulty nodes.
- Building upon the faulty node detection stage provided by the autoencoder, a novel energy-efficient and QoS-aware data routing method using actor–critic reinforcement learning is proposed. The routing framework operates exclusively on the set of healthy nodes identified by the autoencoder, ensuring that data transmission avoids faulty nodes while optimizing for energy efficiency, low latency, and high throughput.

- Measurement models have been developed to assess the effectiveness of the proposed method. These models calculate the average data transmission time, network data throughput, residual energy of the network, and node fault prediction accuracy.
- A real-field IoT dataset is used to evaluate the performance of the proposed method. The real-field IoT dataset is developed based on information gathered from existing methods documented in the literature. The results obtained from this real-world IoT dataset highlight the significance of the proposed approach for real-time applications in medium- and large-scale IoT networks.
- The outcomes of the proposed method are compared with existing methodologies documented in the literature, such as dynamic directional routing (DDR) [49], enhanced fuzzy based energy efficient routing protocol (E-FEERP) [132], area based routing (ABR) [133], and a novel clustering routing protocol for dynamic WSNs (CRPD) [134] over both, simulated IoT testbed and real-field dataset.

5.1 Network Model and Problem Formulation

This section outlines the network model used to design an optimal data routing system with node fault prediction for IoT networks. Following this, the discussion moves on to formulate the problem for predicting node fault and designing QoS-aware data routing.

5.1.1 Network Model

The dynamic IoT network described in this work, presented as a graph $G = (\mathcal{N}, \mathcal{M}, \mathcal{L})$, is illustrated in Figure 5.1. Where $\mathcal{N} = \{N_1, N_2, \cdots, N_n, \cdots, N_N\}$ denotes the set of IoDs, $\mathcal{M} = \{M_1, M_2, \cdots, M_m, \cdots, M_M\}$ denotes the set of gateways and $\mathcal{L}$ represent the set of potential links established in the network depending on the transmission power level of IoDs. Figure 5.2a shows the routing in a dynamic IoT network without node fault prediction, where data packets face congestion and failures due to faulty

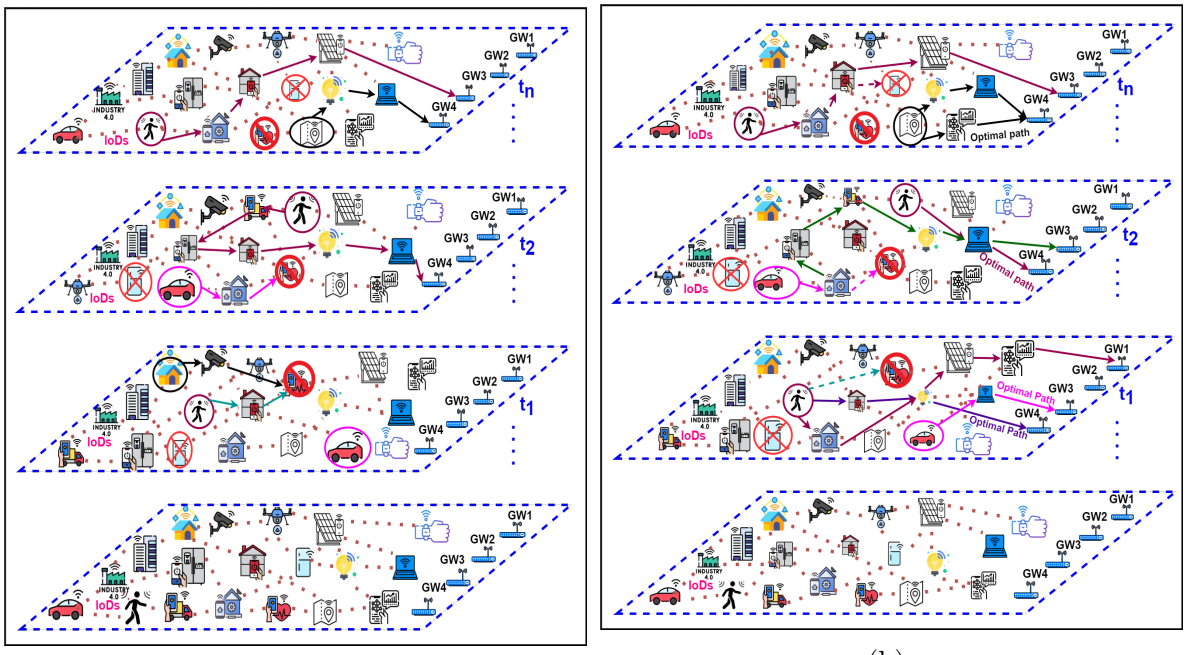


Figure 5.2: An illustration of data routing in a time-varying IoT network. (a) shows routing without node fault prediction, and (b) demonstrates routing with fault prediction

nodes (e.g., green and black paths at t_1 and the pink path at t_2). Successful transmission occurs only when paths are fault-free, like the maroon path. Whereas, Figure 5.2b demonstrates routing in a dynamic IoT network with fault prediction, where data packets are rerouted to the gateway using the most suitable relay. At t_1, the purple path is chosen for fewer hops, while at t_2, the green path bypasses predicted faults, ensuring efficient and reliable data transmission. The terminologies with their definitions are shown in Table 5.1. Since the network comprises mobile nodes, making the $\mathcal{L}$ dynamic and dependent on IoD locations at time t. IoD velocities follow a random normal distribution influenced by collision avoidance and network dimensions. In accordance with the given conditions, If the distance $d_{n,k}$ between nodes n and k is $\leq d_{max}$, then the corresponding element in the incidence matrix $\eta_{n,k}$ is set to 1; otherwise, it is 0. Here, d_{max} is the maximum direct transmission range. Furthermore, the attributes defining both the node and network are represented by $C = (\mathcal{E}, \mathcal{H}, \mathcal{Q}, \mathfrak{P})$. Here, $\mathcal{E}$ is the residual energies of the IoDs, $\mathcal{H}$ denotes the hop counts between IoD pairs, $\mathcal{Q}$

Table 5.1: Terminologies and definitions

Symbols/Parameters	Meaning
$\mathcal{N}$, $\mathcal{M}$, $\mathcal{L}$, $\mathcal{F}$	Set of N IoDs, set of M gateways, set of potential links in the network, set of flag values
$d_{n,k}$	Distance between nth and kth IoDs
d_{max}	Maximum transmission distance for direct data transfer
$\eta_{n,k}$	Incidence matrix element for node pair n and k
$\mathcal{E}$, $\mathcal{H}$, $\mathcal{Q}$, $\mathfrak{P}$	Residual energy, hop count, queue size, and transmission power level between IoDs
$S_{n,m}^{t,\alpha}$	αth path from sensor node n to gateway m at time t
$C(S_{n,m}^{t,l})$	Cost associated with the data transfer path from n to m at time t
ξ^*, $\mathcal{P}^*$	Set of faulty nodes, optimal path
$P_n(t)$, $F_n(t)$	Predicted and actual fault occurrence probabilities
ν	Weighting parameter balancing fault prediction and routing efficiency
$\mathcal{C}_{n,t}^*$	Minimum cost for the optimal data path from n to any gateway at time t
D_t^{tot}, D_t^{conv}	Total cumulative delay and conventional delay
$\beta_{n,k}^t$	Data flow from node n to k at time t
$\delta(\Delta, \Delta_{th})$	Fault prediction flag
E_c, R, D, Th	Energy consumed, data transmitted, transmission delay, throughput
p, q	Weighting factors for balancing metrics
Δ, Δ_{th}	Reconstruction loss and its threshold
x_i, y_i	ith input and reconstructed features
$\mathcal{F}_{level}$	Fault flag value
MF_l, MF_m, MF_h	Low, medium, and high membership functions
$\mathcal{A}_j$, $\mathcal{S}_{s_k}$	Action set, input variables of kth IoD
X_k, Y_k	X and Y coordinates of kth sensor node
V, ϕ	Speed and velocity angle of IoD in Cartesian plane
$\mathcal{E}$, $\mathcal{I}$, D	Residual energy, interference, and expected delay for IoD k
Γ_j	1-D vector of S_{s_k} values for all neighbors of sensor node j
$\mathcal{V}$, ζ	Critic value and TD error
$\Re$	Reward for selecting the next node
$\Re_E$, $\Re_T$, $\Re_P$, $\Re_H$	Rewards based on residual energy, delay, relative position, and number of hops

denotes the queue sizes across the IoDs, and $\mathfrak{P}$ represents the transmission power levels between IoD pairs. Furthermore, fault levels for IoDs are represented as $\mathcal{F} = \{\mathcal{F}_{level,1}, \mathcal{F}_{level,2}, \cdots, \mathcal{F}_{level,k}, \cdots, \mathcal{F}_{level,N}\}$. Where $\mathcal{F}_{level,k}$ refers to the fault level of the kth IoD for predicting the likelihood of a fault occurrence.

The area of the IoT network under consideration is $L \times W$ m^2, with randomly distributed sensor nodes and fixed gateways. Here, L and W represent the length and width of the network. Data packets from IoDs are transmitted to gateways via a multi-hop system. Relay IoD selection depends on the flag value $\mathcal{F}_{level,k}$ determined by the ADF framework and the availability of other resources. The proposed ADF framework is used to determine the flag value $\mathcal{F}_{level,k}$ for the kth IoD. IoDs with $\mathcal{F}_{level} = high$ or mid are excluded from the path. Each IoD maintains a queue to manage data transmission. Power consumption depends on the distance $d_{n,k}$ and data volume, while latency is influenced by the number of hops, interference, congestion, limited processing power, slow data transfer rates, and long transmission distances for a given IoD.

5.1.2 Problem Formulation

The primary objective of this work is to compute faulty nodes and develop an optimal path with minimum cost spent during the transmission of data from sensor node n to any gateway m over the network for all possible time instants, $t = 1, 2, 3, ..., T$. In this context, the problem of joint node fault prediction and optimal data routing over time-varying IoT networks is formulated as:

$$\{\xi^*, \mathcal{P}^*\} = argmin \left(\sum_{t=1}^{T} \sum_{n=1}^{N} (|P_n(t) - F_n(t)|) + \nu \sum_{t=1}^{T} \sum_{n=1}^{N} \mathcal{C}^*_{n,t} \right). \tag{5.1}$$

To achieve this objective, the system must dynamically adjust routing decisions based on the predicted fault probabilities and current network conditions. This adaptation aims to enhance data transmission efficiency while minimizing the impact of potential faults. The objective function is associated with a set of constraints that are:

$$\sum_{n=1}^{N} \mathcal{E}_{n,(res)} \geq \partial, \quad \forall \{n \in \mathcal{N}\}, \tag{5.2}$$

$$D_t^{tot} \leq D_t^{conv}, \quad \{\forall t = 1, 2, 3, ..., T\}, \tag{5.3}$$

$$\underbrace{\sum_{m=1}^{M} R^m \; p(t < \tau)}_{\text{Total data received at Gateways}} \leq \sum_{n=1}^{N} R^n, \tag{5.4}$$

$$\underbrace{\sum_{\forall n,k} \beta_{n,k}^t - \sum_{\forall n,k} \beta_{k,n}^t = 0}_{\text{Data flow constraint at } k\text{th node}} \;, \; \forall t, \quad n \neq k, \text{ and } \{n, k\} \in \mathcal{N}, \tag{5.5}$$

$$\delta(\Delta, \Delta_{th}) = 1, \qquad \text{if } |P_n(t) - F_n(t)| > \Delta_{th}, \tag{5.6}$$

$$\beta_{k,n} = 0, \text{ if } \delta(\Delta, \Delta_{th}) = 1, \quad \forall \{n, k\} \in \mathcal{N}, \; n \neq k. \tag{5.7}$$

The objective function given in (5.1) minimizes both the node fault prediction error and the data routing costs over time. It returns an output of a set of faulty nodes (ξ^*) and a set of optimal paths ($\mathcal{P}^*$). The first term captures the difference between the

predicted probability of a fault, $P_n(t)$, and the actual fault occurrence, $F_n(t)$, at node n and time t. The second term represents the cumulative routing cost over all nodes and time intervals. Here, ν is a weighting parameter that balances between minimizing node fault probabilities and optimizing data routing efficiency. It also considers the cost $\mathcal{C}^*_{n,t}$. $\mathcal{C}^*_{n,t}$ is the function of energy, delay, and throughput as shown in (5.9). By changing the variables p and q, we keep a balance between energy consumption and QoS parameters.

Objective function (5.1) has an important set of constraints attached to it that guide the network's behavior to meet real-time application needs. Constraint (5.2) ensures that at any given time instant t, the residual energy of all nodes ($\mathcal{E}_{n,\text{res}}$) remains above a threshold (∂), ensuring sufficient energy for operation and prevents premature depletion of nodes. Constraint (5.3) limits the cumulative delay along any path to stay within conventional multi-hop data routing delays, maintaining network responsiveness and ensuring timely data delivery. Similarly, constraint (5.4) balances data flow by ensuring that the total data received by gateways does not exceed the data generated by IoT devices. This equilibrium guarantees that the network efficiently handles data transmission without overloading any specific segment. Constraint (5.5) maintains data flow equilibrium at the node level, ensuring incoming and outgoing data volumes match to prevent bottlenecks. Furthermore, constraint (5.6) enables fault prediction by setting a fault flag for a node. It indicates that if the difference between actual and predicted fault probability for the nth node at t time instant is greater than a minimum threshold (Δ_{th}), then the fault prediction flag for that node will be 1. Lastly, constraint (5.7) safeguards the network by prohibiting data transmission to nodes flagged as faulty. It specifies that in the event of nth node being identified as faulty, indicated by a prediction flag value of 1, then no data transmission from any node k to the faulty node n will be permitted, i.e., $\beta_{n,k}= 0$. It helps to avoid unreliable paths, improving network resilience. By integrating these constraints, the optimization model achieves a trade-off

between energy-aware routing and maintaining high-performance data transmission in dynamic IoT environments.

To transfer the data from a randomly selected sensor node to a possible gateway, there can be several paths. Let's consider a matrix that represents all possible paths from sensor node $n = \{1,2,\cdots,N\}$ to gateway $m = \{1,2,\cdots,M\}$. For instance, to transfer the data from sensor node 1 to gateway 1 at time instant t, there are α number of paths available, that is represented as $\{s_{1,1}^{t,1}, s_{1,1}^{t,2}, \cdots, s_{1,1}^{t,\alpha}\}$. Similarly, to transfer the data from sensor node 1 to gateway 2, there are β number of paths available. Therefore, the total possible paths to transfer the data from sensor node 1 to any gateway m $\in \mathcal{M}$, at time instant t are $l_1 = \alpha + \beta + \cdots + \gamma$. Now, the task is to find the optimal path having minimum cost out of all these l_1 paths. Thereafter, the minimum cost associated with that particular path is selected for every sensor node $n \in \mathcal{N}$, i.e., $\mathcal{C}_1^* = min\{\{C(s_{1,m}^{t,|l_1|})\}_{m=1}^{M}\}$, $\mathcal{C}_2^* = min\{\{C(s_{2,m}^{t,|l_2|})\}_{m=1}^{M}\}$, ... ,$\mathcal{C}_n^* = min\{\{C(s_{n,m}^{t,|l_3|})\}_{m=1}^{M}\}$, ..., $\mathcal{C}_N^* = min\{\{C(s_{N,m}^{t,|l_n|})\}_{m=1}^{M}\}$. Here, $\mathcal{C}_n^*$ denotes the minimum cost associated with that one particular path out of all possible paths exist to transfer the data from nth sensor node to any gateway, at time instant t. Here, $l_1 = \{\alpha + \beta + \cdots + \gamma\}$; $l_2 = \{\kappa + \sigma + \cdots + \mu\}$; $\cdots$; $l_n = \{\phi + \chi + \cdots + \psi\}$ are the total possible paths to transfer the data from sensor node $n \in \mathcal{N}$, to any gateway m $\in \mathcal{M}$, respectively. Where,

$$C(s_{n,m}^{t,|l|}) = \underbrace{(pE_c + qD + (1-p-q)\frac{1}{Th}).}_{\text{Sum of the normalized value of energy, delay, and throughput inverse}} \tag{5.9}$$

$C(s_{n,m}^{t,|l|})$ is the cost associated with the path that transfers the data from the nth sensor node to mth gateway. It is calculated by adding the normalized value of consumed energy (E_c), transmission delay (D), and throughput (Th). Here, p and q are weighing factors associated with balancing the equation. The parameters are normalized using the formula: $X_{norm} = \frac{X_{act} - X_{min}}{X_{max} - X_{min}}$. Here, X_{act}, X_{min}, and X_{max}, are the actual, minimum, and maximum values of the parameter X, which are used for the normalization.

5.2 Node Fault Prediction And Data Routing In Time Varying IoT network

This section explains the proposed approach for optimizing the energy efficiency and QoS in data routing while incorporating node fault prediction, using the auto-encoder anomaly detection with fuzzy framework (ADF) for fault detection and actor-critic reinforcement learning-based routing technique.

5.2.1 Node Fault Prediction Using ADF Framework

A sensor node can be classified as normal or defective based on its state pattern. To determine this, the status of node k at the ith iteration (S_k^i) is analyzed as a time series of data. This time series consists of the states of node k for the previous a iterations, represented as $S_k^i = \{E_c, R, D\}_a$. Here, E_c stands for the energy consumed, R represents the total data transmitted, and D indicates the transmission delay. After each iteration, the oldest state is replaced with updated values, including energy consumption, data transfer, and latency, forming a-row time series dataset, where a is 10.

When a sensor node becomes faulty, it shows anomalous behavior in its state patterns, like increased energy consumption, high transmission delay, and increased non-uniform data transmission. To detect this, time-series data representing a node's state is processed through an autoencoder. Auto-encoder compresses the input data into a lower-dimensional representation via the encoder and then reconstructs the original data using the decoder. The decoder reconstructs all a data points based on normal behavior. The reconstruction loss between the original and reconstructed time series is calculated to identify anomalies. If the reconstruction loss exceeds the threshold, then the node is categorized as faulty. The framework for classifying nodes as either faulty or normal is described as follows.

5.2.1.1 Model Architecture

An auto-encoder based neural network architecture is used for anomaly detection that is tailored for sequence-to-sequence tasks, with a particular emphasis on time series data. The proposed architecture uses the potential of long short term memory (LSTM) units for modeling temporal dependencies effectively. The main components of this model architecture are:

Encoder Component: Encoder starts with the input layer. It accepts multivariate time series data with dimensions $(n_{past}, n_{features})$, where n_{past} is the temporal length of historical data and $n_{features}$ is the number of features per data point. As shown in

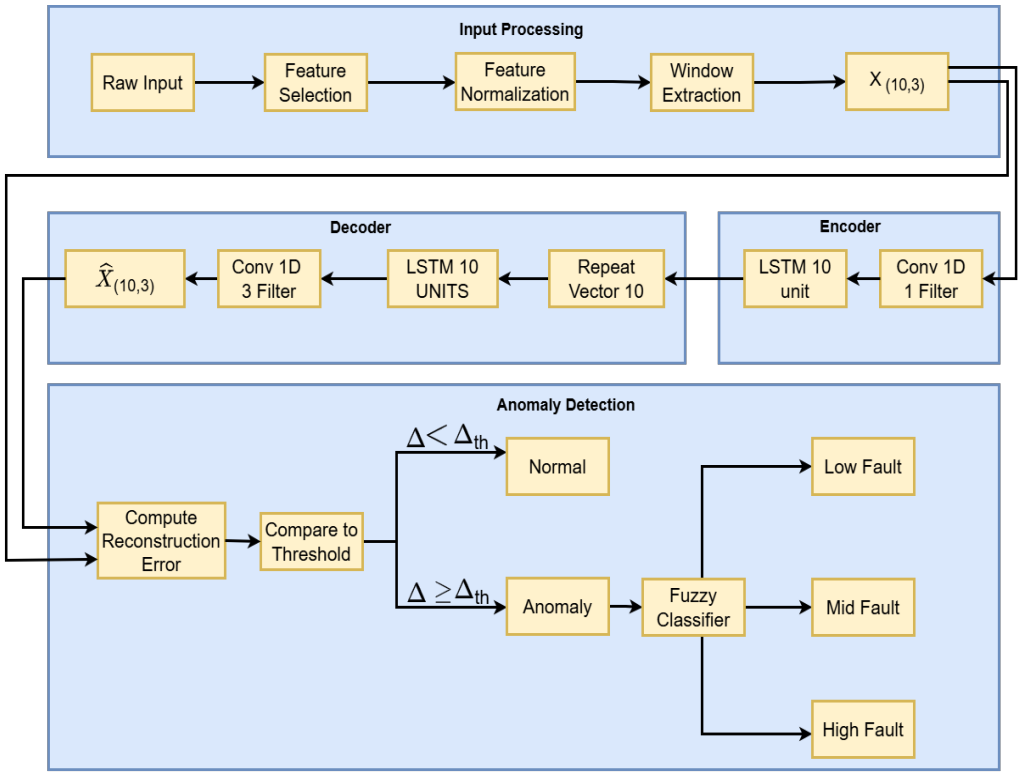


Figure 5.3: Method of node fault prediction using autoencoder anomaly detector with fuzzy (ADF) framework

Figure 5.3, the input sequence $X_{(10,3)}$ is passed through a convolution layer with a single filter ($k_e = 1$) and kernel size (K_{ae}) of 5. It is utilized to reduce the feature space while effectively preserving essential temporal information and reducing the input complexity

for subsequent layers. The utilized convolution layer can be expressed by the following equation

$$y[t,:,k_e] = \sum_{i=0}^{K_{ae}-1} \sum_{j=0}^{n_{features}-1} (\tilde{X}[t+i-2,:,j] \cdot w_e[i,j,k_e]), \tag{5.10}$$

where, $\tilde{X}[t+i-2,:,j]$ represents input X_i with zero padding and $w_e[i,j,k_e]$ represents kernel weights of encoder's convolutional layer.

The convolution operation is followed by an LSTM layer with 10 units, which processes the reduced output sequence. In this, *tanh* activation function is used for the LSTM layer without sequence return. The output of this LSTM layer is expressed as $z_t = h_t = o_t \odot tanh(c_t)$. Here, o_t represents the output gate, c_t represents the cell state, h_t represents the final hidden state. This LSTM output is then passed through the RepeatVector layer, which prepares the input for the decoder component.

Decoder Component: The decoder generates reconstructed sequences of the same length as the input sequence. It consists LSTM layer, which has 10 units and uses tanh activation. As shown in Figure 5.3, the RepeatVector output, i.e., $R(z_t)$ is passed into the LSTM layer, which processes the repeated vector with sequence return. By using repeated vectors, this layer's ability to generate coherent sequences is improved. The LSTM output at each time step is $z'_t = h'_t = o'_t \odot tanh(c'_t)$. The output of the LSTM is then passed to a convolution layer with three filters (k_d=3) and a kernel size of 5. Filters are used to enable the network to reconstruct the full feature space, which aligns with the original feature dimensions. The output of the utilized convolution layer can be expressed as

$$\hat{X}'[t,:,k_d] = \sum_{i=0}^{K_{ae}-1} \sum_{j=0}^{H_{ae}-1} (\hat{X}[t+i-2,:,j] \cdot w_d[i,j,k_d]). \tag{5.11}$$

Here, $w_d[i,j,k_d]$ represents kernel weights of the decoder's convolution layer, and $\hat{X}'$ is the final reconstructed output.

5.2.1.2 Reconstruction Loss

The reconstruction loss is computed as a means of measuring the dissimilarity between the original input time series (X) and the reconstructed time series ($\hat{X}'$) generated by the auto-encoder model. Reconstruction loss is calculated using a mean absolute error function. The mean absolute error (Δ) is calculated as the average absolute difference between each input feature and its corresponding reconstructed feature as $\Delta = \frac{1}{f}\sum_{i=1}^{n}|X_i - \hat{X}_i|$. Where f is the number of features in the data, X_i is the ith feature in the input data, and $\hat{X}_i$ is the ith feature in the reconstructed data.

5.2.1.3 Thresholding

A threshold is decided based on the reconstruction errors of training data. Data points with reconstruction errors above this threshold are classified as anomalies.

$$\delta\left(\Delta, \Delta_{\text{th}}\right) = \begin{cases} 1, & \Delta \geq \Delta_{\text{th}} \\ 0, & \Delta < \Delta_{\text{th}} \end{cases}. \tag{5.12}$$

Where Δ_{th} is the threshold reconstruction loss.

5.2.1.4 Fuzzy Logic for further classification

If an anomalous node is detected, it is further classified into three categories: (1) low, (2) mid, and (3) high fault using fuzzy logic. The linguistic variables for output are defined as:

$$\mathcal{F}_{level} = \{low, mid, high\}. \tag{5.13}$$

Fuzzy membership functions are created for classifying low, mid, and high faults. The fuzzy membership function assigns a degree of membership, ranging from 0 to 1, to elements in the fuzzy set, indicating their degree of belonging. Fuzzy membership functions enable flexible management of ambiguity in real-world data and decision-

making. The membership functions created for fault levels are as follows:

$$MF_{\mathrm{l}} = \begin{cases} 0, & \text{if } \Delta < \mu_1 \\ \frac{\Delta-\mu_1}{\mu_2-\mu_1}, & \text{if } \mu_1 \leq \Delta < \mu_2 \\ 0, & \text{if } \Delta \geq \mu_2 \end{cases}, \tag{5.14}$$

$$MF_{\mathrm{m}} = \begin{cases} 0, & \text{if } \Delta < \tau_1 \\ \frac{\Delta-\tau_1}{\tau_2-\tau_1}, & \text{if } \tau_1 \leq \Delta < \tau_2 \\ \frac{\tau_3-\Delta}{\tau_3-\tau_2}, & \text{if } \tau_2 \leq \Delta < \tau_3 \\ 0, & \text{if } \Delta \geq \tau_3 \end{cases}, \tag{5.15}$$

$$MF_{\mathrm{h}} = \begin{cases} 0, & \Delta < \alpha_1 \\ \frac{\Delta-\alpha_1}{\alpha_2-\alpha_1}, & \alpha_1 \leq \Delta < \alpha_2 \\ \frac{\alpha_3-\Delta}{\alpha_3-\alpha_2}, & \alpha_2 \leq \Delta < \alpha_3 \\ 0, & \Delta \geq \alpha_3. \end{cases} \tag{5.16}$$

Algorithm 5.1 Node Fault Prediction Using ADF Framework

INPUT: In the ith iteration, Time-series data for the kth Node: $S_k^i = \{E_c, R, D\}_a$

INITIALIZATION

1. For each node i (where $i \in \{1, 2, \ldots, N\}$), initialize a list containing the average values of the features: For each node $i : [\bar{E}_c, \bar{D}, \bar{R}]_a$
2. Generate Auto-encoder model, i.e., ADF.

OFFLINE LEARNING

3. **for** 3,000 iterations **do**
4. Feed Non-Anomalous input (X) into the Autoencoder (ADF).
5. The reconstructed data ($\hat{X}$) is produced by the ADF.
6. The reconstruction loss Δ between input (X) and reconstructed data $\hat{X}$ is calculated.
7. Update weights and biases of the ADF to minimize Δ.
8. **end**

Anomaly Detection

9. **for** all the sensor nodes in $\mathcal{N}$ **do**
10. Feed time series data of that node to ADF
11. Reconstruction loss (Δ) is calculated using (5.12)
12. **if** $\Delta > \Delta_{th}$, **then**
13. $\delta(\Delta, \Delta_{th}) = 1$
14. Calculate MF_l, MF_m, and MF_h.
15. **if** $MF_h > MF_m$ and $MF_h > MF_l$, **then**
16. $\mathcal{F}_{level} = high$
17. **else if** $MF_m > MF_h$ and $MF_m > MF_l$, **then**
18. $\mathcal{F}_{level} = mid$
19. **else**
20. $\mathcal{F}_{level} = low$
21. **Return** (1, $\mathcal{F}_{level}$)
22. **else**
23. $\delta(\Delta, \Delta_{th}) = 0$
24. **Return** (0, None)
25. **end**

The fault classification will be based on the value of the membership functions (MF_l, MF_m, and MF_h). The membership function that gives the highest result determines the associated fault. For example, if MF_l possesses the highest numerical value among the three membership functions, it will categorize the fault as low, similar to medium and high faults. The whole process of node fault prediction is shown in Figure 5.3 and *Algorithm 5.1.*

5.2.2 Data Routing Using Actor-Critic RL

For data routing in the network, an actor-critic reinforcement learning (RL) framework is proposed in this work. Actor-critic is an RL algorithm designed for sequential

decision-making, combining the strengths of policy-based and value-based methods. It consists of two core components: the Actor and the Critic. The **Actor** is a policy model used for the action selection mechanism. It returns a probability distribution over actions. The **Critic** is a value model that estimates the expected reward as critic value for a given state-action pair using the current policy. It checks how good a particular action is for the current state. Every RL algorithm comprises an **agent**, a set of **states**, and a set of **actions**. The actor-critic framework proposed in this work considers the message flow across the network as the **agent**, current sensor node N_j as the agent's current **state**, and identifying the next node (s') for message transmission as the **action** taken at that state. The sets of states and actions are denoted as $\mathcal{S} = \{s_1, s_2, s_3, \cdots, s_N\}$ and $\mathcal{A} = \{\ \mathcal{A}_1,\ \mathcal{A}_2, \cdots, \mathcal{A}_j, \cdots, \mathcal{A}_N\}$, respectively, where $\mathcal{A}_j = \{s_k; \forall s_k \in \mathcal{N}_{s_j}\}$. In this representation, $\mathcal{N}_{s_j}$ signifies the collection of the 10 nearest neighbors of the IoD s_j. Where,

$$s_k = \begin{cases} s_k, & d_{jk} \leq d_{max} \\ NULL, & d_{jk} > d_{max} \end{cases}. \tag{5.17}$$

The actor-critic method uses three different models for optimal routing - **policy**, **actor**, and **critic**.

1. *Policy Model:* The policy model selects the final best action from the action set $\mathcal{A}_j$. It accepts Γ_j as input vector where Γ_j is represented by: $\Gamma_j = \{\mathcal{S}_{s_k}: \ \forall s_k \in \mathcal{A}_j\}$. Where, $\mathcal{S}_{s_k} = \begin{cases} \{X_k, Y_k, V, \phi, \mathcal{E}, \mathcal{I}, D\}, & \text{if } s_k \neq \text{NULL}, \\ \{0,0,0,0,0,0,0\}, & \text{otherwise.} \end{cases}$
Here, X_k and Y_k are the X and Y coordinates of kth sensor node. V represents the speed of kth sensor node, and ϕ represents the angle of velocity of an IoD as observed in the cartesian plane. $\mathcal{E}$, $\mathcal{I}$, and D denote the residual energy, interference, and expected delay for the kth IoD, respectively. Γ_j is a 1-D vector which contains $\mathcal{S}_{s_k}$ values for all 10 nearest neighbour nodes of jth sensor node.

The policy model returns the probabilities of selecting each node in the action set $\mathcal{A}_j$.

2. *Critic Model:* The critic model accepts Γ_j as an input vector for a state and returns a critic value ($\mathcal{V}$) for that state. The critic values of the current state and next state are calculated in every iteration of learning. The next state is computed using the action returned by the policy model. The calculated critic values are used to calculate the temporal difference error, popularly known as TD error (ζ), in an iteration. It is calculated as:

$$\zeta = \Re + (\gamma \times \mathcal{V}_k) - \mathcal{V}_j. \tag{5.18}$$

In the above equation, TD error (ζ) is calculated for the transition of the message from state j to state k. $\mathcal{V}_j$ and $\mathcal{V}_k$ represents the critic values of states j and k respectively. γ is the discount factor and $\Re$ is the observed **reward** for the particular action. The reward value is calculated using (5.22). The critic loss used for training the critic model is taken as the **mean squared value** of the TD error.

3. *Actor Model:* The actor model is similar to the policy model with identical weights. It has one extra input field for **TD error** (ζ), obtained from the values predicted by the critic model. ζ is appended to the vector Γ_j and passed on to the actor as input for training. The actor model is trained using the **categorical cross-entropy loss**. The updated weights of the actor model are copied to the policy model after each iteration of training.

The reward is computed using four important parameters i.e., residual energy ($\Re_E$) of the next node, expected transmission delay ($\Re_T$) between two nodes, relative positions ($\Re_P$) of the two nodes and the number of hops ($\Re_H$) the message has undergone. $\Re_E$ is defined as

$$\Re_E = \frac{\text{Residual energy of the next node}}{\text{Initial energy of the next node}} \tag{5.19}$$

The expected transmission delay is calculated as described in the data latency measurement model provided in **Section 5.3.3**.

To maintain equal weightage of all the parameters in the reward function, we have taken all the terms in the form of ratios. This returns unitless values on a scale of 0 to 1. Thus, $\Re_T$ is the normalized value of transmission delay. $\Re_P$ is calculated as

$$\Re_P = \frac{X_k - X_j}{\text{TR}}. \tag{5.20}$$

Where TR is the transmission range. If the x coordinate of the next node is less than the x coordinate of the current node, it shows that the message is being forwarded in the opposite direction. Thus, it will have a negative impact on the total reward. $\Re_H$ accounts for the contribution of hop counts in the reward. $\Re_H$ is simply defined as the normal distribution of the number of hop counts ($\mathcal{H}$), i.e.,

$$\Re_H = \exp\left(-\frac{\mathcal{H}^2}{2}\right), \tag{5.21}$$

promoting the message to take as less hop counts as possible. The final reward $\Re$ for transmission of message from node j to node k is calculated using the following relation:

$$\Re(j,k) = \begin{cases} \Re_H \times (2 + \Re_E - \Re_T + \Re_P), & k \in \mathcal{N} \\ \rho, & k \in \mathcal{M} \end{cases}. \tag{5.22}$$

Here, 2 is added in reward to ensure that the calculated reward is always positive. If the reward is negative, then the term $\Re_H$ will have opposite effect than expected. ρ is higher reward value associated with gateway nodes. Gateway nodes have been assigned higher reward value to give them more priority than other neighboring nodes. The final routing decisions are made by policy model, which was obtained after training the actor and critic models during offline learning. The complete process of data transmission and offline learning is explained in *Algorithm 5.2* and shown in Figure 5.4.

Algorithm 5.2 Node Fault Prediction Based Optimal Data Routing Using Actor-Critic RL in IoT Netwroks

INPUT : Nodes set $\mathcal{N}$= $\{N_1, N_2,\cdots,N_n,\cdots,N_N\}$, gateways set $\mathcal{M} = \{M_1, M_2,\cdots,M_m,\cdots,M_M\}$, $\forall\{m = 1, 2, \cdots, M; n = 1, 2, \cdots, N\}$

INITIALIZATION

1. Randomly distribute N sensor nodes and M gateways across a network area of $L \times W$ m^2, each initialized with an energy level E and a unique identification number.

2. Create a list of 10 neighboring nodes using distance values for each sensor node, which is similar to the action set $\mathcal{A}_n$ for each node.

$\mathcal{A}_n = \{s_k;\ \forall s_k \in \mathcal{N}_{s_n}\}$, where s_k is computed using (5.17).

3. Generate 3 CNN models, one for each - *policy*, *actor*, and *critic*.

OFFLINE LEARNING

5. **for** η iterations **do**

6. Generate a *message* by randomly selecting an IoD from the network

7. Set *message.curr_device* = IoD_1

8. Make Γ_j vector for *message.curr_device*

9. Compute action probabilities by passing Γ_j to the actor model

10. Select *message.next_device* using **Epsilon-Greedy** from action probabilities

11. Calculate reward $\Re$ using (5.19)

12. Generate Γ_k vector for *message.next_device*

13. Calculate critic values $\mathcal{V}_j$ and $\mathcal{V}_k$ from the *critic* model

14. Calculate ζ using (5.18)

15. Update weights of *actor* and *critic* models using respective loss functions

16. Copy weights of *actor* model to *policy* model

17. **end**

18. **Reset** all the sensor nodes to their initial Energy levels E, as well as other attributes like node interference and total data transmitted/received, to 0.

DATA ROUTING

19. Initialize *messages* list and set $i = 0$

20. **while** ($i \leq 2000$)
21. **for** all the sensor nodes in $\mathcal{N}$ **do**
22. Generate *message* with a probability of generating message as 50%.
23. Set $message.curr\_device = IoD_1$
24. Add *message* to *messages* list
25. **end**
26. **for** each *message* in *messages* **do**
27. **for** each *neighbour* of *message.curr_device*
28. **if** $neighbour.\mathcal{F}_{level} == mid$ or *high*
29. Set $s_k = NULL$
30. **end**
31. Update Γ vector for *message.curr_device*
32. $message.next\_device \leftarrow policy(\Gamma)$
33. Update $\mathcal{E}$ and $\mathcal{I}$
34. Calculate and update $\mathcal{F}_{level}$ value for *message.curr_device* using **Algorithm 5.1.**
35. $message.curr\_device = message.next\_device$
36. **if** $message.curr\_device \in \mathcal{M}$
37. *messages*.remove(*message*)
38. **end**
39. $i = i + 1$
40. **end**
41. **end**

During offline learning, the routing decisions are made using the Epsilon-Greedy algorithm. In this, the value of the probability of exploration (ϵ) accounts for randomness in the action selection. Initially, ϵ is taken as a larger value so that the model learns all the possibilities of the network. The value of ϵ is degraded over time to promote the best actions. After offline learning, all the nodes are reset to their initial energy levels, and all the other properties of the IoDs are reset to their initial values. Every device has a parameter $\mathcal{F}_{level}$, which indicates the level of faultiness in each device. The $\mathcal{F}_{level}$

for a device is updated whenever a message is transmitted by that IoD. While defining the Γ_j vector for selecting the next node, the $\mathcal{F}_{level}$ of each neighbour IoD is checked. If the $\mathcal{F}_{level}$ for any neighbor is *mid* or *high*, then that IoD is discarded from the set of 10 nearest neighbors, i.e., its value of s_k is set to $NULL$. Thus, the policy model will never select a faulty node for transmission.

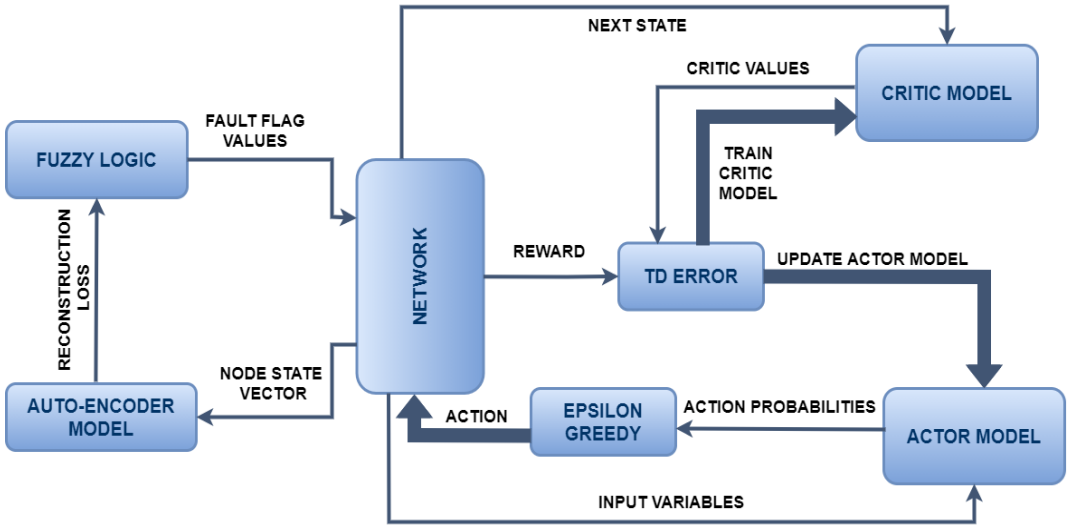


Figure 5.4: Method of joint node fault prediction and optimal data routing in dynamic IoT networks

5.3 Measurement Models For Node Fault Prediction, Energy-Efficiency and QoS

This section presents measurement models for evaluating the proposed approach, including node fault prediction metrics (accuracy, precision, recall, F1 score), QoS metrics (throughput ratio, data latency), and energy consumption based on transferred data and distance.

5.3.1 Node Fault Prediction Measurement Model

For the measurement of node fault, the following parameters are used:

$$Accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}, \tag{5.23}$$

$$Precision = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{5.24}$$

$$Recall = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{5.25}$$

$$F1 \text{ Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{5.26}$$

Where, true positive (TP) is the number of nodes correctly predicted as faulty, true negative (TN) is the number of nodes correctly predicted as non-faulty, false positive (FP) is the number of nodes incorrectly predicted as faulty when they are not, and false negative (FN) is the number of nodes incorrectly predicted as non-faulty when they are faulty. This is explained with the help of the confusion matrix as shown in Table 5.2.

Table 5.2: Confusion matrix

	Predicted Values(+)	Predicted Values(-)
Actual Values (+)	TP (True Positive)	FN (False Negative)
Actual Values (-)	FP (False Positive)	TN (True Negative)

5.3.2 Throughput Ratio Measurement Model

The throughput ratio is defined as the proportion of the overall data received by the gateways to the total data generated across the network. If R^m represents the total data received by the mth gateway and R^n represents the total data generated by the nth node, then the throughput ratio expressed as

$$\text{Throughput Ratio} = \frac{\sum_{m=1}^{M} R^m}{\sum_{n=1}^{N} R^n}. \tag{5.27}$$

5.3.3 Data Latency Measurement Model

Within a multi-hop IoT network, several delays are experienced that are:

5.3.3.1 Transmission Delay

Transmission delay denoted as D_{tra}, represents the time needed to transmit data directly from one IoD to another through a direct link. This time duration depends upon both the separation between nodes ($d_{n,k}$) and the transmission speed (TS), mathematically expressed as

$$D_{\text{tra}} = \frac{d_{n,k}}{TS} \tag{5.28}$$

5.3.3.2 Processing Delay

Processing delay or data packet processing time in an IoD, is the time taken to perform various tasks such as error checking and determining subsequent actions on a data packet. The processing delay is computed using a random exponential variable, represented as:

$$D_{\text{pro.}}(x, \lambda) = \begin{cases} \lambda e^{-\lambda x}, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}, \tag{5.29}$$

here, λ is the distribution parameter ($\lambda > 0$).

5.3.3.3 Packetization Delay

It refers to the time it takes to assemble a data packet before it can be transmitted into the channel, i.e.,

$$D_{\text{pkt.}} = \frac{\text{Packet Size (bits)}}{\text{Transmission Rate (kbps)}} \tag{5.30}$$

5.3.3.4 Queuing Delay

The period during which data packets are held in a queue after processing by an IoD is termed queuing delay. Queuing delay is modeled as a Poisson-distributed random variable and can be calculated as

$$D_{\text{que.}} = P_{\text{distribution}}(k \text{ events}) = \frac{\mu^k e^{-\mu}}{k!} \tag{5.31}$$

Where μ represents the mean of the Poisson distribution. Consequently, the overall delay experienced by an IoD during data transmission is determined as

$$D_{\text{total}} = D_{\text{tra.}} + D_{\text{pro.}} + D_{\text{pkt.}} + D_{\text{que.}} \tag{5.32}$$

5.3.4 Energy Consumption Measurement Model

Sensor nodes consume energy for sensing, processing, transmitting, and receiving. Data transmission is the main reason for energy consumption. Energy usage depends on the number of packets and the distance (d) between the transmitter and receiver. The free-space model is used for distances less than a threshold (d_o), where power loss scales as d^2, while the multi-path fading model applies for greater distances, with power loss scaling as d^4. The transmitter uses energy for both radio electronics and amplification, whereas the receiver's energy dissipation is limited to radio electronics. Consequently, the power utilized by the source IoD to transmit a data packet of k bits to a receiver situated at a distance d is calculated as

$$E_{tx}(k,d) = \begin{cases} k\varepsilon_e + k\varepsilon_{fs}d^2, & \text{if } d < d_0 \\ k\varepsilon_e + k\varepsilon_{mp}d^4, & \text{if } d \geq d_0 \end{cases} \tag{5.33}$$

Whereas, the energy utilized by the receiver IoD for receiving a k bit data packet is expressed as $E_{rx} = k \times \varepsilon_e$. Here, the symbols hold the following meanings: ε_e represents the energy expenditure associated with activating and deactivating the receiver and transmitter circuitry, ε_{fs} signifies the energy consumption linked to the free space path loss model, and ε_{mp} denotes the energy consumption related to the multi-path power loss model. The energy model in our experiments utilizes specific values: $\varepsilon_e = 50 \times 10^{-9}$ J/bit, $\varepsilon_{fs} = 10 \times 10^{-12}$ J/bit/m^2, and $\varepsilon_{mp} = 0.013 \times 10^{-12}$ J/bit/m^4.

5.4 Performance Evaluation

Both real-world and simulated IoT testbeds are used to evaluate the effectiveness of the proposed method. The experimental conditions employed for the evaluation are outlined first, followed by an analysis of node fault prediction, QoS, and energy efficiency results. Notably, the assessment of node fault prediction focused on accuracy, precision, and recall values. QoS metrics include throughput ratio and average data latency, while energy efficiency is covered through network lifetime and the energy balancing phenomenon.

5.4.1 Experimental Conditions

This section outlines the experimental configurations used for assessing the effectiveness of the proposed method. Results are obtained using Python programming and relevant libraries. The implementation is carried out using TensorFlow and Keras for developing and training the machine learning models, including the ADF framework and the actor-critic reinforcement learning model. NumPy and Pandas are utilized for data processing and manipulation, while Matplotlib and Seaborn are employed for visualizing performance metrics such as loss curves, convergence graphs, and comparative analyses. Additionally, Scikit-learn is used for evaluating classification performance.

5.4.1.1 Simulated IoT testbed

The simulation employs a network of size 80×80 m^2, accommodating a total of 104 sensor nodes (IoDs), with 4 of them designated as gateways. The placement of IoDs follows a uniform distribution, and the gateways are strategically positioned at coordinates (80,70), (80,50), (80,30), and (80,10). Each IoD has a broadcast range of 20 meters, allowing them to transmit data directly or in a multi-hop manner to the gateway or neighboring IoD, based on the connection range of their radio signals. The data packet size fluctuates depending on the number of sensors equipped: 1100 bytes for

IoDs with one sensor, 1200 bytes for those with two sensors, and 1300 bytes for IoDs with three sensors.

5.4.1.2 Real-field IoT testbed

Experiments are carried out on an actual IoT testbed situated in a real-world environment. Data for the experiments are collected from 54 sensor nodes deployed within the Intel Berkeley research laboratory [124]. The Intel Berkeley dataset is widely used for IoT research due to its real-world sensor data. While it does not fully capture the complexity of modern large-scale, dynamic IoT networks, it still provides valuable insights into sensor behavior, fault occurrences, and network dynamics. Accordingly, these features make it a useful benchmark for evaluating the proposed method. The area of the network in the real-field dataset is 45 $\times$35 m^2, encompassing a total of 56 sensor nodes, including 2 gateways. The Mica2Dot sensors, equipped with weatherboards, collected timestamped topological information. Additionally, humidity, temperature, light, and voltage measurements are recorded at 31-second intervals.

5.4.2 Node Fault Prediction Analysis

The effectiveness of the suggested node fault prediction method is assessed based on accuracy, precision, and recall values. These metrics are evaluated using measurement models for node fault prediction outlined in **Section 5.3.1**.

5.4.2.1 Accuracy Performance

The accuracy of the proposed method is evaluated by comparing it to various node defect prediction models, such as the random forest (RF), fuzzy, local outlier factor (LOF), and K-means. Figures 5.5a, 5.5b, 5.5c, and 5.5d depict the outcomes of the simulated dataset, the simulated dataset with a changed environment, real-field dataset, and real-field dataset with a changed environment, respectively. The proposed approach

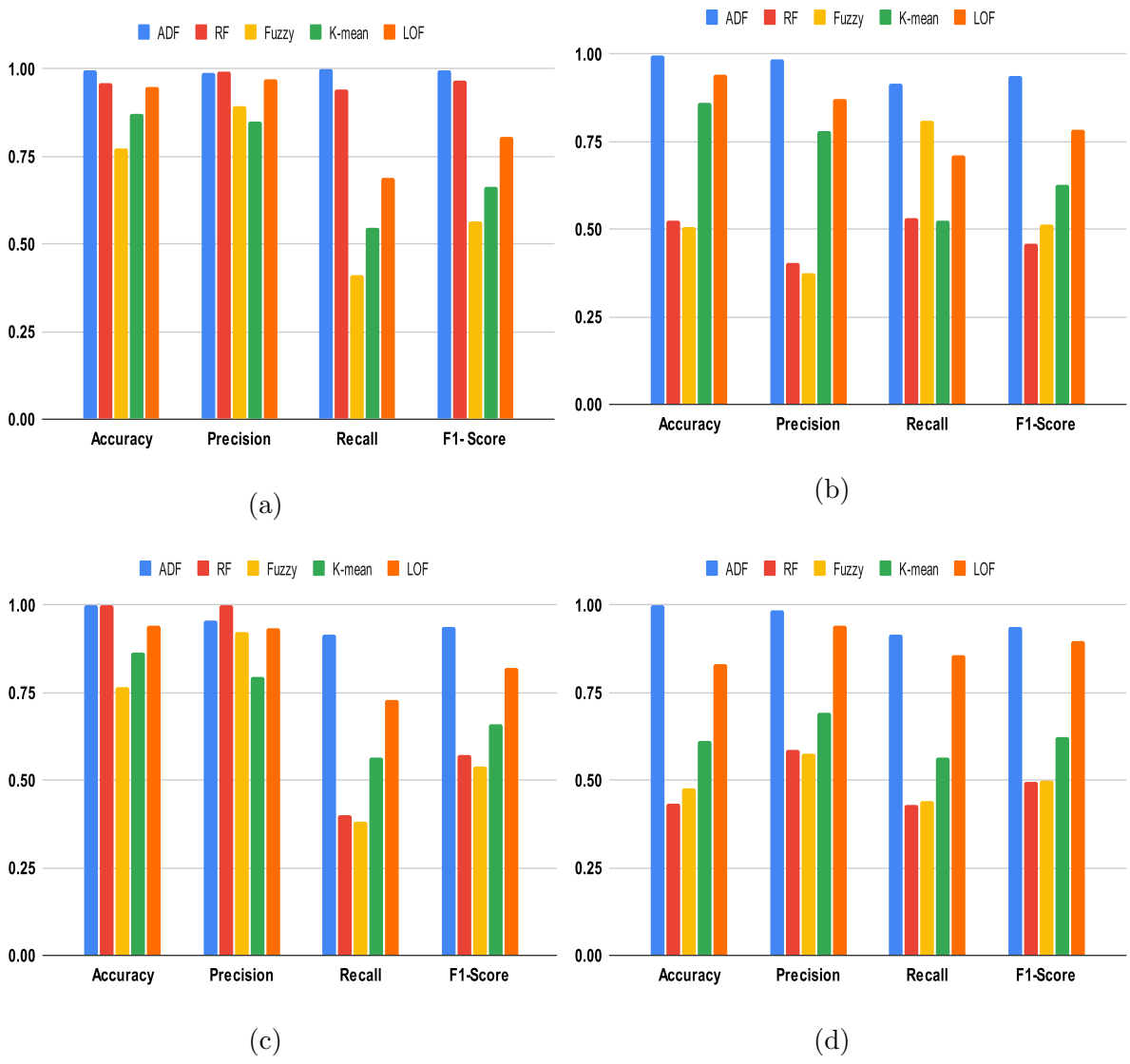


Figure 5.5: Comparative analysis of node fault prediction methods in terms of accuracy, precision, and recall values in the range of [0, 1]. The effectiveness of the suggested approach (ADF) is compared with existing methods, including RF, Fuzzy, K-means, and LOF. Performance analysis of each method across (a) simulated IoT testbed, (b) simulated IoT testbed with changed environment conditions, (c) over real-field dataset, and (d) over real-field dataset with changed environment conditions

achieves perfect accuracy, i.e., 1, on both simulated datasets and simulated datasets with changed environment, as depicted in Figures 5.5a and 5.5b, respectively. In contrast, other methods achieve values that are significantly lower than 1. This behavior occurs because K-means assumes data clusters are spherical and equidistant, an assumption that often fails with complex datasets. Fuzzy logic depends on predefined membership functions and rules, while LOF focuses on local density variations. These methods are less effective in detecting complex, global patterns in high-dimensional data. Moreover, on real-field datasets, the proposed method and RF achieve perfect accuracy, i.e., 1, as illustrated in Figure 5.5c, because RF effectively handles high-dimensional data and captures non-linear relationships through ensemble learning. However, this is not the case for real-field datasets with changed environment, as shown in Figure 5.5d.

5.4.2.2 Precision Performance

The precision performance of the proposed method is evaluated by comparing it to various node fault prediction models, including RF, Fuzzy, LOF, and K-means. Figures 5.5a, 5.5b, 5.5c, and 5.5d feature the outcomes of the simulation dataset, the simulation dataset with changed environment, the real-field dataset, and the real-field dataset with changed environment, respectively. Figure 5.5a demonstrates the precision of the proposed method and RF approaches 1. Figures 5.5b and 5.5d show that the precision of the proposed method among all other methods approaches 1 over simulated datasets with a changed environment and real-field datasets with a changed environment. Whereas other methods yield a significantly lower precision value. However, as shown in Figure 5.5c , RF is the only method outperforming the proposed approach on real-field datasets, as tuning and feature representation are crucial for precision.

5.4.2.3 Recall Performance

The recall analysis results for the simulation dataset, the simulation dataset with modified environment, the real-field dataset, and the real-field dataset with changed envi-

ronment are illustrated in Figures 5.5a, 5.5b, 5.5c, and 5.5d, respectively. Analysis of Figures 5.5a and 5.5c indicates that the proposed method performs better recall than alternative node fault prediction models, namely RF, Fuzzy, LOF, and K-means, as the reason is discussed in the previous section. Moreover, Figures 5.5b and 5.5d reveal a significant increase in recall value when implementing the suggested approach. Whereas, K-means, LOF, fuzzy, RF, and fuzzy all exhibit low values. The suggested method demonstrates consistently superior recall performance compared to existing methods, even in a changed environment.

5.4.2.4 F1 Score

The F1-Score of the proposed method is compared with RF, Fuzzy, LOF, and K-means across different datasets: simulation (Figure 5.5a), simulation with changed environment (Figure 5.5b), real-field (Figure 5.5c), and real-field with changed environment (Figure 5.5d). Figure 5.5a illustrates that the proposed method's F1-score approaches 1 in the simulated IoT testbed. Only RF achieves results near the proposed method, but its F1-score is slightly lower. In simulation with a changed environment (Figure 5.5b), F1-score of the proposed method approaches 0.9, whereas all other methods fall significantly short of this performance. In the case of the real-field dataset (Figure 5.5c), the proposed method stands out as the only approach to achieve an F1-score value exceeding 0.9. Similarly, as illustrated in Figure 5.5d, the F1-Score of the proposed method on the real-field dataset with changed environment also surpasses 0.9, while other methods yield notably lower F1-score values.

5.4.3 Quality-of-Service Analysis

The proposed method's performance is evaluated based on throughput ratio and average data latency, reflecting its QoS. These parameters are examined using the network measurement models discussed in **Sections 5.3.2 and 5.3.3**, respectively.

5.4.3.1 Throughput Ratio Performance

Throughput ratio analysis for the simulated IoT testbed and the real-field dataset is shown in Figures 5.6a and 5.6b, respectively. The proposed method compared with

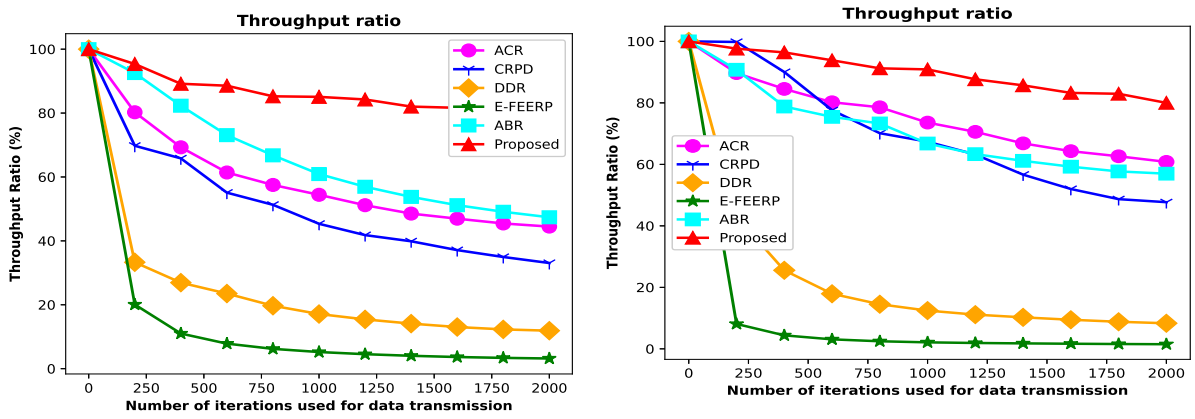


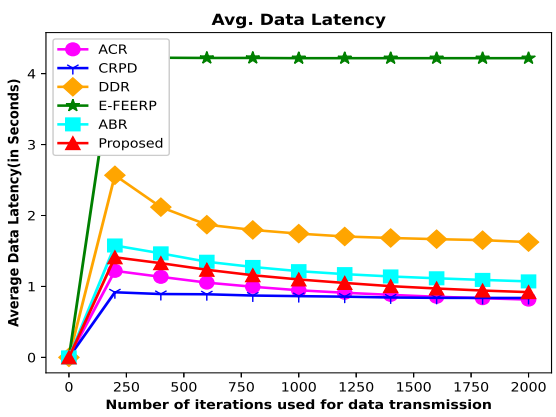
Figure 5.6: A graphic representation of the throughput ratio throughout a range of network iterations. The throughput ratio is evaluated for the proposed method across (a) simulated IoT testbed and (b) real-field dataset, and contrasted against ACR, CRPD, DDR, E-FEERP, and ABR transmission methods

the actor-critic routing algorithm without fault prediction (ACR), clustering routing protocol for dynamic wireless sensor networks (CRPD) [134], dynamic directional routing algorithm (DDR) [135], enhanced fuzzy based energy efficient routing protocol (E-FEERP) [132], and area base routing (ABR) [133]. As shown in the figures, the proposed method transfers the highest number of data packets in the simulated IoT testbed and the real-field dataset. As the proposed method predicts node fault in advance and prevents data loss by avoiding faulty nodes, it transfers approximately 85% of the total generated data packets after 2000 iterations on the simulated testbed as shown in Figure 5.6a. Whereas, under the same conditions, ACR (50%), ABR (52%), CRPD (38%), DDR (17%), and E-FEERP (7%) deliver a lesser number of packets generated by IoDs as these methods do not predict the node fault before transmitting the data. Moreover, Figure 5.6b illustrates that the proposed method successfully transfers more data packets (82%) in comparison to ACR (65%), ABR (62%), CRPD (53%),

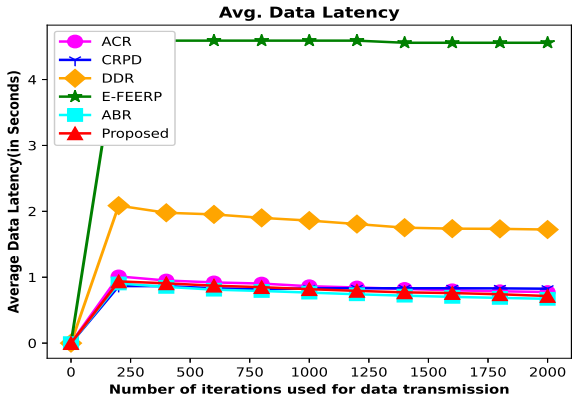
DDR (10%), and E-FEERP (4%) after 2000 iterations of transmission on the real-field dataset.

5.4.3.2 Average Data Latency Performance

Figures 5.7a and 5.7b illustrate the average data latency for all methods, separately for the simulated IoT testbed and real-field dataset. Figure 5.7a shows that the proposed



(a)



(b)

Figure 5.7: A graphic representation of the average data latency (measured in seconds) as the number of iterations changes across the network. The assessment of average data latency is conducted on proposed approach across both (a) simulated IoT testbed scenarios and (b) real-field datasets. The results are compared with ACR, CRPD, DDR, E-FEERP, ABR transmission methods

method has an average data latency of 1.1 seconds (sec) on the simulated IoT testbed at 2000 iterations which is less when compared to ABR (1.4 sec), DDR (1.9 sec), and E-FEERP (4.2 sec). CRPD, which uses a clustering protocol and hence a smaller number of hops, gives lesser data latency (1 sec), and ACR, which uses actor-critic based routing without fault prediction, also gives lesser data latency (0.8 sec). Incorporating fault prediction in the proposed method leads to the denial of the most optimal route in terms of other QoS parameters to minimize packet loss. This, in turn, leads to increased average data latency compared to CRPD and ACR. Figure 5.7b illustrates that the average data latency of the proposed method (0.7 sec) is lesser than the data latencies returned by DDR (1.9 sec) and E-FEERP (4.1 sec) over the real-field dataset

at 2000 iterations. The average data latencies returned by CRPD, ABR, and ACR are very close to the average data latency of the proposed method, which is in the range of 0.7 sec to 0.95 sec.

5.4.4 Energy efficiency Analysis

To evaluate energy efficiency, three measures are examined, i.e., network lifetime, residual energy within the network, and energy distribution across the network. These network parameters are computed using the energy consumption model outlined in **Section 5.3.4**.

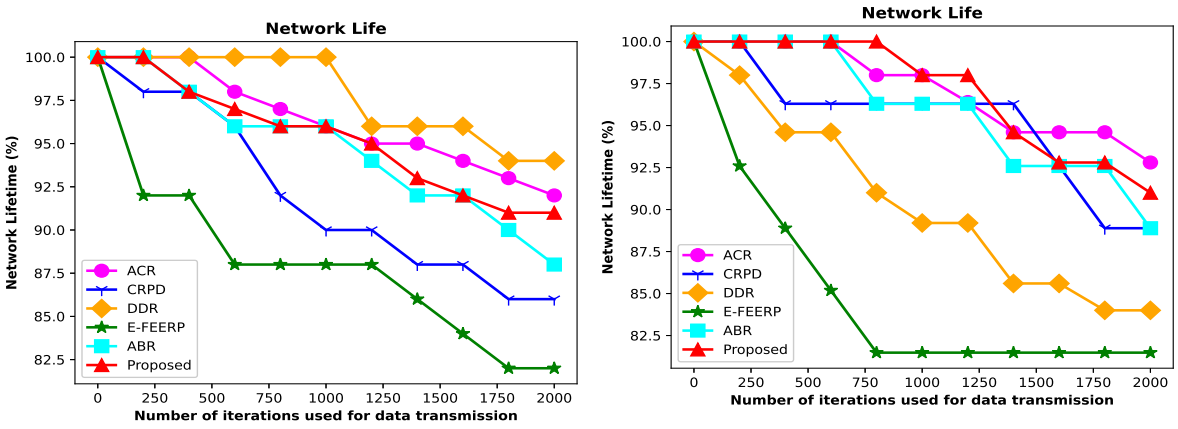


Figure 5.8: A visual representation showcasing the network lifetime across different numbers of iterations. The assessment of the overall network lifetime is conducted through the proposed method across both (a) simulated IoT testbed environments and (b) real-field datasets. The results are compared with the ACR, CRPD, DDR, E-FEERP, ABR transmission methods

5.4.4.1 Network Lifetime Performance

The network lifetime parameter gives a measure of how reliable and long-lasting a network can be. It is simply computed as the percentage of the number of alive IoDs to the total number of IoDs taken at the beginning of the network. An IoD is declared dead when its residual energy falls below a certain threshold.

Figures 5.8a and 5.8b show the network lifetime performance of the proposed method with respect to other methods over a simulated IoT testbed and real-field dataset. The

figures show that the proposed method gives better lifetime performance than CRPD, E-FEERP, and ABR in both the simulated testbed and real-field datasets. The proposed method gives 92% lifetime after 2000 iterations over simulated IoT testbed whereas ABR (82%), CRPD (86%) and E-FEERP (82%) give significantly lower lifetimes, as evident from Figure 5.8a. In the proposed method, data packets occasionally must traverse longer transmission routes to avoid faulty nodes, leading to increased involvement of IoDs and, consequently, higher energy consumption. As a result, it demonstrates a shorter lifetime compared to DDR (95%) and ACR (93%) in the simulated IoT testbed. Over the real-field dataset, as depicted in Figure 5.8b, the proposed method gives 91% lifetime performance, which is better than all the other methods, ABR (89%), CRPD (89%), DDR (84%) and E-FEERP (81%) except ACR (93%) after 2000 iterations due to the reason discussed previously.

5.4.4.2 Residual Energy Performance

The residual energy of the network is calculated as the sum of energies of all sensor nodes at that instant. Figures 5.9a and 5.9b show the residual energy variation with iterations over the simulated IoT testbed and real-field dataset, respectively. The initial energy of each sensor node is taken as 2 Joules (J). Figure 5.9a shows that the residual energy after 2000 iterations in the proposed method is 147 J, greater than all other methods except ACR (151 J). The lesser residual energy of the proposed method than ACR is due to longer transmission routes to avoid faulty nodes. When compared to CRPD (128 J), E-FEERP (119 J), ABR (104 J), and DDR (78 J), the proposed method returns significantly higher residual energy values after 2000 iterations.

Figure 5.9b illustrates the residual energy variations over the real-field dataset. After 2000 iterations, the residual energy using the proposed method is 88.5 J, whereas using CRPD (78 J), ABR (77 J), DDR (56 J), and E-FEERP (51 J), the residual energy after 2000 iterations is significantly lower. Only ACR (91 J) gives better residual energy as

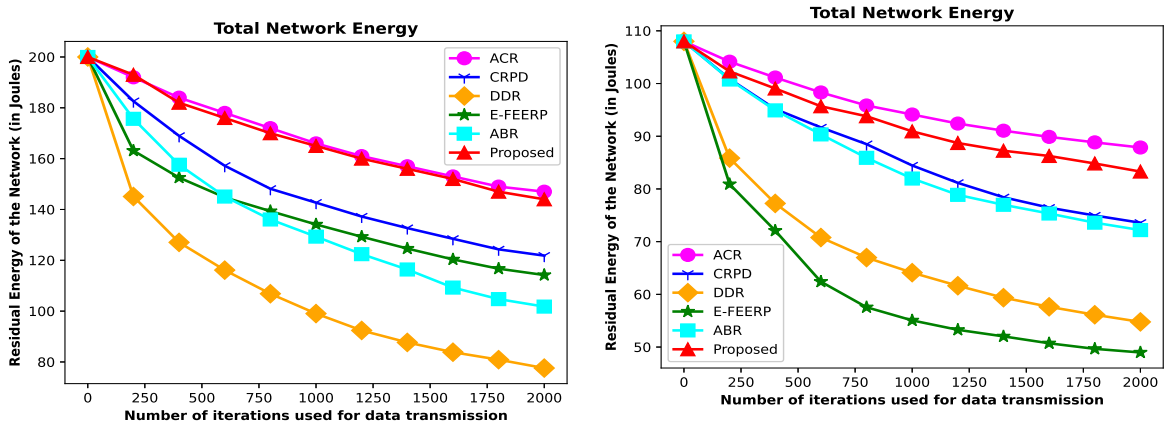
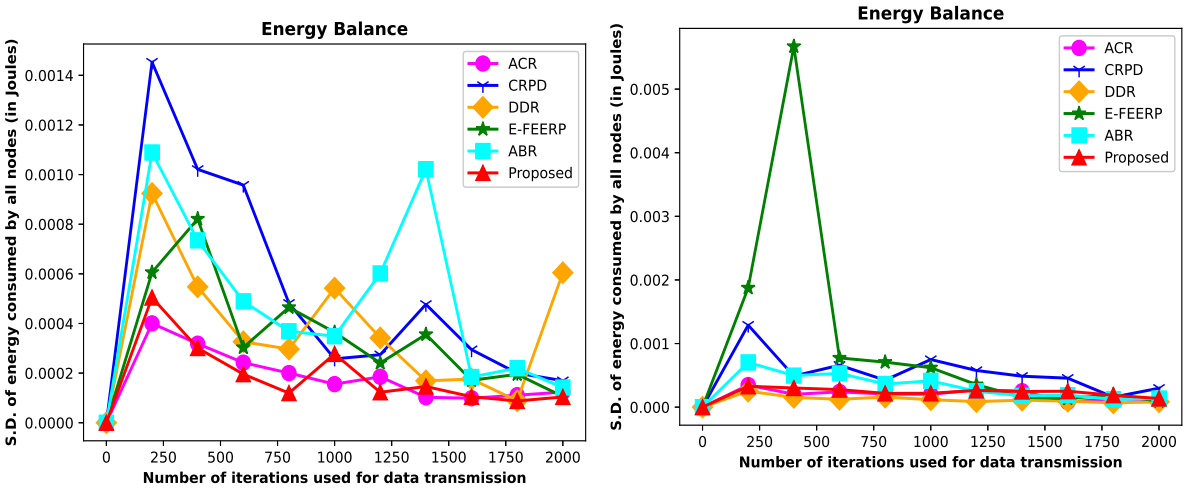


Figure 5.9: A graphic representation of total residual energy throughout a range of network iterations. This evaluation assesses the total residual energy of the network through the proposed method across (a) simulated IoT testbed scenarios and (b) real-field datasets. A comparison is drawn with ACR, CRPD, DDR, E-FEERP, and ABR transmission methods

it does not predict faulty nodes before transmitting the data.

5.4.4.3 Energy Balancing Performance

The energy balancing parameter is evaluated by measuring the standard deviation (S.D.) of energy consumption across all nodes for each iteration. Figures 5.10a and 5.10b show the energy-balancing performance on the simulated IoT testbed and real-field dataset. Figure 5.10a shows the energy balance of the proposed method maintains a constant low value over 2000 iterations on the simulated testbed. This indicates that all nodes are consuming a similar amount of energy in each iteration. Only ACR shows similar behavior in energy balancing metrics, while CRPD, ABR, E-FEERP, and DDR show irregular behavior in energy balancing metrics. Figure 5.10b illustrates the energy balancing performance over the real-field dataset. Figure 5.10b shows that E-FEERP shows irregular behavior. Whereas, the proposed method, ACR, DDR, CRPD, and ABR, maintains low constant values in energy balancing parameters, out of which the proposed method maintains more constant values.



(a) (b)

Figure 5.10: An illustration depicting the fluctuation in the standard deviation (S.D.) of energy consumption across the network as the number of iterations for data transmission varies. The variation of the S.D. of the proposed method is assessed across (a) simulated IoT testbed and (b) real-field dataset. The results are compared with the ACR, CRPD, DDR, E-FEERP, ABR transmission methods

5.4.5 Sensitivity and Convergence Analysis

5.4.5.1 Sensitivity and Convergence Analysis to Compute Optimal Learning Rate for Node Fault Prediction

A sensitivity analysis of the ADF Framework is demonstrated in Table 5.3. It illus-

Table 5.3: Sensitivity analysis of ADF Framework across simulated IoT network at different threshold values

Threshold (percentile)	Accuracy	Precision	Recall	F1-Score
80	0.91	0.89	0.99	0.94
85	0.93	0.91	0.99	0.95
95	0.97	0.96	0.98	0.97
99	0.99	0.98	0.98	0.98
100	0.98	0.98	0.99	0.98

trates a comparative analysis of various performance metrics (i.e., Accuracy, Precision, Recall, and F1-Score) at different threshold percentiles of the reconstruction loss array

during training. These percentiles represent the cutoff values above which the nodes are considered faulty. The sensitivity analysis table includes five threshold percentiles: 80th, 85th, 95th, 99th, and 100th. The highest performance is observed at the 99th percentile, with an accuracy of 0.999018, precision of 0.985788, recall of 0.989159, and an F1-score of 0.988989. The convergence curve for this approach has also been plotted for different learning rates. Figure 5.11 illustrates the validation loss of the auto-encoder

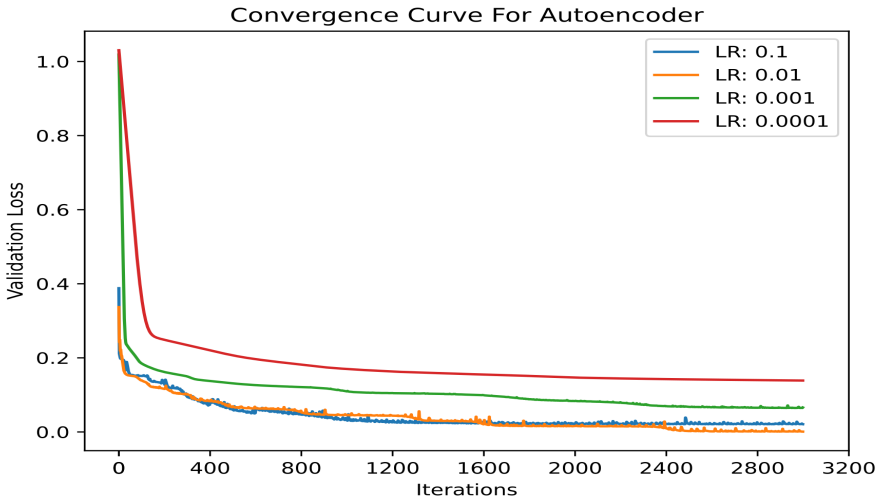


Figure 5.11: Convergence curve for auto-encoder model architecture

during training for different learning rates (0.1, 0.01, 0.001, 0.0001). The x-axis represents training iterations, and the y-axis shows the validation loss. Higher learning rates (0.1 and 0.01) demonstrate significantly better convergence and lower final validation loss compared to lower learning rates (0.001 and 0.0001). learning rate of 0.1 demonstrates the lowest validation loss compared to others, with a gradual decrease in validation loss over the iterations.

Furthermore, to assess the performance of the proposed ADF framework's classification capabilities, the confusion matrix, as demonstrated in Figure 5.12, is used. Where the rows indicate the actual fault levels, while the columns denote the predicted fault levels. The diagonal elements reflect accurate classifications, with higher counts signifying strong performance for each fault level. Conversely, the off-diagonal elements

illustrate misclassifications, revealing the types and frequencies of errors committed by the model. For example, the model accurately classified 31,117 instances as NoFault, 8,926 instances as Low, 6,548 as Mid, and 5,168 as High fault severity. Off-diagonal el-

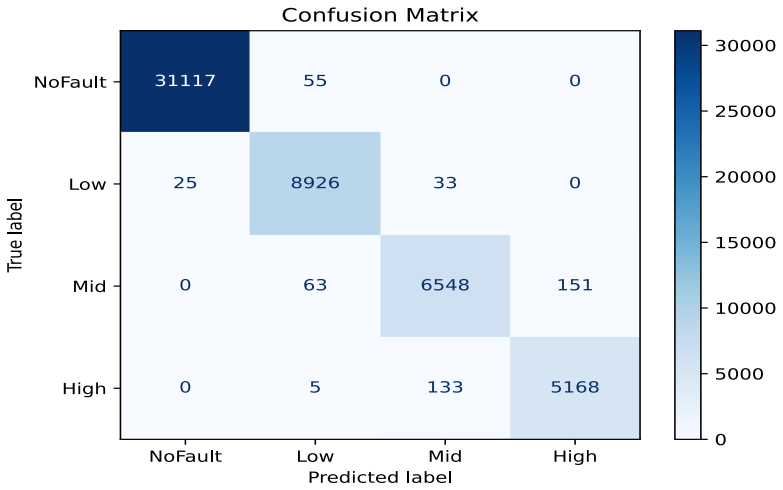


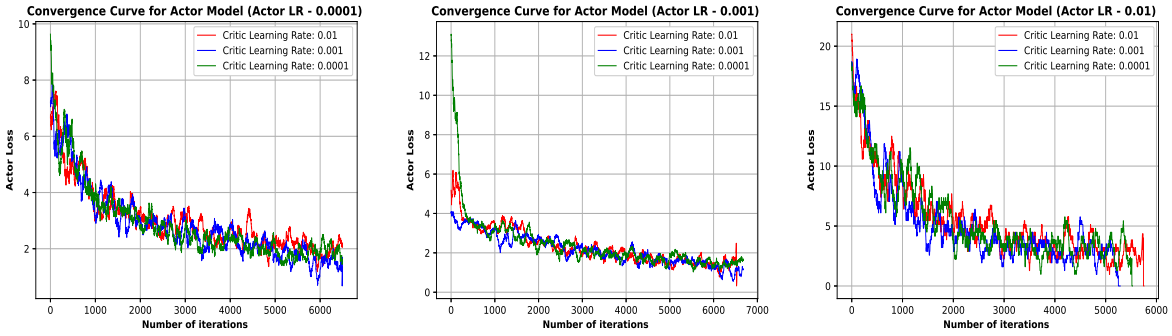
Figure 5.12: Confusion matrix for node fault prediction over simulated IoT network

ements reveal misclassifications. For instance, 55 instances with a true label of NoFault were incorrectly predicted as Low, and 63 instances of Mid fault were misclassified as Low. Overall, the confusion matrix demonstrates a strong performance for the NoFault category with a high number of true positives. The performance for Low, Mid, and High fault severities is also considerable, although there are some instances of misclassification, primarily between adjacent severity levels, which is expected in a multi-class classification problem dealing with potentially nuanced distinctions between fault levels.

5.4.5.2 Sensitivity and Convergence Analysis to Compute Optimal Learning Rate for Data Transfer

As discussed in the description of actor and critic models, the actor learns from the values predicted by the critic, and the critic learns from the action taken by the actor and the reward associated with that action.

This indicates that both the actor and critic models learn together and hence, their



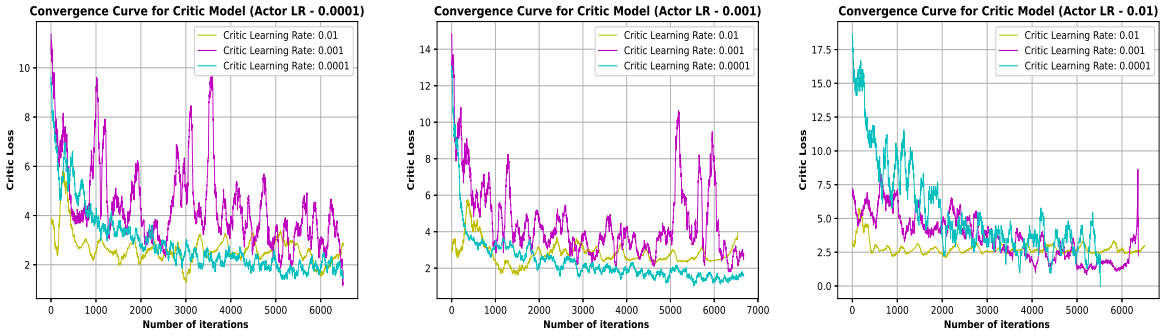
(a) (b) (c)

Figure 5.13: An illustration of training loss of the actor model for 9 possible combinations of learning rates (LRs). The convergence of loss for different combinations is compared to select the best one. Traning loss of actor model when (a) actor LR = 0.0001 and critic LR $\in \{0.0001, 0.001, 0.01\}$, (b) actor LR = 0.001 and critic LR $\in \{0.0001, 0.001, 0.01\}$, and (c) actor LR = 0.01 and critic LR $\in \{0.0001, 0.001, 0.01\}$

individual learning rates (LRs) also affect the training of each other. To select the best combination of LRs of both models, we did offline learning for 9 possible combinations and observed their respective loss convergence curves. For each combination, the offline learning was carried out for η iterations, after which the loss generally converges. The 9 combinations of LRs that we used are, $LR_{Actor} \times LR_{Critic}$, where $LR_{Actor} = \{0.0001, 0.001, 0.01\}$ and $LR_{Critic} = \{0.0001, 0.001, 0.01\}$. The convergence curve for actor loss is given in Figure 5.13, and the convergence curve for critic loss is given in Figure 5.14. A learning rate of 0.0001 for both actor and critic models returns the lowest converged values of both losses.

5.4.6 Complexity analysis of proposed Hybrid ML framework

Computational complexity serves as a vital metric for understanding how the resource requirements of an algorithm scale with increasing input size. This analysis is essential for evaluating the feasibility and efficiency of different algorithms, especially in resource-constrained or time-sensitive applications. Determining the next node for message transmission using the proposed framework in an IoT network involves several



(a) (b) (c)

Figure 5.14: An illustration of the training loss of the critic model for 9 possible combinations of LRs. The convergence of loss for different combinations is compared to select the best one. Traning loss of critic model when (a) actor LR = 0.0001 and critic LR $\in \{0.0001, 0.001, 0.01\}$, (b) actor LR = 0.001 and critic LR $\in \{0.0001, 0.001, 0.01\}$, and (c) actor LR = 0.01 and critic LR $\in \{0.0001, 0.001, 0.01\}$

stages. Initially, the calculation to identify the closest nodes takes place, contributing $O(N \log N)$ complexity, where N is the number of nodes. Subsequently, the forward pass through actor neural network takes place to process the state and determine actions (i.e., next node selection), incurring a computational cost of $O(C_{Actor})$, which is approximated as $O(L_{actor} \cdot H_{actor}^2)$ based on the number of layers (L_{actor}) and hidden units (H_{actor}) in the actor network. Following message transmission to the next node, ADF based fault detection is performed for current node, adding a complexity of $O(C_{ae})$, approximated as $O(n_{past} \cdot H_{ae}^2)$ based on the sequence length of input data (n_{past}) and the number of hidden units in the autoencoder LSTM layers (H_{ae}). Summing these steps, the overall complexity per routing decision is $O(N \log N + L_{actor} \cdot H_{actor}^2 + n\_steps \cdot H_{ae}^2)$.

5.5 Conclusions

This chapter presents a novel method for optimal data routing with node fault prediction in dynamic IoT networks. Using actor-critic reinforcement learning, the method optimizes data routing, while the Auto-Encoder Anomaly Detection with Fuzzy (ADF) framework predicts faulty nodes by analyzing sensor behavior. This combined approach

enhances network reliability by preventing data transfer through fault-prone nodes, minimizing data loss, and improving throughput. Evaluated on real-field datasets and simulated testbeds, the method outperforms existing techniques like DDR, E-FEERP, and ABR in terms of precision, recall, F1-score, and network throughput under varying conditions. The results establish the proposed framework as an effective solution for medium- and large-scale IoT networks.