Journal of Information Systems Engineering and Management
2025, 10(27s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Evaluating Fault Tolerance in Distributed Systems using Predictive Analytics with Gated Recurrent Unit and Long Short-Term Memory Models

Faizan Ahmad[1], Mohd Haroon[2], Zeeshan Ali Siddiqui[3]
[1] Ph. D candidate, Department of Computer Science and Engineering, Integral University, Lucknow, India
[2] Professor, Department of Computer Science and Engineering, Integral University, Lucknow, India
[3] Associate Professor, Department of Computer Science and Engineering, University of Lucknow, Lucknow, India
* Corresponding Author: fahmad@iul.ac.in

ARTICLE INFO

Received: 18 Dec 2024

Revised: 15 Feb 2025

Accepted: 28 Feb 2025

ABSTRACT

Fault tolerance is crucial for ensuring reliability in distributed systems, where minor disruptions can cascade into significant failures, causing downtimes, productivity loss, and financial damage. The complexity and interdependencies of distributed systems make them particularly prone to faults. Designing robust fault-tolerant mechanisms is therefore essential to cater the reliability demands of modern systems. Predictive analytics has become a game-changing approach, transitioning from managing faults reactively to detecting and preventing them proactively. This study examines the integration of Gated Recurrent Units (GRU) and Long Short-Term Memory (LSTM), into predictive analytics frameworks to enhance fault tolerance in distributed systems. GRUs efficiently process sequential data, whereas LSTMs are particularly adept at capturing long-term dependencies, making them well-suited for analyzing historical fault patterns. The proposed approach leverages these models to identify critical failure indicators and predict faults with high accuracy. By enabling early detection and response to potential failures, the models prevent disruptions from escalating. Experimental results demonstrate that GRU and LSTM-based models significantly reduce unexpected downtimes through precise fault predictions. Real-time monitoring capabilities further enhance decision-making and preemptive fault-handling processes, ensuring system reliability and performance. This study highlights the practical application of GRU and LSTM models in advancing fault tolerance in distributed environments. By offering a data-driven solution, the research improves fault prediction accuracy, strengthens system resilience, and enhances operational efficiency, addressing key challenges in distributed system management.

Keywords: Fault tolerance, distributed system, deep learning, system reliability, Gated Recurrent Unit, Long Short-Term Memory

INTRODUCTION

Distributed systems have become the cornerstone of modern computing infrastructure, enabling a wide array of applications, from cloud services to large-scale data processing and real-time analytics. As these systems scale in size and complexity, ensuring their fault tolerance has turn out to be a top priority. The increasing reliance on distributed systems has their demerits also like even minor faults, such as hardware failures, software bugs, network disruptions, or unexpected spikes in workloads, can lead to significant operational issues, including downtime, data loss, and financial repercussions [1]. The need for robust fault tolerance mechanisms is therefore crucial to maintaining the reliability, efficiency, and performance of these systems [2]. Traditional faulttolerance techniques, such as redundancy, check-pointing, and failover strategies, have long been employed to mitigate the impact of system failures. These approaches are effective in some contexts but often face limitations in large, dynamic environments where workloads and system conditions constantly evolve. For instance, traditional redundancy techniques require maintaining extra copies of resources, which can lead to inefficiencies and higher operational costs. Similarly, check-pointing strategies that save system states at regular intervals can be computationally expensive and may not be capable of handling more complex or unpredictable failure scenarios in real-time. As distributed systems become

more complex, the scalability and efficiency of these traditional fault-tolerance methods are often insufficient, creating a pressing requirement for proactive solutions. Predictive analytics enables fault tolerance by foreseeing potential failures and allowing preemptive action [3]. Predictive analytics uses historical data to detect patterns and enable proactive fault response. By using datadriven models to predict potential disruptions, predictive analytics reduces the reliance on reactive fault-handling techniques, thus improving reliability and minimizing downtime of system. Among the various predictive 2 models available, deep learning methods, specifically Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) networks, have proven to be instrumental for fault prediction in distributed systems [4]. These models excel at processing time-series data like system logs and performance metrics. GRUs and LSTMs excel at capturing complex temporal patterns that can indicate the existence of emerging faults, even in noisy or incomplete data [5]. The GRU is an RNN optimized for efficient sequential data processing. It uses gating mechanisms to focus on the most relevant input data [6]. GRUs have been demonstrated to act efficiently to tasks involving sequential data, particularly when the data contains long-term dependencies. This is crucial in the context of distributed systems, where failures may emerge over extended periods and may be influenced by a range of system variables, such as hardware performance, network latency, and workload fluctuations. The LSTM networks are another type of RNN that is particularly adept at handling long-term dependencies in sequential data [7]. LSTMs were specifically designed to cater the vanishing gradient problem, which occurs in standard RNNs when learning from long sequences. By using memory cells and gates, LSTMs are able to retain information over longer periods, enabling them to capture more complex and subtle patterns in the data. This ability to maintain and process long-term dependencies makes LSTM an ideal choice for analysing system metrics over time, as many fault events in distributed systems exhibit delayed or gradual symptoms before a failure occurs. In distributed systems, GRU and LSTM models boost fault tolerance by detecting early failure warnings. By continuously analysing system parameters and performance metrics, these methods may recognize anomalies that signal potential concerns, such as abnormal CPU usage, memory leaks, or network congestion. Prompt findings of such issues allow for timely interventions, such as redistributing workloads, adjusting resource allocations, or initiating failover mechanisms, thereby preventing more severe system failures and minimizing downtime. As distributed systems breed large volume of data, often in real-time, traditional statistical methods may struggle to identify subtle correlations or interactions between variables. In contrast, deep learning models (DLMs) efficiently analyze large datasets, revealing hidden patterns [8]. However, the execution of predictive analytics in distributed systems also presents several challenges. One of the primary hurdles is the collection and preparation of data. For predictive models to be effective, they require vast amounts of high-quality, labelled data, which may be difficult to obtain in practice. Training and choosing the right models are another difficulty to implement DLMs. Training GRU and LSTM models is resource-intensive, particularly with large datasets [9]. Additionally, The training process must be managed to prevent over fitting, as DLMs may memorize training data instead of generalizing. Deployment and integration of predictive models into existing fault management workflows also pose challenges. Predictive models must be integrated to monitor performance in real-time and trigger corrective actions when needed. This involves developing interfaces for data collection, anomaly detection, and decision-making, ensuring models scale efficiently [10]. Despite these challenges, the use of GRU and LSTM-based predictive analytics in distributed systems has shown great promise in enhancing fault tolerance [11].

RELATED WORK

In fault-tolerant systems, interpretability is crucial as it allows engineers to comprehend the reasoning behind a model's predictions, particularly when identifying faults [12]. By understanding which factors contributed most to a specific prediction, engineers can verify the model's accuracy, ensuring its outputs are reliable and actionable. This transparency fosters trust in the system and enables effective root cause analysis, which is essential for preventing failures and improving system performance. Without interpretability, it becomes challenging to assess whether the model is making correct assumptions, potentially leading to misdiagnosis or missed opportunities to address critical issues proactively. Explainability techniques like SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations) are good tools to considerate the inner workings of machine learning models [13 and 14]. These methods analyze and quantify the contributions of individual input features to a model's output, offering insights into how predictions are made. For instance, if a fault prediction is driven by factors such as spikes in memory utilization or high CPU load, SHAP or LIME can identify and highlight these specific influences. By providing a clearer view of the decisionmaking process, these techniques enhance trust in the model and support effective troubleshooting and optimization.

Attention mechanisms may be incorporated into GRU [15] and LSTM networks to increase their interpretability and performance. Attention mechanisms assign weights to key parts of the input sequence, improving prediction accuracy and revealing patterns or anomalies that precede faults. This enhances both fault diagnosis and preventive actions. In proactive fault mitigation, accurately predicting potential faults is just the first step; the system must also be equipped with mechanisms to respond preemptively and minimize impact [16]. Once a fault is anticipated, the system can take automated actions such as migrating workloads to healthier nodes to maintain service continuity, restarting affected services to restore functionality, or activating redundancy protocols to ensure failover systems are operational. These measures prevent minor issues from becoming critical, improving overall reliability [17, 18]. By combining prediction with effective response strategies, the system can maintain high availability and ensure seamless user experiences even under stress. Automation platforms like Ansible and 3 Terraform and orchestration tools like Kubernetes may automate pre-configured recovery actions. This minimizes human intervention, accelerates fault response times, and reduces operational costs, ensuring efficient and seamless recovery processes in the event of potential issues or system failures. Lin et al. (2022) recommended time series-based groundwater level forecasting using GRU [19]. Wang et al., (2018) suggested short-term load forecasting with multi-source data using GRU neural networks [20]. Farah et al., (2022) proposed gated recurrent unit DLM for wind power prediction [21]. Hai et al. (2022) proposed use of GRU for hard disk drive (HDD) failure prediction [22]. Le et al. (2019) suggested an application of LSTM neural network for flood forecasting [23]. Assis et al. (2021) recommended a GRU deep learning system to cater the attacks in software defined networks [24]. Escalation protocols in predictive fault tolerance systems are designed to prioritize and respond to issues based on their severity. For instance, when a minor fault is predicted, the system may simply log the event or flag it for continuous monitoring, requiring no immediate action. However, for more critical predictions, the system can initiate automatic corrective actions, such as activating failover mechanisms, or escalate the issue by notifying on-call engineers for immediate intervention. This tiered approach ensures efficient resource allocation and swift handling of critical faults. The use of predictive analytics and DLMs, especially GRU [25, 26] and LSTM networks, for fault tolerance in distributed systems has garnered growing research interest because of their capacity to analyze temporal data and predict system failures accurately. This literature review explores significant studies and developments in the area of predictive fault tolerance, highlighting the application of deep learning techniques for fault detection and mitigation. It delves into the role of GRUs and LSTMs in identifying complex failure patterns from historical data, emphasizing their effectiveness in enhancing system reliability and operational efficiency.

Fault Tolerance in Distributed Systems

Traditional fault tolerance methods in distributed systems, such as replication, check-pointing, and failover mechanisms, have been extensively studied and widely applied. These techniques, as described in the seminal work of [27], are designed to ensure system reliability and continuity in the face of hardware and software failures. Replication involves maintaining multiple copies of critical data or services, check-pointing periodically saves system states for recovery, and failover mechanisms redirect operations to backup systems during failures. Together, these methods have formed the backbone of fault tolerance strategies in distributed environments. However, as Bennani and Menasce (2005) pointed out, these traditional approaches have notable drawbacks, particularly in large-scale distributed systems [28]. They are often resource-intensive, requiring substantial computational and storage overhead, and can become inefficient as systems scale. Moreover, their reactive nature means they address failures only after they occur, which can lead to latency, downtime, or undetected failures, especially in systems with unpredictable workloads. These challenges have highlighted the limitations of traditional methods, paving the way for proactive, predictive fault-tolerance strategies. By leveraging advanced analytics and machine learning, these strategies aim to anticipate potential failures, enabling systems to take preemptive actions that reduce downtime and improve overall efficiency. Tao et al. (2021) suggested GRUbased parallel network traffic anomaly detection using subagging ensembles [29].

Predictive Analytics for Fault Detection

Predictive analytics prevents system failures by analyzing historical data to identify patterns of future issues [30]. For example, Canizo et al. (2017) demonstrated the efficacy of predictive models in industrial IoT systems [31, 32]. Their study showed that by detecting anomalies early, organizations could significantly reduce unplanned maintenance and minimize system downtimes, leading to increased operational efficiency. Similarly, Mahmud et al. (2018) explored the uses of predictive analytics in cloud environments, focusing on forecasting resource failures [33].

Their research highlighted how predictive models could anticipate critical events such as hardware degradation or network bottlenecks, allowing for proactive interventions that improve service reliability. Both studies emphasize time-series analysis for predicting system failures, but traditional models struggle with complex, non-linear dependencies and long-term patterns. These shortcomings make it challenging to address dynamic workloads and multifaceted failure scenarios effectively. DLMs, including LSTMs and GRUs, provide a stronger alternative. These models excel at learning intricate patterns over time, making them better suited for accurately forecasting failures in both IoT and cloud-based distributed systems [34- 36].

Application of GRU and LSTM in Time-Series Fault Prediction

GRU and LSTM networks have gained widespread recognition for their ability to effectively model sequential and time-dependent data, making them highly suitable for fault prediction tasks [37, 38]. Both networks are designed to capture the temporal dependencies present in time-series data, which is essential for predicting system failures that unfold over time. LSTM, introduced by [39, 40], was developed as an extension of traditional recurrent neural 4 networks [41] and was specifically designed to cater the vanishing gradient problem. This problem often hinders traditional RNNs from learning long-term dependencies, especially in tasks like fault prediction, where gradual degradation signals can be subtle and span long periods. LSTM's ability to maintain and update memory over extended timeframes enables it to detect these long-term dependencies effectively. Novaes et al. (2020) suggested anomaly detection and mitigation in software-defined network environment using LSTM and fuzzy logic [42, 43].

Malhotra et al. (2015) further validated the efficacy of LSTM in the area of time-series anomaly detection [44]. They applied LSTM to sensor data from industrial systems and successfully identified subtle shifts in the data that could indicate impending failures. This work was pivotal in establishing LSTM's potential for predictive maintenance, especially in systems characterized by non-linear and complex data patterns. LSTM's ability to model complex time-series relationships is a key to accurate predictive fault tolerance and failure forecasting.

Deep Learning for Fault Prediction in Distributed Systems

Recent research has increasingly focused on leveraging DLMs, particularly GRUs and LSTMs, for fault tolerance in distributed computing environments [45, 46]. These models have shown significant promise in predicting and mitigating faults by analyzing time-series data, which is common in such systems. Yadav et al. (2024) applied LSTM networks to sentiment analysis in performance logs, where they effectively captured the sequential dependencies in logs generated by distributed systems [47]. Their approach demonstrated the potential of LSTMs to uncover patterns in system behaviour that may indicate emerging faults or performance issues, yielding promising results in both accuracy and predictive capability. Similarly, Rihi et al. (2024) developed an LSTM-based model for vibration forecasting and predictive maintenance in mining grinding mills [48]. Their work demonstrated significant improvements in prediction accuracy and the early detection of failures by analyzing vibration data over time. The LSTM model effectively identified subtle anomalies that were indicative of impending failures, leading to timely maintenance actions and reducing costly downtimes. These studies validate the efficacy of GRU and LSTM networks in fault tolerance applications within distributed computing environments. These models adapt to complex fault patterns, making them ideal for proactive detection and mitigation in dynamic environments.

Comparative Studies and Challenges

Recent studies by [49, 50] have provided valuable insights into the comparative performance of various deep learning approaches for fault prediction in distributed systems. Both studies evaluated models such as GRUs, LSTMs, and Convolutional Neural Networks (CNNs) to determine their suitability for fault detection in complex environments. The findings reveal that LSTMs excel in detecting temporal patterns and long-term dependencies but have limitations. Specifically, LSTMs require significant training times and large datasets to achieve high accuracy, and their internal workings can be difficult to interpret, limiting their usability in environments where transparency is crucial. In contrast, CNNs showed promise in identifying spatial patterns but were less effective in capturing the sequential dependencies essential for fault prediction in dynamic systems. Despite their advantages, LSTMs and GRUs still face challenges related to data requirements and model complexity. Further emphasizing this, Ma et al. (2024) underscored the critical need for interpretability in deep learning-based fault prediction models [51]. For distributed systems, especially in mission-critical applications, understanding how a model arrives at its predictions is essential for effective fault management. Transparent models can help engineers make informed decisions, respond proactively, and enhance trust in automated fault detection systems. Real-time fault prediction is essential for

ensuring the reliability and stability of distributed systems, where immediate detection and response to faults can prevent system-wide failures. Recent research has focused on adapting LSTM networks for real-time fault prediction, particularly in resource-constrained environments. Fan et al. (2024) presented a framework that integrates real-time data streams for fault prediction in distributed sensor networks [52]. Their work demonstrated how LSTM models could be effectively implemented in these settings, adapting to the continuous flow of data while maintaining high prediction accuracy. The key challenge in these applications is the need for low-latency processing and minimal resource consumption, as distributed systems often operate under tight computational and memory constraints. Fan et al.'s framework showcased the ability of LSTM to make predictions on-the-fly, facilitating rapid decision-making and preventing cascading failures that can occur if faults are not detected and addressed quickly. This real-time adaptability is crucial for maintaining system performance and reliability in dynamic, large-scale environments.

This review traces the evolution of fault tolerance strategies from traditional, reactive methods to modern, predictive analytics-based approaches. Earlier techniques, such as replication, check-pointing, and failover mechanisms, primarily focused on responding to failures after they occurred, often with significant system overhead. By leveraging time-series data and identifying patterns indicative of potential issues, these models allow for timely interventions, reducing downtime and improving system reliability. However, despite these advancements, several challenges remain. The large volume of data generated in distributed systems can overwhelm traditional processing methods, and the complexity of training DLMs demands substantial 5 computational resources. Additionally, the interpretability of these DLMs is yet a critical concern, as understanding how predictions are made is essential for trust and effective decision-making. Moving forward, further research is needed to address these issues, refining GRU and LSTM models to enhance their applicability and reliability in fault-tolerant systems.

METHODOLOGY

Four essential steps of proposed framework for the fault detection in distributed systems are data collection, feature selection, pre-processing, and model training. Different inputs from dispersed systems are gathered during the data gathering phase in order to find possible flaws. Important sources of information include system logs, which document specific events and transactions; performance measurements, which show the health of the system and include CPU load, memory utilization, and I/O rates; and environmental metrics, which can affect performance and include network latency and power status. Raw data is cleaned up and organized into a machine learning-ready state during the pre-processing stage. This includes labelling observations according to fault or non-fault statuses, processing missing data, normalizing data to a common scale, and encoding categorical variables into numeric representations. After that, feature selection reduces the complexity of the dataset and identifies the most important predictors of system failures. Recursive feature elimination (RFE), correlation analysis, and domain expertise are used to identify relevant features. The dataset is typically divided 70/30 into training and test sets. By combining these stages, a thorough framework for anticipating and identifying distributed system failures is created, improving the systems' dependability and operational effectiveness. Flowchart of proposed framework is shown in Figure 1.

Data Collection

The initial stage involves gathering information from dispersed systems, including system logs and other performance metrics that may point to possible issues. Important resources include system logs, which offer thorough documentation of system activities and events, and performance metrics, which show the general health of the system and include CPU load, memory utilization, and I/O rates. Furthermore, because these contextual elements may have a significant effect on system performance, environmental measurements like network latency and power status are taken into account. When taken as a whole, these data sources offer a thorough basis for examining and resolving possible problems in distributed system operations.

Data Pre-processing

Data preparation ensures input data is clean, organized, and ready for model training. Handling missing data by locating and filling in gaps to preserve data integrity is one of the many important responsibilities involved in this procedure. By standardizing variables to a single scale, data normalization improves model performance. Encoding converts categorical variables into numerical formats for machine learning algorithms. Data labeling assigns target labels to observations, enabling supervised learning and accurate fault prediction.

Feature Selection

Feature selection identifies key predictors of system failures, reduces dataset dimensionality, and improves model effectiveness. Correlation analysis examines statistical relationships between variables to identify key connections. Another technique is called RFE, in which the most predictive subset is isolated by iteratively removing features. Furthermore, by applying specific knowledge to select and refine pertinent aspects, domain expertise plays a critical role. By working together, these strategies make sure that the dataset concentrates on the most important factors, improving machine learning models' performance and accuracy.

Model Training

The dataset is typically split 70/30 into training and test sets for balanced evaluation. After that, labelled data trains GRU and LSTM models to identify trends and accurately predict system failures.

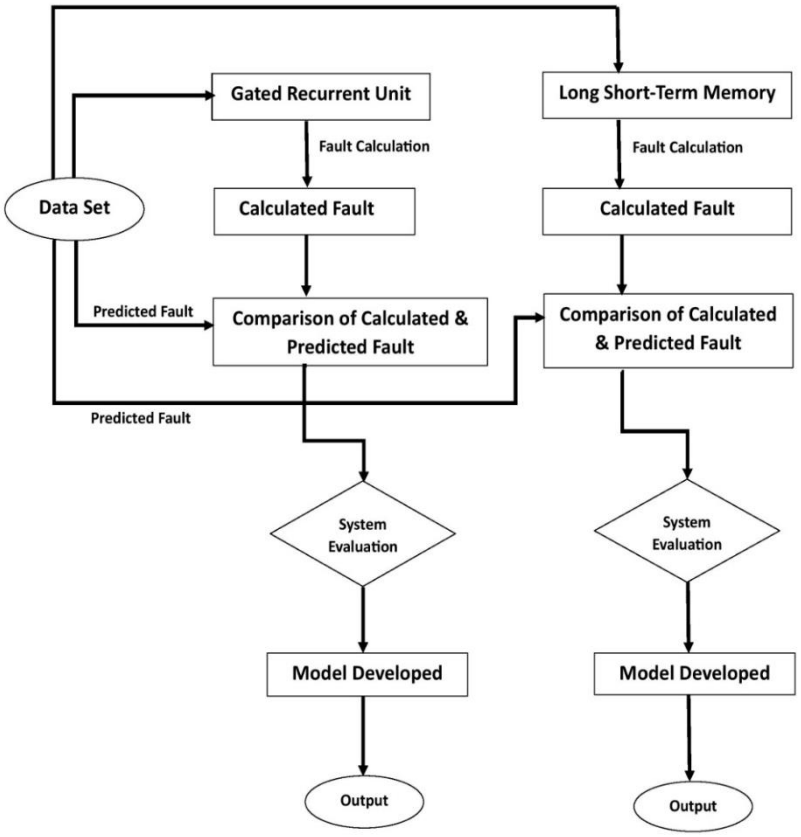

Figure 1: Flowchart of proposed framework

Gated Recurrent Unit (GRU)

A GRU [53, 54] is a type of RNN that is intended to alleviate the vanishing gradient issue that is frequently observed in conventional RNNs while efficiently capturing relationships in sequential data, such as text, time series, or audio. GRUs use two specialized gates, the update gate and the reset gate, to accomplish this. The reset gate regulates the addition of new data, whereas the update gate establishes the amount of historical data that should be kept. By constantly balancing historical and present data, this gating mechanism enables GRUs to learn and understand sequential patterns efficiently.

Model Evaluation

A variety of performance metrics designed to handle the problem of imbalanced data, a prevalent problem in fault detection because system faults are usually infrequent, are used to thoroughly assess the models on the test set following training. Measures used include:

• Accuracy: Measures the overall proportion of correctly classified instances, providing a general performance overview.

• Precision: Measures the true positive fault predictions among all instances classified as faults, highlighting prediction reliability.

• Recall (Sensitivity): Represents the proportion of actual faults correctly identified by the model, reflecting its detection ability.

• F1-Score: The harmonic means of precision and recall, offering a balanced assessment of false positives and negatives.

• Receiver Operating Characteristic (ROC) Curve: It graphically shows the trade-off between true and false positive rates, with AUC as a measure of model discrimination.

These metrics collectively provide a inclusive assessment of model performance, enabling a nuanced evaluation of its effectiveness in detecting rare faults within complex systems. Table 1 shows multiple traits and their descriptions.

Table 1: Traits and their description

Traits	Description	Data Type	Example
CPU Usage (%)	Percentage of CPU usage at the given timestamp.	float	75.3
Memory Usage (%)	Percentage of memory utilization at the given timestamp.	float	68.5
Network Latency(ms)	Network latency in milliseconds.	float	15.2
Packet Loss (%)	Percentage of packets lost in the network during communication.	float	0.5
Request_Rate (req/s)	Number of incoming requests handled per second.	int	325
Error Count	Number of errors logged during this time period.	int	3
Network_Throughput (MB/s)	Data throughput in megabytes per second across the network.	float	10.3
Fault	Target label indicating whether a fault occurred (1 for fault, 0 for no fault).	binary	1 (Fault) or 0

Prospective investigations should prioritize the formation of more efficient and scalable machine learning models tailored for predictive fault tolerance. Advanced deep learning architectures like RNNs and CNNs offer new opportunities for analyzing time-series data and system logs. Furthermore, developing resilient hybrid models that combine traditional machine learning with modern techniques could greatly improve fault prediction accuracy and reliability in distributed systems. By embracing these cutting-edge techniques and integrative approaches, future research can substantially elevate the efficacy and robustness of fault tolerance mechanisms, thereby ensuring more resilient and dependable distributed infrastructures.

Working of GRU

The GRU is an advanced type of RNN that uses a gating mechanism to efficiently control the input flow. By solving the vanishing gradient issue that conventional RNNs frequently face, it enables the network to identify long-term dependencies in sequential data. Two essential gates used by GRUs are the update gate and the reset gate. These gates enable the network to more effectively learn temporal patterns by dynamically controlling its memory. By investigating the mathematical formulas that underlie these gates, we may understand more about how GRUs function and perform very well while processing sequential input.

A. Input Sequence and Initial Hidden States

- A sequence of input vectors $x_1, x_2 \ldots\ldots, x_t$, where T is the number of time steps in the sequence.

- h_0: Initial hidden state (usually initialized to zero).

The GRU processes each time step t to produce an updated hidden state h_t, which summarizes the sequence information up to that step.

B. GRU Cell Computations

At each time step t, the GRU cell performs the following computations:

Update Gate (zt)

The update gate looks how much of the preceding state (ht−1) wishes to be carried forward to the current state. This helps the model decide whether to retain the information/sdata from the previous state or replace it with new information

Formula:

$$z_t = \sigma(W_z \cdot x_t + U_z \cdot h_{t-1} + b_z)$$

Where:

- x_t: The input vector at time step t.
- h_{t-1}: The hidden state from the previous time step.
- W_z: Weight matrix for the input xt in the update gate.
- U_z: Weight matrix for the previous hidden state ht−1 in the update gate.
- b_z: Bias term for the update gate.
- σ: Sigmoid activation function, which outputs values between 0 and 1.
- The sigmoid function constrains zt between 0 and 1, which allows it to act as a soft "gate":
- When z_t is close to 1, the network retains a significant amount of past information from h_{t-1}.
- When z_t is close to 0, the network focuses more on new information from x_t.

Reset Gate (rt)

The reset gate controls how much of the previous hidden state should be ignored. This gate helps the GRU to forget irrelevant parts of the previous state when processing the current input.

Formula:

$$r_t = \sigma(W_r \cdot x_t + U_r \cdot h_{t-1} + b_r)$$

Where:

- W_r: Weight matrix for the input x_t in the reset gate.
- U_r: Weight matrix for the previous hidden state ht−1 in the update gate.
- b_r: bias term for the update gate.

Candidate Hidden State ($\tilde{h}_t$)

The candidate hidden state is calculated based on the reset gate r_t, the input x_t, and the previous hidden state h_{t-1}. It determines the new content that will be added to the current hidden state

Formula:

$$\tilde{h}_t = tanh(W_h \cdot x_t + U_h \cdot (r_t \odot h_{t-1}) + b_h)$$

Where:

- W_h: Weight matrix for the input x_t in calculating the candidate hidden state.
- U_h: Weight matrix for the previous hidden state h_{t-1} in calculating the candidate hidden state.
- b_h: Bias term for the candidate hidden state.
- $(r_t \odot h_{t-1})$: Element-wise multiplication (denoted by $\odot$) of the reset gate r_t and the previous hidden state h_{t-1}.
- tanh: Hyperbolic tangent activation function, which outputs values between -1 and 1.
- The reset gate r_t controls how much of the previous hidden state influences the candidate hidden state:
- If r_t is close to 0, the effect of h_{t-1} is minimized, and $\tilde{h}_t$ is more influenced by x_t.
- If r_t is close to 1, the candidate hidden state $\tilde{h}_t$ incorporates both h_{t-1} and x_t.

Final Hidden State (h_t)

The final hidden state h_t is the output of the GRU cell at the current time step. It combines the previous hidden state h_{t-1} and the candidate hidden state $\tilde{h}_t$ controlled by the update gate z_t.
Formula:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

Where:

- $z_t \odot h_{t-1}$: Part of the previous hidden state retained by the update gate.
- $(1 - z_t) \odot \tilde{h}_t$: Part of the candidate hidden state that is combined with h_{t-1}.
- This formula shows that:
- When z_t is close to 1, h_t is heavily influenced by h_{t-1}, meaning the GRU "remembers" more past information.
- When z_t is close to 0, h_t relies more on the candidate hidden state $\tilde{h}_t$, meaning the GRU updates its memory with more new information from x_t.

The GRU cell repeats these steps for each time step t, ultimately producing a sequence of hidden states. For classification tasks, we typically use only the final hidden state h_t as a summary of the entire sequence.

C. Classification Layer

The final hidden state h_t from the last time step represents the entire sequence. To generate a classification output, we pass h_t through a fully connected (dense) layer with a softmax (for multiclass classification) or sigmoid (for binary classification) activation function.

Output Layer (Multiclass Classification)

Let K be the number of classes.

- Weights and biases of the output layer: Let W_{out} and b_{out} be the weight matrix and bias vector for the output layer.

The logits z for each class can be computed as:

$$z = W_{out} \cdot h_t + b_{out}$$

The final class probabilities $\hat{y}$ are obtained by applying the softmax activation to the logits:

$$\hat{y}k = \frac{exp(z_k)}{\Sigma_{j=1}^{K} exp(z_j)} \; for \; k = 1,2, \ldots \ldots \ldots, K$$

Output Layer (Binary Classification)

For binary classification, we use a sigmoid activation, which outputs a single probability score $\hat{y}$ between 0 and 1.

$$\hat{y} = \sigma(W_{out} \cdot h_t + b_{out})$$

where σ is the sigmoid function, defined as:

$$\sigma(x) = \frac{1}{1 + exp(-x)}$$

D. Loss Function

To train the GRU for classification, we use an appropriate loss function based on the type of classification task:

- **Binary Cross-Entropy Loss** (for binary classification):

$$Loss = -\frac{1}{N}\sum_{i=1}^{N} (y^{(i)} log(\hat{y}^{(i)}) + (1 - y^{(i)}) log(1 - \hat{y}^{(i)}))$$

- **Categorical Cross-Entropy Loss** (for multiclass classification):

$$Loss = -\frac{1}{N}\sum_{1=1}^{N}\sum_{k=1}^{N} y_k^{(i)}\, log\left(\hat{y}_k^{(i)}\right)$$

where N is the number of training samples, $\boldsymbol{y}^{(i)}$ is the true label of the i-th sample, and $\hat{y}^{(i)}$ or $\hat{y}_k^{(i)}$ is the predicted probability for that sample.

Working of Long Short-Term Memory (LSTM)

To use an LSTM for classification, we need to adapt its final hidden states to produce a categorical output. In a typical LSTM model for sequence classification, the LSTM processes the input sequence, and its last hidden state is fed through a dense layer with a softmax (or sigmoid) activation function, depending on the classification task (multiclass or binary). Let's go through each step in building the LSTM mathematical model for classification.

A. Input Sequence and Initial Hidden States

- A sequence of input vectors x_1, x_2....., x_t, where T is the number of time steps in the sequence.
- h_0: Initial hidden state (usually initialized to zero).
- C_0: Initial cell state (usually initialized to zero).

For each time step t, the LSTM processes the input x_t and produces an updated hidden state h_t and cell state C_t.

B. LSTM Cell Computations

At each time step t, the LSTM cell performs the following operations, as derived from the LSTM formulas

Forget Gate (f_t)

The forget gate decides what part of the previous cell state C_{t-1} to "forget" or retain. This gate helps the LSTM selectively keep or discard information over time.

$$f_t = \sigma\left(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f\right)$$

The forget gate outputsf_t where:

- f_t close to 1 means retaining information in the cell state.
- f_t close to 0 means discarding information in the cell state.

Input Gate (i_t)

The input gate decides which parts of the current input x_t will be used to update the cell state.

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i)$$

Candidate Cell State ($\tilde{C}_t$)

The candidate cell state ($\tilde{C}_t$) is a potential new addition to the cell state based on the current input x_t and previous hidden state h_{t-1}. It captures the "new information" that could be added to the cell state.

$$\tilde{C}_t = tanh(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c)$$

The input gate i_t and candidate cell state $\tilde{C}_t$ work together to update the cell state.

Cell State Update (C_t)

The cell state C_t is the LSTM's memory. The cell state is updated by combining the previous cell state C_{t-1}, modified by the forget gate f_t, with the candidate cell state $\tilde{C}_t$, scaled by the input gate i_t.

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

This equation shows that:

- The previous cell state C_{t-1} is selectively kept or discarded through f_t.
- The candidate cell state $\widetilde{C}_t$ is added based on the input gate i_t, allowing new information to influence the cell state.

Output Gate (o_t)

The output gate o_t controls the part of the cell state that is output as the hidden state h_t. The hidden state is used both as the LSTM's output for this time step and as input to the next step.

$$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o)$$

Hidden State Update (h_t)

The hidden state h_t is calculated by applying the output gate o_t to the updated cell state C_t (after passing it through a tanh activation function).

$$h_t = o_t \odot tanh(C_t)$$

This hidden state h_t serves as both the output for the current time step and the input for the next time step, carrying information forward in the sequence.

Where:

- x_t: The input vector at time step t.
- h_{t-1}: The hidden state from the previous time step.
- C_{t-1}: The cell state from the previous time step.
- W, U, and b: Weight matrices and biases for each gate
- σ: Sigmoid activation function, which outputs values between 0 and 1.
- tanh: Hyperbolic tangent activation function, which outputs values between -1 and 1.
- $f_t \odot C_{t-1}$: Retained memory from the previous cell state, controlled by the forget gate.
- $i_t \odot \tilde{C}_t$: New memory added to the cell state, controlled by the input gate and candidate cell state.
- o_t: Controls how much of the cell state to output.
- $tanh(C_t)$: Squashes the updated cell state to a range between -1 and 1.

The LSTM iterates through these equations for each time step t in the sequence, producing a hidden state h_t at each step.

C. Classification Layer

For classification, we typically use the final hidden state h_t, which contains information summarizing the entire sequence. This final hidden state h_t is passed through a dense (fully connected) layer with a softmax activation for multiclass classification or a sigmoid activation for binary classification.

Output Layer (Multiclass Classification)

Let's assume we have K classes.

- **Weights and biases of the output layer:** Let W_{out} and b_{out} be the weight matrix and bias vector for the output layer.

The logits z for each class can be calculated as:

$$z = W_{out} \cdot h_t + b_{out}$$

The final class probabilities $\hat{y}$ are obtained by applying the softmax activation to the logits:

$$\hat{y}k = \frac{exp(z_k)}{\Sigma_{j=1}^{K} exp(z_j)} \; for \; k = 1,2, \ldots \ldots \ldots, K$$

Output Layer (Binary Classification)

For binary classification, we use a sigmoid activation, which outputs a single probability score $\hat{y}$ between 0 and 1.

$$\hat{y} = \sigma(W_{out} \cdot h_t + b_{out})$$

where σ is the sigmoid function, defined as:

$$\sigma(x) = \frac{1}{1+ exp(-x)}$$

D. Loss Function

For training the LSTM model, we use a suitable loss function for classification:

- **Binary Cross-Entropy Loss** (for binary classification):

$$Loss = -\frac{1}{N}\sum_{i=1}^{N}(y^{(i)} log(\hat{y}^{(i)}) + (1 - y^{(i)}) log(1 - \hat{y}^{(i)}))$$

- **Categorical Cross-Entropy Loss** (for multiclass classification):

$$Loss = -\frac{1}{N}\sum_{1=1}^{N}\sum_{k=1}^{N} y_k^{(i)} log\left(\hat{y}_k^{(i)}\right)$$

where N is the number of training samples, $\boldsymbol{y^{(i)}}$ is the true label of the i-th sample, and $\hat{y}^{(i)}$ or $\hat{y}_k^{(i)}$ is the predicted probability for that sample.

RESULTS

The dataset (Table 2) encompasses a range of features, including performance metrics and logs, along with a target label that indicates whether a fault occurred (Fault = 1) or not (Fault = 0). Data is typically collected at regular intervals (e.g., every minute or 10 seconds) over a defined period, capturing both normal operations and failure events. Table 3 and Table 5 showcase the training datasets for the GRU and LSTM models, respectively, while Table 4 and Table 6 display the testing datasets for these models. These tables serve as a foundation for evaluating the predictive performance of each model. Table 3 and Table 5 show the training dataset GRU and LSTM respectively whereas Table 4 and Table 6 show the testing dataset GRU and LSTM respectively.

Table 2: Multiple feature dataset

CPU Usage (%)	Memory Usage %)	Network Latency(ms)	Packet Loss (%)	Request_Rate (req/s)	Error Count	Network_Throughput (MB/s)	Fault
75.3	68.5	15.2	0.5	325	3	10.3	0
80.1	70.2	16.3	0.3	400	5	11.2	1
65.2	60.4	14.8	0.1	290	2	9.8	0
60.5	60.8	15.1	0.2	310	2	9.7	0
72.9	64.3	16.2	0.4	335	4	10.9	0
84.2	71.5	17.4	0.6	415	6	12.2	1
89	75	18.6	0.8	440	8	13	1
66.9	59.7	14.7	0.1	280	1	9.3	0
74.6	65.9	16	0.3	350	5	11.1	0

83.7	70.8	17.2	0.5	410	6	12	1
77.2	68.2	16.8	0.6	390	4	11.7	0
91.4	78.5	19.1	0.9	460	9	13.4	1
69.3	61.2	15.3	0.3	320	2	10.1	0
75.8	66.5	16.5	0.4	345	4	11.4	0
86.3	73.2	18	0.7	425	7	12.8	1
62.8	58	14.5	0.1	290	1	9.6	0
79.4	67.1	16.7	0.5	375	5	11.5	0
88.7	76	18.4	0.8	450	8	12.9	1
67.5	60.1	14.9	0.2	275	2	9.8	0
52.6	56.2	14.3	0.2	256	5	12.5	1
89.6	58.6	12.3	0.3	247	4	11.6	0
45.2	76.4	15	0.8	269	9	13.7	0
68.2	78.5	20.3	0.7	354	7	14.8	0
56.5	64.8	25	0.1	159	2	11.5	1
55.9	69.3	25	9	157	3	10.5	1
47.6	78.8	12	0.8	341	9	6.9	1
25.69	45.96	14	0.6	356	1	13.6	0
78.4	87.5	13	0.4	452	6	17.5	1
56.5	69.6	16	3	560	8	9.6	1

Table 3: Training dataset GRU

CPU Usage (%)	MemoryUsage %)	Network Latency(ms)	Packet Loss (%)	Request_Rate (req/s)	Error Count	Network_Throughput (MB/s)	Fault
56.5	64.8	25	0.1	159	2	11.5	1
47.6	78.8	12	0.8	341	9	6.9	1
67.5	60.1	14.9	0.2	275	2	9.8	0
45.2	76.4	15	0.8	269	9	13.7	0
75.3	68.5	15.2	0.5	325	3	10.3	0
72.9	64.3	16.2	0.4	335	4	10.9	0
89	75	18.6	0.8	440	8	13	1
56.5	69.6	16	3	560	8	9.6	1
83.7	70.8	17.2	0.5	410	6	12	1
69.3	61.2	15.3	0.3	320	2	10.1	0
80.1	70.2	16.3	0.3	400	5	11.2	1
74.6	65.9	16	0.3	350	5	11.1	0
65.2	60.4	14.8	0.1	290	2	9.8	0
84.2	71.5	17.4	0.6	415	6	12.2	1

Table 4: Testing dataset GRU

CPU Usage (%)	MemoryUsage %)	Network Latency(ms)	Packet Loss (%)	Request_Rate (req/s)	Error Count	Network_Throughput (MB/s)	Fault
78.4	87.5	13	0.4	452	6	17.5	1
86.3	73.2	18	0.7	425	7	12.8	1
62.8	58	14.5	0.1	290	1	9.6	0
52.6	56.2	14.3	0.2	256	5	12.5	1
79.4	67.1	16.7	0.5	375	5	11.5	0
91.4	78.5	19.1	0.9	460	9	13.4	1
66.9	59.7	14.7	0.1	280	1	9.3	0
25.69	45.96	14	0.6	356	1	13.6	0
68.2	78.5	20.3	0.7	354	7	14.8	0
77.2	68.2	16.8	0.6	390	4	11.7	0
60.5	60.8	15.1	0.2	310	2	9.7	0
75.8	66.5	16.5	0.4	345	4	11.4	0
55.9	69.3	25	9	157	3	10.5	1
89.6	58.6	12.3	0.3	247	4	11.6	0
88.7	76	18.4	0.8	450	8	12.9	1

Table 5: Training dataset LSTM

CPU Usage (%)	MemoryUsage %)	Network Latency(ms)	Packet Loss (%)	Request_Rate (req/s)	Error Count	Network_Throughput (MB/s)	Fault
65.2	60.4	14.8	0.1	290	2	9.8	0
69.3	61.2	15.3	0.3	320	2	10.1	0
52.6	56.2	14.3	0.2	256	5	12.5	1
89.6	58.6	12.3	0.3	247	4	11.6	0
74.6	65.9	16	0.3	350	5	11.1	0
86.3	73.2	18	0.7	425	7	12.8	1
80.1	70.2	16.3	0.3	400	5	11.2	1
62.8	58	14.5	0.1	290	1	9.6	0
72.9	64.3	16.2	0.4	335	4	10.9	0
78.4	87.5	13	0.4	452	6	17.5	1
45.2	76.4	15	0.8	269	9	13.7	0
75.3	68.5	15.2	0.5	325	3	10.3	0
60.5	60.8	15.1	0.2	310	2	9.7	0
91.4	78.5	19.1	0.9	460	9	13.4	1

Table 6: Testing dataset LSTM

CPU Usage (%)	MemoryUsage %)	Network Latency(ms)	Packet Loss (%)	Request_Rate (req/s)	Error Count	Network_Throughput (MB/s)	Fault
25.69	45.96	14	0.6	356	1	13.6	0
66.9	59.7	14.7	0.1	280	1	9.3	0
55.9	69.3	25	9	157	3	10.5	1
56.5	69.6	16	3	560	8	9.6	1
84.2	71.5	17.4	0.6	415	6	12.2	1
83.7	70.8	17.2	0.5	410	6	12	1
68.2	78.5	20.3	0.7	354	7	14.8	0
77.2	68.2	16.8	0.6	390	4	11.7	0
75.8	66.5	16.5	0.4	345	4	11.4	0
56.5	64.8	25	0.1	159	2	11.5	1
89	75	18.6	0.8	440	8	13	1
47.6	78.8	12	0.8	341	9	6.9	1
67.5	60.1	14.9	0.2	275	2	9.8	0
88.7	76	18.4	0.8	450	8	12.9	1
79.4	67.1	16.7	0.5	375	5	11.5	0

Following the submission of the proposed framework, a confusion matrix for the GRU model was generated. Figure 2 presents this confusion matrix, while Figure 3 illustrates the logistic regression graphs. The performance metrics were calculated as follows: precision = TP / (TP + FP) = 0.86, accuracy = (TP + TN) / (TP + TN + FP + FN) = 0.80, recall = TP / (TP + FN) = 0.75, and F1-score = 2 * (precision * recall) / (precision + recall) = 0.80. These metrics provide a comprehensive evaluation of the model's predictive performance.

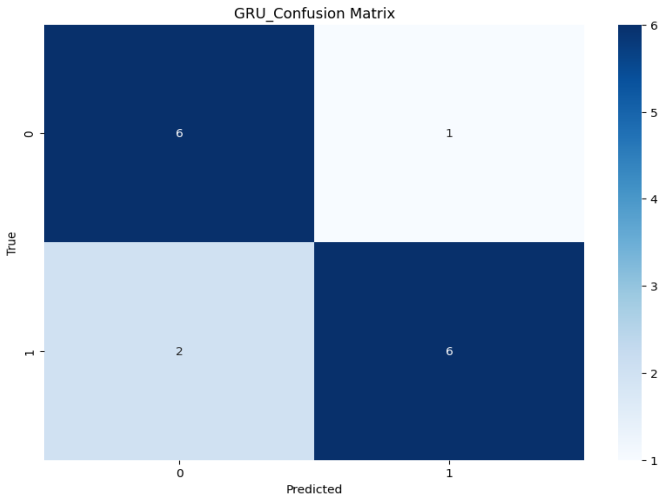

Figure 2: GRU Confusion Matrix

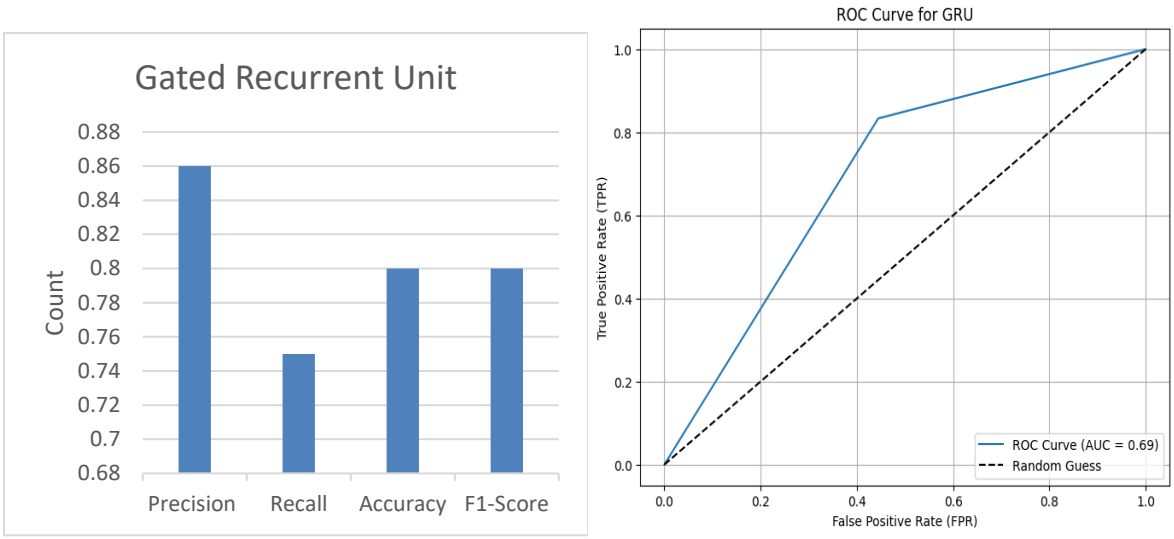

Figure 3: Gated Recurrent Unit Graphs

After implementing the proposed framework, the confusion matrix for the LSTM model was generated. Figure 4 displays the LSTM confusion matrix, while Figure 5 presents the LSTM graphs. The calculated performance metrics are as follows: precision = TP / (TP + FP) = 0.83, accuracy = (TP + TN) / (TP + TN + FP + FN) = 0.87, recall = TP / (TP + FN) = 0.83, and F1-score = 2 * (precision * recall) / (precision + recall) = 0.83. These results offer a detailed assessment of the LSTM model's performance.

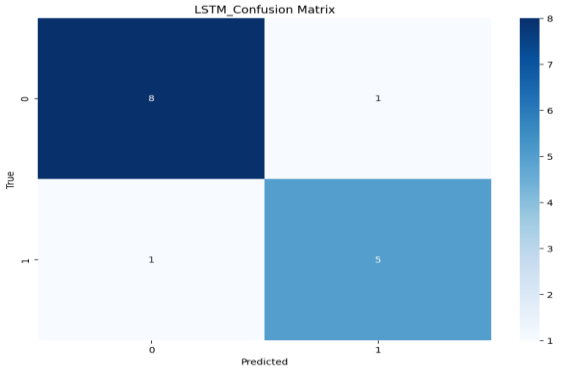

Figure 4: LSTM Confusion Matrix

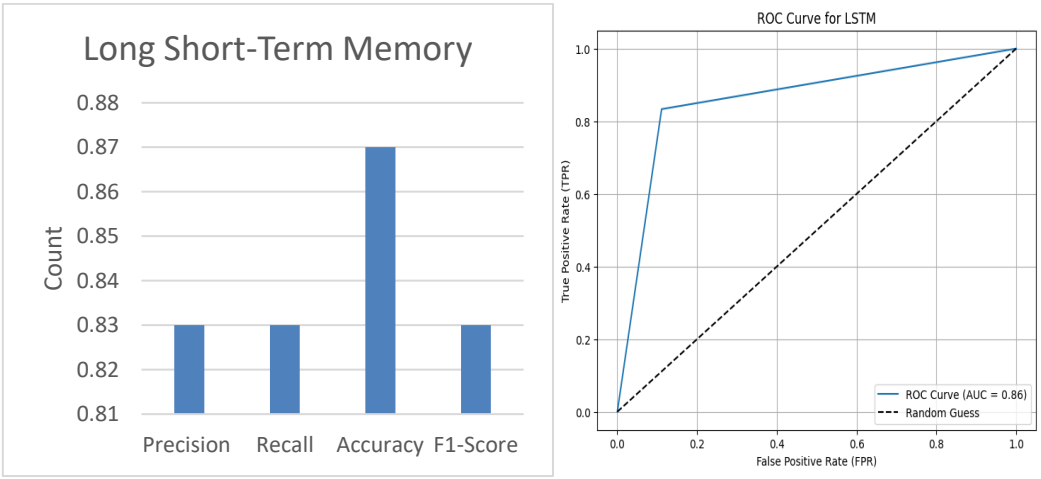

Figure 5: LSTM Graph

COMPARATIVE STUDY OF GRU AND LSTM

The comparative analysis of the two methods is presented in Figure 6. Logistic Regression is a probabilistic model that estimates the likelihood of faults based on input features. One of its key strengths is interpretability, as it provides easily understandable probabilities, allowing for clear decision-making. Another advantage is its simplicity; the model is computationally less complex, making it faster to train on smaller or less intricate datasets compared to more advanced models. Logistic Regression is ideal for estimating outcome probabilities, especially with a linear relationship between input features and log-odds. It is ideal for applications requiring efficient, interpretable models, particularly with simpler or smaller datasets.

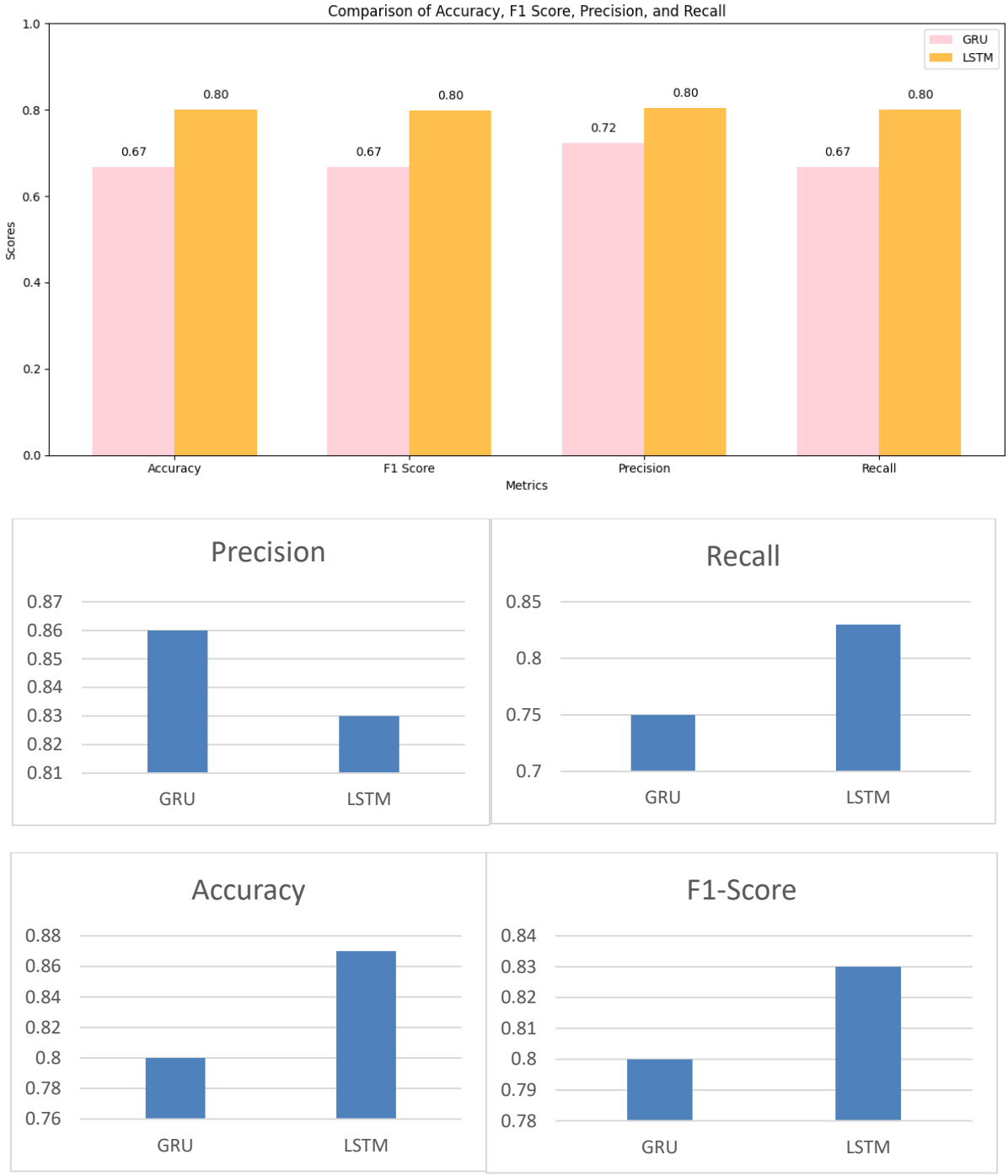

Figure 6: Performance Matrix

The comparison graph clearly illustrates the superior performance of the LSTM model over the GRU model across all key metrics: accuracy, F1 score, precision, and recall. The LSTM model consistently achieves a score of 0.80 in

each of these metrics, demonstrating its strong ability to precisely predict faults within a distributed system. This uniformity across all metrics emphasizes LSTM's effectiveness in capturing complex patterns and dependencies, making it a highly reliable option for fault tolerance applications where both precision and recall are critical. The model's performance across multiple metrics demonstrates its robustness and suitability for precise fault detection.

In contrast, the GRU model demonstrates lower scores, with accuracy, F1 score, and recall all at 0.67, while precision shows a modest improvement at 0.72. Although the GRU's simpler architecture offers computational efficiency, its relatively lower performance suggests that it may fail to capture certain fault patterns that the LSTM model can identify. This trade-off between computational efficiency and predictive performance makes the GRU less reliable in applications where fault detection is crucial. Ultimately, the LSTM model emerges as the more robust and dependable choice for systems requiring high fault tolerance and proactive fault detection. Its ability to capture complex patterns and consistently deliver strong results makes it the preferred model for fault prediction in distributed systems.

DISCUSSION

The results of the proposed framework reveal notable differences between the performance of the GRU and LSTM models for fault detection in distributed systems. The confusion matrices for both models, shown in Figures 2 and 4, provide a clear picture of their respective classification behaviors. The GRU model demonstrates a precision of 0.86, accuracy of 0.80, recall of 0.75, and an F1-score of 0.80, which are respectable but indicate a need for improvement in capturing certain fault patterns. While precision is relatively high, the recall suggests that the GRU model misses some fault instances, leading to a moderate decrease in overall performance. On the other hand, the LSTM model outperforms the GRU across all key metrics. With a precision of 0.83, accuracy of 0.87, recall of 0.83, and an F1-score of 0.83, the LSTM model demonstrates more consistent and balanced performance. These results suggest that the LSTM is better at both detecting faults and minimizing false positives, offering a more reliable fault detection mechanism. The higher recall and F1-score, in particular, highlight LSTM's ability to correctly identify fault instances while maintaining a low rate of false alarms. This is crucial for fault tolerance applications, where both precision (minimizing false positives) and recall (maximizing fault detection) are of equal importance. The comparative analysis in Figure 6 underscores these differences, emphasizing the LSTM model's superior performance. The GRU's lower accuracy, F1 score, and recall (all around 0.67) indicate its limitations in handling the complexity of fault detection tasks within distributed systems. While GRU offers computational efficiency and simpler architecture, these advantages come at the cost of predictive accuracy. This trade-off between speed and performance is significant when fault detection is a critical application, where high accuracy and recall are essential for ensuring system reliability.

LIMITATIONS AND THREAT TO VALIDITY

Despite promising results, the study has key limitations and potential validity threats. First, the models were evaluated on a specific dataset, and their performance may vary with different distributed systems or fault conditions. The dataset may not cover all fault scenarios, limiting the findings' generalizability. Additionally, while the LSTM model consistently outperformed the GRU in terms of predictive accuracy, It is computationally expensive and may not suit real-time applications requiring quick inference. Moreover, the evaluation metrics used (precision, accuracy, recall, and F1-score) provide a comprehensive overview of the models' performance but do not capture all aspects of their behavior. For instance, metrics such as AUC-ROC or precision-recall 20 curves might provide more deep insight into the models' performance, especially in imbalanced datasets where fault events are rare. Finally, the comparison between the models assumes equal importance for precision and recall, but in some contexts, one of these metrics might be prioritized over the other, which could influence the selection of the most suitable model. The LSTM model proves to be the more robust and effective choice for fault detection in distributed systems, the study's findings are not without limitations. Future work should aim to address these limitations by testing the models on more diverse datasets, considering additional performance metrics, and exploring the trade-offs between model complexity and real-time operational requirements.

CONCLUSION

Integrating GRU and LSTM models for predictive fault tolerance strengthens the reliability of distributed systems by shifting from reactive fault handling to proactive prevention. These models predict potential failures, enabling systems to reduce downtime, optimize resource utilization, and minimize the impact of faults on endusers. For instance, if an LSTM model forecasts network latency, the system can take preemptive action, such as rerouting traffic

or adjusting bandwidth, to maintain service continuity. Additionally, predictive fault tolerance helps extend the lifespan of system components by reducing stress on hardware and software. Our results suggested that the LSTM model outperforms the GRU model across all key metrics, demonstrating superior accuracy and reliability for fault detection in distributed systems. Its ability to capture complex patterns makes it the preferred choice for high fault tolerance and proactive fault prediction. By addressing faults proactively, the need for emergency repairs and intensive use of backup systems is minimized, leading to more stable operations. Over time, this results in better overall system performance and continuous availability, even in the face of failures. GRU models, with their computational efficiency, are suitable for real-time applications with simpler fault patterns, while LSTM models, capable of capturing complex dependencies, excel in systems requiring detailed fault prediction. Together, these models enable early anomaly detection, resource adjustment, and preemptive repairs. This approach reduces downtime, boosts performance, and strengthens fault tolerance, moving from reactive to proactive fault management.

COMPETING INTERESTS AND FUNDING

The authors have no relevant financial or non-financial interests to disclose.

ACKNOWLEDGMENTS

Manuscript Communication Number (MCN): IU/R&D/2025-MCN0003462 office of research and development, Integral University, Lucknow.

REFERENCES

[1]. Khan, M., & Haroon, M. (2023). Artificial Neural Network-based Intrusion Detection in Cloud Computing using CSE-CIC-IDS2018 Datasets. In 2023 3rd Asian Conference on Innovation in Technology (ASIANCON) (pp. 1-4). IEEE. https://doi.org/10.1109/ASIANCON58793.2023.10269948

[2]. Tiwari, R. G., Haroon, M., Tripathi, M. M., Kumar, P., Agarwal, A. K., & Jain, V. (2024). A System Model of Fault Tolerance Technique in Distributed System and Scalable System Using Machine Learning. In Software-Defined Network Frameworks (pp. 1-16). CRC Press. https://doi.org/10.1201/9781040018323

[3]. Haroon, M., Siddiqui, Z. A., Husain, M., Ali, A., & Ahmad, T. (2024). A Proactive Approach to Fault Tolerance Using Predictive Machine Learning Models in Distributed Systems. Int. J. Exp. Res. Rev, 44, 208-220. https://doi.org/10.52756/ijerr.2024.v44spl.018

[4]. Ibrahim, M. S., Abbas, W., Waseem, M., Lu, C., Lee, H. H., Fan, J., & Loo, K. H. (2023). Long-Term Lifetime Prediction of Power MOSFET Devices Based on LSTM and GRU Algorithms. Mathematics, 11(15), 3283. https://doi.org/10.3390/math11153283

[5]. Alazab, M., Khan, S., Krishnan, S. S. R., Pham, Q. V., Reddy, M. P. K., & Gadekallu, T. R. (2020). A multidirectional LSTM model for predicting the stability of a smart grid. Ieee Access, 8, 85454-85463. https://doi.org/10.1109/ACCESS.2020.2991067

[6]. Zhou, J., Liu, K., Zhao, J., Wang, Q., Jin, C., Pan, X., ... & Chen, P. (2024). Open-Circuit Fault Diagnosis and Analysis for Integrated Charging System Based on Bidirectional Gated Recurrent Unit and Attention Mechanism. IEEE Transactions on Instrumentation and Measurement. https://doi.org/10.1109/TIM.2024.348156221

[7]. Wang, H., Yang, Z., & Yu, Q. (2017). Online reliability prediction via long short term memory for service-oriented systems. In 2017 IEEE International Conference on Web Services (ICWS) (pp. 81-88). IEEE. https://doi.org/10.1109/ICWS.2017.19

[8]. Seba, A. M., Gemeda, K. A., & Ramulu, P. J. (2024). Prediction and classification of IoT sensor faults using hybrid deep learning model. Discover Applied Sciences, 6(1), 9. https://doi.org/10.1007/s42452-024- 05633-7

[9]. Munir, H. S., Ren, S., Mustafa, M., Siddique, C. N., & Qayyum, S. (2021). Attention based GRU-LSTM for software defect prediction. Plos one, 16(3), e0247444. https://doi.org/10.1371/journal.pone.0247444

[10]. Khan, W., & Haroon, M. (2022). An efficient framework for anomaly detection in attributed social networks. International Journal of Information Technology, 14(6), 3069-3076. https://doi.org/10.1007/s41870-022-01044-2

[11]. Mateus, B. C., Mendes, M., Farinha, J. T., Assis, R., & Cardoso, A. M. (2021). Comparing LSTM and GRU models to predict the condition of a pulp paper press. Energies, 14(21), 6958. https://doi.org/10.3390/en14216958

[12]. Peng, Y., Shao, H., Yan, S., Wang, J., Xiao, Y., & Liu, B. (2024). A systematic review on interpretability research of intelligent fault diagnosis models. Measurement Science and Technology. https://doi.org/10.1088/1361-6501/ad99f4

[13]. Antwarg, L., Miller, R. M., Shapira, B., & Rokach, L. (2021). Explaining anomalies detected by autoencoders using Shapley Additive Explanations. Expert systems with applications, 186, 115736. https://doi.org/10.1016/j.eswa.2021.115736

[14]. Zafar, M. R., & Khan, N. (2021). Deterministic local interpretable model-agnostic explanations for stable explainability. Machine Learning and Knowledge Extraction, 3(3), 525-541. https://doi.org/10.3390/make3030027

[15]. Lent, D. M. B., Novaes, M. P., Carvalho, L. F., Lloret, J., Rodrigues, J. J., & Proença, M. L. (2022). A gated recurrent unit deep learning model to detect and mitigate distributed denial of service and portscan attacks. IEEE Access, 10, 73229-73242. https://doi.org/10.1109/ACCESS.2022.3190008

[16]. Mukwevho, M. A., & Celik, T. (2018). Toward a smart cloud: A review of fault-tolerance methods in cloud systems. IEEE Transactions on Services Computing, 14(2), 589-605. https://doi.org/10.1109/TSC.2018.2816644

[17]. Siddiqui, Z. A., & Haroon, M. (2024). Ranking of components for reliability estimation of CBSS: an application of entropy weight fuzzy comprehensive evaluation model. International Journal of System Assurance Engineering and Management, 1-15. https://doi.org/10.1007/s13198-024-02263-5.

[18]. Singh, V., Pandey, D., Sahu, K., Khan, M. W., Optimizing the Impact of Security Attributes in Requirement Elicitation Techniques using FAHP, International Journal of Innovative Technology and Exploring Engineering, Volume-9, Issue-4, pp.1656 1661, 2020.

[19]. Lin, H., Gharehbaghi, A., Zhang, Q., Band, S. S., Pai, H. T., Chau, K. W., & Mosavi, A. (2022). Time series-based groundwater level forecasting using gated recurrent unit deep neural networks. Engineering Applications of Computational Fluid Mechanics, 16(1), 1655-1672. https://doi.org/10.1080/19942060.2022.2104928

[20]. Wang, Y., Liu, M., Bao, Z., & Zhang, S. (2018). Short-term load forecasting with multi-source data using gated recurrent unit neural networks. Energies, 11(5), 1138. https://doi.org/10.3390/en11051138

[21]. Farah, S., Humaira, N., Aneela, Z., & Steffen, E. (2022). Short-term multi-hour ahead country-wide wind power prediction for Germany using gated recurrent unit deep learning. Renewable and Sustainable Energy Reviews, 167, 112700. https://doi.org/10.1016/j.rser.2022.112700

[22]. Hai, Q., Zhang, S., Liu, C., & Han, G. (2022). Hard disk drive failure prediction based on gru neural network. In 2022 IEEE/CIC International Conference on Communications in China (ICCC) (pp. 696-701). IEEE. https://doi.org/10.1109/ICCC55456.2022.9880794

[23]. Le, X. H., Ho, H. V., Lee, G., & Jung, S. (2019). Application of long short-term memory (LSTM) neural network for flood forecasting. Water, 11(7), 1387. https://doi.org/10.3390/w11071387 22

[24]. Assis, M. V., Carvalho, L. F., Lloret, J., & Proença Jr, M. L. (2021). A GRU deep learning system against attacks in software defined networks. Journal of Network and Computer Applications, 177, 102942. https://doi.org/10.1016/j.jnca.2020.102942

[25]. Javaid, N., Qasim, U., Yahaya, A. S., Alkhammash, E. H., & Hadjouni, M. (2022). Non-technical losses detection using autoencoder and bidirectional gated recurrent unit to secure smart grids. IEEE Access, 10, 56863-56875. https://doi.org/10.1109/ACCESS.2022.3171229

[26]. Saurabh, P., Velmurugan, K., N, N., Patange, G., Mohan Raj, G. B., Amarendra, C., & Kumar Reddy, M. V. (2024). Intelligent controller design and fault prediction for renewable energy sources using bidirectional GRU and GEO Methods. Electric Power Components and Systems, 52(2), 277-291. https://doi.org/10.1080/15325008.2023.2218368

[27]. Van Steen, M., & Tanenbaum, A. S. (2017). Distributed systems (p. 20). Leiden, The Netherlands: Maarten van Steen.ISBN: 978-90-815406-2-9

[28]. Bennani, M. N., & Menasce, D. A. (2005). Resource allocation for autonomic data centers using analyticperformance models. In Second international conference on autonomic computing (ICAC'05) (pp. 229-240). IEEE. DOI: 10.1109/ICAC.2005.50

[29]. Tao, X., Peng, Y., Zhao, F., Yang, C., Qiang, B., Wang, Y., & Xiong, Z. (2021). Gated recurrent unitbased parallel network traffic anomaly detection using subagging ensembles. Ad Hoc Networks, 116, 102465. https://doi.org/10.1016/j.adhoc.2021.102465

[30]. Selmy, H. A., Mohamed, H. K., & Medhat, W. (2024). A predictive analytics framework for sensor data using time series and deep learning techniques. Neural Computing and Applications, 36(11), 6119-6132. https://doi.org/10.1007/s00521-023-09398-9

[31]. Canizo, M., Onieva, E., Conde, A., Charramendieta, S., & Trujillo, S. (2017). Real-time predictive maintenance for wind turbines using Big Data frameworks. In 2017 IEEE international conference on prognostics and health management (icphm) (pp. 70-77). IEEE. DOI: 10.1109/ICPHM.2017.7998308

[32]. Ansar, S.A., Arya, S., Soni, N. et al. Architecting lymphoma fusion: PROMETHEE-II guided optimization of combination therapeutic synergy. Int. j. inf. tecnol. (2024). https://doi.org/10.1007/s41870-024-02194-1

[33]. Mahmud, M., Kaiser, M. S., Hussain, A., & Vassanelli, S. (2018). Applications of deep learning and reinforcement learning to biological data. IEEE transactions on neural networks and learning systems, 29(6), 2063-2079. DOI: 10.1109/TNNLS.2018.2790388

[34]. Khan, M., & Haroon, M. (2023). Detecting Network Intrusion in Cloud Environment through Ensemble Learning and Feature Selection Approach. SN Computer Science, 5(1), 84. https://doi.org/10.1007/s42979-023-02390-z

[35]. Parveen, N., Khan, M. W., Proposed Algorithm and Models for Sentiment Analysis and Opinion Mining Using Web Data, Nanotechnology Perceptions, Vol.20, No.6, pp. 1-11, 2024.

[36]. Haroon, M., Misra, D. K., Husain, M., Tripathi, M. M., & Khan, A. (2023). Security issues in the internet of things for the development of smart cities. In Advances in Cyberology and the Advent of the Next-Gen Information Revolution (pp. 123-137). IGI Global. https://doi.org/10.4018/978-1-6684-8133-2.ch007

[37]. Ansar, S.A., Kumar, S., Khan, M.W., Yadav, A., Khan, R.A., "Enhancement of Two-Tier ATM Security Mechanism: Towards Providing a Real-Time Solution for Network Issues", International Journal of Advanced Computer Science and Applications, Vol.11, No.7, pp. 123-130, 2020.

[38]. Kilichev, D., Turimov, D., & Kim, W. (2024). Next–Generation Intrusion Detection for IoT EVCS: Integrating CNN, LSTM, and GRU Models. Mathematics, 12(4), 571. https://doi.org/10.3390/math12040571

[39]. Sahu, V. K., Pandey, D., Singh, P., Ansari, H., Shamsul, M., Khan, A., Khan, V.N., Khan, M. W., An empirical analysis of evolutionary computing approaches for IoT security assessment, Journal of Intelligent & Fuzzy Systems, vol. Pre-press, no. Pre-press, pp. 1-13, 2024. DOI: 10.3233/JIFS-233759

[40]. Hochreiter, S. and Schmidhuber, J. (1997). "Long Short-Term Memory," in Neural Computation, vol. 9, no. 8, pp. 1735-1780, 15 Nov., doi: 10.1162/neco.1997.9.8.1735.

[41]. Carvalho, E. C., Ferreira, B. V., Geraldo Filho, P. R., Gomes, P. H., Freitas, G. M., Vargas, P. A., Pessin, G. (2019). Towards a smart fault tolerant indoor localization system through recurrent neural networks. In 2019 International Joint Conference on Neural Networks (IJCNN) (pp. 1-7). IEEE. https://doi.org/10.1109/IJCNN.2019.8852007

[42]. Attaallah, A., Sulbi, K., Alasiry, A., Marzougui, M., Khan, M. W., Mohd Faizan, M., Agrawal, A., Pandey,D., Security Test Case Prioritization through Ant Colony Optimization Algorithm Computer Systems Science and Engineering (CSSE), vol.47, no.3, pp. 3165-3195, 2023.

[43]. Novaes, M. P., Carvalho, L. F., Lloret, J., & Proença, M. L. (2020). Long short-term memory and fuzzy logic for anomaly detection and mitigation in software-defined network environment. IEEE Access, 8, 83765-83781. https://doi.org/10.1109/ACCESS.2020.299204423

[44]. Malhotra, P., Vig, L., Shroff, G., & Agarwal, P. (2015). Long short term memory networks for anomaly detection in time series. In 23rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (Vol. 2015, p. 89).

[45]. Khan, S. A., Khan, M.W., Pandey, D., A Fuzzy Multi-Criteria Decision-Making for Managing Network Security Risk Perspective, Cloud-Based Data Analytics in Vehicular AdHoc Networks, IGI Global, pp.115-140, 2020.

[46]. Zarzycki, K., & Ławryńczuk, M. (2022). Advanced predictive control for GRU and LSTM networks. Information Sciences, 616, 229-254. https://doi.org/10.1016/j.ins.2022.10.078

[47]. Yadav, N. S. S., Yadav, P. S., & Goar, V. (2024). Deep Learning, Neural Networks, and Their Applications in Business Analytics. In Intelligent Optimization Techniques for Business Analytics (pp. 288-313). IGI Global. https://doi.org/10.4018/979-8-3693-1598-9.ch013

[48]. Rihi, A., Baïna, S., Mhada, F. Z., El Bachari, E., Tagemouati, H., Guerboub, M., Abdelwahed, E. H. (2024). Innovative predictive maintenance for mining grinding mills: from LSTM-based vibration forecasting to pixel-based MFCC image and CNN. The International Journal of Advanced Manufacturing Technology, 1-19. https://doi.org/10.1007/s00170-024-14588-3

[49]. Vatanchi, S. M., Etemadfard, H., Maghrebi, M. F., & Shad, R. (2023). A comparative study on forecasting of long-term daily streamflow using ANN, ANFIS, BiLSTM and CNN-GRU-LSTM. Water Resources Management, 37(12), 4769-4785. https://doi.org/10.1007/s11269-023-03579-w

[50]. Airlangga, G. (2024). A Comparative Analysis of Deep Learning Models for SMS Spam Detection: CNN-LSTM, CNN-GRU, and ResNet Approaches. Journal of Computer Networks, Architecture and High Performance Computing, 6(4), 1952-1960. DOI: 10.47709/cnahpc.v6i4.4827

[51]. Ma, Q., Zhang, Q., Liu, M., Zhang, J., Zhu, Y., Liang, Z., ... & Dai, J. (2024). Research on the Interpretability Analysis Method of Transient Stability Assessment in Power Systems Based on Deep Learning. In Proceedings of the 2024 3rd International Conference on Artificial Intelligence, Internet and Digital Economy (ICAID 2024) (Vol. 11, p. 398). Springer Nature. ISBN 978-94-6463-490-7

[52]. Fan, W., Yao, J., Cui, S., Wang, Y., Xu, S., Tan, Y., ... & Wu, W. (2024). Bi-LSTM/GRU-based anomaly diagnosis for virtual network function instance. Computer Networks, 249, 110515. https://doi.org/10.1016/j.comnet.2024.110515

[53]. Li, X., Ma, X., Xiao, F., Wang, F., & Zhang, S. (2020). Application of gated recurrent unit (GRU) neural network for smart batch production prediction. Energies, 13(22), 6121. https://doi.org/10.3390/en13226121

[54]. Sridevi, S., & Karpagam, G. R. (2022). Genetic algorithm-optimized gated recurrent unit (GRU) network for semantic web services classification. Malaysian Journal of Computer Science, 35(1), 70-88. https://doi.org/10.22452/mjcs.vol35no1.5