FAULT TOLERANCE PREDICTION IN DISTRIBUTED SYSTEMS USING GRU, TCN AND LSTM

Faizan Ahmad[1] , Mohd Haroon[2], Zeeshan Ali Siddiqui[3*], and Mohammad Husain[4]

[1,2]Department of Computer Science and Engineering,
Integral University, Lucknow, Uttar Pradesh, India

[3]Department of Computer Science and Engineering,
University of Lucknow, Lucknow, Uttar Pradesh, India

[4]Faculty of Computer and Information System,
Islamic University of Madinah, Madinah, Saudi Arabia

Emails: fahmad@iul.ac.in[1], mharoon@iul.ac.in[2], zeealis@gmail.com[3*] (Corresponding author),
dr.husain@iu.edu.sa[4]

ABSTRACT

Disruptions in distributed systems can cause widespread failures and downtime which costs the company and lowers productivity. A recovery and fault tolerance system is essential because distributed system's complexity, unpredictability, and inner workings exacerbate failures. Predictive analytics has emerged in fault management, helping firms to move from reactive to proactive fault management. In this research, we examine the achievements of intelligent fault prediction approach that employs Gated Recurrent Units (GRU), Temporal Convolutional Networks (TCN), and Long Short-Term Memory (LSTM) networks to improve fault tolerance in distributed systems. The GRU and LSTM models can describe temporal, sequential data for a distributed system and identify and comprehend data changes. By employing these designs and TCNs, companies may better recognize fault patterns that emerge over time, forecast future failures, and increase fault management efficiency. TCNs identify long-range time dependencies and allow parallel processing to swiftly discover and respond to defects in large-scale settings using causal and dilated convolutions. This framework uses deep learning models to examine system log and resource consumption data to identify probable failure symptoms and accurately predict future problems. The experimental results show that combining GRU, LSTM, and TCN models improves fault prediction accuracy and reduces unexpected downtime by identifying faults quickly and taking preventative action. Additionally, since the framework continuously collects data from the distributed system and monitors the logged information in real time, users can make better decisions and implement proactive responses to failures, proving that predictive analytics driven by deep learning technology increases intelligent fault tolerance in distributed systems. With 95% accuracy, 93% precision, 95% recall, 94% F1-score, and 0.97 ROC-AUC, the GRU + LSTM + TCN model outperforms all single and dual-model configurations. With a +7% accuracy boost over GRU, +5% over LSTM, and +4% over TCN, multi-model temporal feature fusion is beneficial for fault prediction.

Keywords: LSTM; Fault tolerance; TCN; Distributed system; GRU; Deep learning technques

1.0 INTRODUCTION

Modern computing infrastructures rely heavily on distributed systems, which support many different types of applications like large-scale data processing and cloud computing, to name a few. Due to their continuous growth in scale, distributed systems are getting more complicated, variety (i.e., heterogeneous), and their operations (i.e., operational complexity). As such, the requirement for creating robust fault-tolerant designs to ensure their operational reliability has become gradually essential.

As the number of distributed systems increases, so too does the potential for those systems to be at risk of failure, as there are now several components involved in a distributed system that could fail or experience an unexpected error/catastrophic event. Even a minor failure, such as a failed piece of hardware, software defect, network failure, or sudden increase in workload, could have a domino effect on the other components of the distributed system, resulting in service outages, data inconsistency, or extensive financial losses [1]. Therefore, fault-tolerant mechanisms need to be developed to maintain reliability, performance, and continuity of operations [2].

Fault-tolerant methods including checkpointing, redundancy, and failover have been used to reduce user impact from system failures. These methods perform well in controlled or static contexts but typically fail in large-scale dynamic distributed systems with shifting workloads and system states. Redundancy-based fault-tolerance

techniques require maintaining additional system components, thus incurring additional operational and financial overheads linked with maintaining those resources, while checkpoint-style fault-tolerance techniques impose additional processing and storage overheads, and may not respond effectively to irrational, dynamically evolving failure conditions. As distributed systems continue to grow in complexity, many innovative fault-tolerant techniques have been developed to accommodate the ongoing changes.

By utilizing predictive analytics, we can create a model for fault tolerance that not only allows us to predict when a failure will occur but it also allows us to take proactive steps to mitigate it before it occurs [3]. Analyzing old logs and telemetry data can reveal the historical context of each event leading up to the approaching breakdown and the performance measure connected with them. This means that we can reduce our reliance on reactive recovery approaches and provide the best opportunity of minimizing system downtime by acting to prevent the impending failure as soon as we identify a pattern of events leading up to its occurrence. The predictive analytics that provide us with these insights can be developed using numerous methods and can utilize the advantages of deep learning (DL) techniques over others that would model complex, multidimensional time-varying behaviors of distributed systems. Amongst the many other predictive approaches developed, Gated Recurrent Units (GRU) and Long Short-Term Memory (LSTM) neural networks have tempted interest for use in predictive modeling techniques for distributed systems due to their capability to work with time-ordered input data (i.e., resource utilization traces and system log entries) [4].

GRU and LSTM neural network designs may find and train to understand temporal relationships needed to detect slow performance decline and delayed failure symptoms in distributed systems [5]. GRUs are deliberate to selectively store relevant historical information, which permits for proficient training while being less resource-intensive than the respective architectures of GRUs [6]. This makes them suitable for use in training predictive analytics in those cases wherein the fault patterns evolve over a longer period of time under a wider range of system conditions. LSTM networks utilize LSTM cells and 4' for a gated approach to correctly learning from sequential input data in order to properly overcome the challenges associated with vanishing gradients [7]. Continuous monitoring of system characteristics, including CPU use, Memory, Disk I/O, and Network Latency, can assist deep-learning models spot abnormalities that may indicate approaching failure. Early detection of these abnormalities enables for the deployment of proactive measures to avoid problems from growing into greater problems that result in prolonged service interruptions [8]. DL Models are able to efficiently analyze very large and rapidly changing data streams, which allows the identification of less apparent or hidden patterns.

Although predictive analytics has numerous advantages, there are many roadblocks to using it effectively within distributed systems [9]. One major challenge of creating predictive models is acquiring high-quality labeled training data. Data Pre-Processing can also be a complex and time-consuming process. Lastly, integrating DL models into current fault management processes requires scalable architectures for real-time data acquisition and analysis, as well as real-time inference and automatic response capabilities [10]. DL-based predictive analytics increase distributed system fault tolerance by recognizing faults early and lowering downtime, according to various studies [11].

1.1 Problem Statement

Traditionally, organisations have relied on a variety of fault tolerance techniques, such as redundancy (having multiple copies of the same item), checkpointing (storing data periodically), reactively recovering from failures (the system reacts to an event after it has already happened) [12]. The issue with these types of methods is that they are only reactive, meaning that if something breaks, the system responds after the event. As we move forward into more dynamic and larger-scale environments, the reactive methods will be insufficient to prevent the occurrence of cascading failures, prolonging the amount of time a system is down, and to predict when failures will occur [13]. For example, current prediction methodologies, using traditional machine-learned methods, fail to identify complex temporal dependencies, to work on scale, in real-time, and to deal appropriately with noisy and imbalanced monitoring data [14]. Therefore, we still need an intelligent, scalable, and proactive fault prediction system that can forecast problems and give an organization early mitigation alternatives.

1.2 Motivation

With the increase in access to high-frequency system logs and performance telemetry data, there is a prospect to change the method we use for managing faults from one that is reactive to one that is proactive (predictive). Several recent advances in DL, and specifically, sequence modeling techniques, show a lot of promise in learning temporal trends from time series data and detecting subtle degradation trends that can lead to system failure. Currently, we see the most success using GRUs and LSTMs for modeling sequential system behavior. Due to their sequential processing, GRUs and LSTMs are not scalable for large-scale distributed systems [15].

Temporal Convolutional Networks (TCNs) present a new prospect for tackling with long range, temporal dependencies using a novel mechanism - causal and dilated convolutions allow processing of data in parallel while still learning long-term temporal relationships throughout the entire sequence of data [16]. While using TCNs for predicting system faults is promising, their practice has been restricted in research and has not undergone extensive

comparative evaluation against the more established recurrent architectures. For this reason, we are motivated to create a unified predictive fault tolerance (PFT) methodology to integrate the advantages of GRU, LSTM and TCN into a single predictive fault tolerance framework.

1.3 Novelty of the Work

The innovative aspect of this research is the opportunity to compare and integrate the Benefits of GRU, LSTM and TCN Models as Predictive Fault Tolerant Techniques of Distributed Systems. In contrast to previous research based on either a single predictive model or one that exclusively examines the accuracy of each individual predictive model, this study provides an overall structured experiment to evaluate and compare Recurrent Neural Networks (RNNs) and Convolutional Networks of Time Series Data. A TCN is more efficient and scalable for modeling long-term temporal dependencies, but this research also used GRU and LSTM networks, which can capture short- and long-term temporal correlations in sequential data. Using real-time telemetry data to identify defects will also challenge reactive fault management.

1.4 Main Contributions:

Key contributions of this study are as follows:

1. A fault tolerance framework employs predictive algorithms to split continuous monitoring data into predictive fault data for proactive mitigation.
2. Evaluating the different types of models (GRU, LSTM) and TCNs for generating fault information and comparing them with regard to their strengths and weaknesses to describe temporal system behavior.
3. Demonstrating the capability of TCNs as a scalable and efficient way of generating fault information in large distributed computing environments when compared with the use of recurrent network architectures.
4. Quantitative investigations using system telemetry data to validate the predictive fault tolerance framework's dependability and resilience. Confusion matrices, ROC curves, ablation studies, and statistical significance were tested.
5. Quantifying the improvement in reliability of fault prediction and decreasing the length of downtime associated with fault occurrence confirms that analytic methods developed using DL techniques may be used successfully to increase resilience and improve operational reliability in large distributed computing environments.

2.0 RELATED WORK

Recently, fault tolerance in distributed systems is changing from reactive solutions (e.g., recovery methods) to predictive ones (e.g. intelligent predictive methods). Historically, redundancy, checkpointing, and failover were the primary focus, and typically lead to significant overhead and lack of proactiveness in a dynamic environment. Researchers have used ML and DL to forecast system failures in recent years. Many researchers have researched the prediction of failures using classical ML algorithms (e.g., Support Vector Machines (SVMs), Random Forests (RFs), Decision Trees) by utilizing logs from the system and metrics related to resource utilization.

While the performance of models built using classical ML algorithms is generally good in many cases, they do not effectively capture complex temporal dependencies, and they tend to perform poorly when adaptive in an evolving, distributed environment. Therefore, researchers are increasingly looking towards DL-based sequence models. LSTM and GRU architectures, both types of RNNs, are possible because they have the ability to build very complex sequential models [17 and 18]. [19] found that RNN models based on LSTMs have a significant positive effect on speed and accuracy in detecting early faults in HPC (High-Performance Computing) applications through building long-term temporal learning models. Similarly, [20] used LSTM and GRUs for predictive fault tolerance in distributed systems, which resulted in a decreased amount of unexpected downtime and false alarms.

Recently, researchers have begun to explore TCNs as an alternative to RNN architectures [21]. Unlike RNN approaches, TCNs utilize causal and dilated convolution methods to analyze and model temporal dependencies without the need for sequential processing like RNNs do. [22] evaluated TCNs against RNN architectures and showed that they are more efficient than RNNs when modeling sequences, and there have been subsequent applications of TCN methodologies for detecting anomalies and predicting faults within distributed and industrial systems. Both LSTM networks and TCN networks have demonstrated improvements in regards to expanding scalability and reduced latency relative to their respective use cases.

There have been multiple attempts to tackle the Privacy, Scalability, and Structural Dependence issues associated with fdedrated learning through the application of federated learning or graph-based DL. For instance, Federated Fault Prediction frameworks facilitate collaborative learning among numerous nodes without requiring a centralized repository for data sharing [23], while Graph Neural Networks identify inter-component relationships by simulating the methodical approach by which failures propagate between components [24]. Though there have

been many advances in this area, there continue to exist several challenges, including handling imbalanced data to facilitate real-time deployment, and achieving interpretation of complex model behaviour.

Currently, the available research supports that no single model is capable of solving every problem associated with Fault Prediction in Distributed Systems. To accomplish this, the researcher recommends implementing an innovative/creative hybrid-comparative framework that incorporates at least three of the predictive DL techniques listed here: GRU, LSTM, and TCN. With these hybrid-comparative approaches, it will be possible to maximize the predictive accuracy, scale, and these multi-component systems will become even more robust than they were before implementation. Table 1 compares existing fault prediction approaches across techniques, datasets, prediction types, strengths, and limitations. It highlights trade-offs between accuracy, scalability, interpretability, and complexity, while showing that the proposed work achieves improved proactive fault prediction with reduced downtime compared to prior methods.

Table 1: Comprehensive Comparison of Existing Studies

Author / Year	Model / Technique	Dataset / Environment	Prediction Type	Strengths	Limitations
Zhang et al. (2020) [25]	SVM, RF	Cloud metrics	Binary fault classification	Simple, interpretable	Poor temporal modeling
Das et al. (2021) [26]	LSTM	HPC system logs	Time-series fault prediction	Captures long-term dependencies	High training cost
Munir et al. (2021) [9]	GRU–LSTM + Attention	Software defect data	Defect prediction	Improved recall	Complex architecture
Hao et al. (2021) [27]	Statistical + ML	Industrial systems	Fault diagnosis	Effective in stable systems	Poor generalization
Hao et al. (2020) [28]	TCN	Benchmark time-series	Sequence modeling	Parallel processing, stable gradients	Limited domain validation
Deng and Hooi (2021) [29]	GNN	Distributed service graphs	Anomaly detection	Models fault propagation	Graph construction overhead
Alzamil (2025) [30]	Federated DL	Edge & distributed nodes	Privacy-aware prediction	Preserves privacy	Communication cost
Seba et al. (2024) [8]	Hybrid DL	IoT sensor data	Fault classification	Scalable, robust	Limited distributed focus

2.1 Research Gap

The field of distributed system fault prediction has made significant strides. Nevertheless, there remain many distinct deficiencies in existing work on this topic. The majority of current research is based on individual models of ML or DL (e.g., LSTM, SVM or GRU) and lacks a complete comparative assessment of several advanced models/archaeological resources. In addition, research into the use of TCN in the context of fault prediction in distributed environments has not yet been significantly established as opposed to traditional recurrent models. Furthermore, the primary focus of many past studies was to investigate the accuracy of predictions whereas little attention has been paid to the objectives associated with system performance such as reducing downtime, detecting faults at an early stage, achieving scalability in real-time, and avoiding uncertainties. Important localised issues such as responding to changing load conditions, producing fault data of different types, creating a variation of fault data based on prior experience, etc., have not been sufficiently developed or investigated and therefore there are no presently accepted comprehensive, sustainable and proactive frameworks for developing and implementing effective fault tolerance in distributed systems that have been well integrated and tested using reliable temporal

models. This paper introduces and evaluates a new approach that compares DL predictive models using GRU, LSTM, and TCN models to solve proactive fault detection and downtime problems.

2.2 Fault Tolerance in Distributed Systems Using GRU, LSTM, and TCN Models

To maintain steady and reliable operation in a distributed system in spite of component failure, asynchronous degradation in performance, or unexpected environmental factors, effective fault tolerance mechanisms are essential. The issues arising from increased scale, heterogeneity, and dynamic behavior in today's distributed infrastructure have rendered reactive fault tolerance methods inadequate. Predictive fault tolerance, utilizing the capabilities of intelligent learning models, now offers a means to anticipate future failures and proactively mitigate them. A consequence of the strong ability of DL models to develop fault predictions by learning complex temporal patterns from monitoring time series has been the use of these models, particularly GRUs, LSTMs, and TCNs, to improve the fault tolerance of a distributed system through the early detection and prediction of faults [28].

2.2.1 GRU-Based Fault Tolerance

GRUs are a subtype of RNNs designed to work with sequential data; they do this through use of a unique architectural approach that employs two types of gates (update and reset gates), allowing only a limited amount of historical information to be retained, thus minimizing redundant information. Finally, because of this unique design, the complexity of the GRU architecture allows them to have lower computational needs than LSTM architectures which make GRU architectures best suited for high-volume, highly variable, global and redundant fault prediction applications where real-time predictions are required [26].

GRU may develop long-term connections in a controlled manner, unlike typical RNNs that frequently suffer from the vanishing gradient problem. The GRU uses two special types of gates (reset and update) that are capable of facilitating the training process of a network by determining when a specific memory cell should remember a piece of information [9]. Learning how these gates operate from the mathematical equations that represent both gates can provide insights into how GRUs operate and their high degree of performance when processing a sequence of input.

A. Input

1. A sequence of input vectors $x_1, x_2 \ldots\ldots, x_t$, where T is the number of time steps in the sequence.
2. h_0: Initial hidden state (usually initialized to zero).
3. 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. Computations

The GRU cell does the following calculations at each time step t [20]:

Update Gate (z_t)

With the update gate, it shows what portion of the last hidden state (h_{t-1}) will be carried over to the next hidden state. Therefore, it is responsible for deciding whether to keep the previous hidden state (h_{t-1}) memory or to replace it with new data in the current state, ht.

Formula:

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

Where: The vector of inputs at time t is represented by x_t. The hidden state is denoted as h_{t-1}, W_z represents the input weight matrix and U_z represents the previous hidden state weight matrix in the update gate. The update gate also has an associated bias term b_z, and the sigmoid activation function (σ) will result in values that should fall within the range of 0 to 1 when applied to the output.

Reset Gate (rt)

The reset gate determines how much of the concealed state to ignore. This gate allows the GRU to ignore irrelevant former state when processing current input.

$$r_t = \sigma(W_r \cdot x_t + U_r \cdot h_{t-1} + b_r) \quad (2)$$

Where: Weight matrix Wr for input xt is in the reset gate, Weight matrix U_r for prior hidden state h_{t-1} is in the update gate, and bias term br is used by the update gate.

Candidate Hidden State ($\tilde{h}_t$)
The candidate hidden state is computed using the reset gate r_t, input x_t, and prior hidden state h_{t-1}. It elects fresh material for the hidden state [20].

Final Hidden State (h_t)
The current time step GRU cell output is the final hidden state h_t. Combining the prior hidden state h_{t-1} with the candidate state $\tilde{h}_t$ controlled by the update gate z_t [20].

C. Classification Layer
Final hidden state h_t from last time step represents complete sequence. h_t is sent via a dense, fully connected layer with a softmax (for multiclass classification) or sigmoid (for binary classification) activation function to classify [20].

D. Loss Function
We train the GRU for classification using a task-specific loss function [20]:

- **Binary Cross-Entropy Loss**:

$$Loss = -\frac{1}{M}\sum_{i=1}^{M}(y^{(i)}\log(\hat{y}^{(i)}) + \left(1 - y^{(i)}\right)log(1 - \hat{y}^{(i)})) \quad (3)$$

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

$$Loss = -\frac{1}{M}\sum_{1=1}^{M}\sum_{k=1}^{M} y_k^{(i)}\, log\left(\hat{y}_k^{(i)}\right) \quad (4)$$

With M training examples, $\boldsymbol{y^{(i)}}$ represents the real label of the i-th sample, and $\hat{y}^{(i)}$ or $\hat{y}_k^{(i)}$ represents the projected probability.

2.2.2 LSTM-Based Fault Tolerance

Memory cells and gate control techniques for the vanishing gradient problem add complexity to RNN architectures in LSTM networks [20]. These networks may capture longer temporal data dependencies than typical RNN structures. Specifically, LSTMs are particularly capable of capturing gradual fault development, or failure isolation, over an extended period, which is characteristic of distributed systems [31]. Furthermore, LSTM-based models for fault prediction enable learning of delayed symptoms in system behaviour or subtle changes in trends that might not be immediately visible. Therefore, LSTMs facilitate early fault detection and enable more timely provisions for recovery, thus improving trust in the reliability and reducing downtime of distributed systems.

A. Input Sequence and Initial Hidden States
The LSTM will take a series of input vectors (x_1-x_T) as input one time-step at a time. In a typical LSTM implementation, the initial hidden state (h0) is usually set to be 0, and the initial cell state C_0 is also usually set to 0. For each time step t, a new input x_t is processed resulting in the calculation of an updated hidden state h_t and updated cell state C_t. LSTM processes input x_t and updates hidden state h_t and cell state C_t at time step t.

B. LSTM Cell Computations
LSTM cells handle sequential input by retaining and updating two important states at each time step t [20]: the cell state C_t, which is long-term memory, and the hidden state ht, which is output and short-term memory. Gates govern information flow in these states. How much of C_{t-1} should be kept or destroyed is determined by the forget gate f_t. A sigmoid activation function creates values between 0 and 1, with values near 1 preserving information and values near 0 removing it. LSTMs may selectively forget irrelevant prior knowledge [31]. The input gate regulates how much current input xt updates the cell state. Parallelly, the candidate cell state C_t is calculated using the current input and the preceding hidden state h_{t-1}, signifying possible new memory. The input gate and candidate state integrate only relevant new data. Cell state updates combine these. The forget gate scales the previous cell state, then the input gate and candidate cell state element-wise product adds fresh information. This method lets

the LSTM balance memory retention and updating across lengthy sequences. The output gate ot selects concealed cell state portions. The output gate filters the cell state to create the hidden state ht after a tanh activation function bounds its values. LSTM iterates these procedures at each time step to capture long-term relationships in sequential data, solving the vanishing gradient problem and enabling robust sequence learning [20].

C. Classification Layer

In an LSTM-based classification model, the final hidden state ht is commonly used as a compact representation of the entire input sequence [20]. This hidden state captures the relevant temporal patterns and contextual information learned across all time steps. For classification, ht is passed to a dense (fully connected) output layer, where the choice of activation function depends on the type of classification task.

For multiclass classification, where the number of classes is K, the output layer consists of a weight matrix Wout and a bias vector bout. The linear transformation of the final hidden state produces a set of logits:

$$z = W_{out} \cdot h_t + b_{out} \quad (5)$$

These logits are converted into class probabilities using the softmax activation function, which normalizes the outputs so that the probabilities of all classes sum to one. Each probability reflects the model's confidence that the input sequence belongs to a particular class. For binary classification, the output layer produces a single scalar value, which is passed through a sigmoid activation function. The sigmoid function maps the output to a value between 0 and 1, representing the probability of the positive class.

To train the LSTM model effectively, an appropriate loss function is used to measure the difference between predicted outputs and true labels [31]. For binary classification, binary cross-entropy loss is applied, penalizing incorrect probability estimates for both positive and negative classes. For multiclass classification, categorical cross-entropy loss is used, which evaluates how well the predicted probability distribution matches the true class distribution. During training, these loss functions guide the optimization process, enabling the LSTM to learn meaningful sequence representations and improve classification accuracy.

2.2.3 TCN-Based Fault Tolerance

TCNs are a new kind of time-series fault prediction model. In contrast to recurrent architectures, TCNs are built using causal and dilated convolutions, allowing for parallel computation and more flexible time-series modelling than traditional methods [28]. Therefore, training TCNs is significantly more efficient and scaled than that of RNNs and thus TCNs are better suited for use in high-throughput distributed systems. Thus, TCN-based fault prediction algorithms can evaluate vast amounts of telemetry data in real time and detect potential faults quickly. The stable gradients of TCNs combined with their ability to model a longer temporal context make TCNs a significant complement to GRU and LSTM networks in predictive fault tolerance.

1. Input Representation

We can represent the multivariate time-series data from the distributed system for training the model by defining a variable called "X". The letter "X" represents all time-point values in the system: {x1, x2, … xT(e.g. T = current time point)}, with x_T corresponding to the state of the system (S) represented by a vector of d system metrics (CPU usage, disk input/ooutput, memory consumption, and network latency, etc.).

2. Causal Convolution

Causal convolution ensures that predictions made at time t only rely on the current time point and all previous time points. Causal Convolution creates a series of prediction outputs at each time-step that includes all prior inputs and outcomes.

3. Dilated Convolution

Dilated Convolution allows us to efficiently model long-range temporal dependencies in the inputs and create predictions based on more than just the immediately preceding time point. The time-step "dilation" factor for the current layer is represented as "dl" in the above formula. The dilated convolution allows us to make predictions by leveraging all previous time steps, which extends the depth of our predictions.

4. Residual Learning

Residual connections improve training stability and gradient flow:

$$yt(l) = f(ht(l)) + h(t)(l-1) \quad (6)$$

where f(.) represents a nonlinear activation function such as ReLU.

5. Fault Probability Estimation

A sigmoid function maps the final TCN layer output to fault probability:

$$\hat{y}(t+1)^{TCN} = \sigma(W_{out} \cdot h(L)t + b_{out}) \quad (7)$$

$where\ \hat{y}(t+1)^{TCN} \in [0,1]$ represents the predicted fault probability.

6. Fault Tolerance Decision Rule

A fault is predicted if the estimated probability exceeds a predefined threshold θ:

$$Fault(t+1) = 1, \text{ if } \hat{y}(t+1)^{TCN} \geq \theta; \text{ otherwise } Fault(t+1) = 0. \quad (8)$$

Upon fault prediction, proactive mitigation actions such as workload redistribution, resource scaling, or failover are initiated.

7. Loss Function

Training the TCN model using binary cross-entropy loss:

$$L = -(1/N) \Sigma [\ y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)\ ] \quad (9)$$

2.3 Integrated Fault Tolerance Framework

By combining the capabilities of three cutting-edge architectures, GRU, LSTM, and TCN, when creating a predictive analytics platform for distributed systems, we can utilize the advantages each of them brings [31]. In particular, GRU provides superior computation efficiency; LSTM is appropriate for marking and modeling long-term dependencies; and TCN provides a highly efficient way to simultaneously capture and model temporal trends across multiple nodes in a distributed system. By leveraging the capabilities of these models, we can create a highly accurate method of predicting faults and detecting abnormalities, enabling users to quickly intervene before a catastrophic failure event occurs. The integration of GRU, LSTM, and TCN into our Fault Tolerance Framework allows us to not only react to faults after they happen (through recovery mechanisms), but also allows us to anticipate faults before they occur by using intelligent predictions. Therefore, by making the most of these advances in predictive analytics, we will both increase the overall availability of distributed systems, and will reduce the overall downtime of these systems, while enhancing their overall resilience.

3.0 METHODOLOGY

This research project establishes a comprehensive predictive fault tolerance model for distributed systems that allow early detection of faults through the use of DL based temporal modal in addition to implementing proactive measures. The new approach will systematically convert the monitoring data produced in the distributed environment to provide usable fault predictions through a complete workflow consisting of collecting monitoring data from the distributed system, preparing and labeling the monitoring data for use as a training set, selecting and reducing the number of data features for model training, training fault prediction models using both GRU, LSTM, and TCN based neural networks, predicting faults based on the use of the trained models, and finally evaluating the fault predictions made by the models and completing a comparative analysis of the models.

The suggested approach has several steps:

1. Stage Reference Number 1: Gathering the Monitoring Data from the Distributed Infrastructure
2. Stage Reference Number 2: Preparing and Labeling(making labeled) the Data
3. Stage Reference Number 3: Selecting and Reducing the Features of the Data
4. Stage Reference Number 4: Training Models using GRU, TCN, and LSTM neural networks
5. Stage Reference Number 5: Predicting Faults and Making Decisions
6. Stage Reference Number 6: Evaluating the Performance of Fault Predictions

Utilizing a structured methodology is important in allowing the framework to transition from the traditional reactive fault handling model to a proactive fault tolerance model that reduces downtime and increases the reliability of distributed systems.

Telemetry data from components of a distributed system will be collected continuously; from compute nodes, to storage devices, to networking equipment. Both "normal' operational states and "fault' states will be documented in the telemetry data to support supervised learning. The principal sources of telemetry data are: Event Logs - these will record all event occurrences (including warnings, errors, exceptions, and transactions). Performance measures, such as CPU Utilization (%), Memory (RAM) Used (%), Network Latency (ms), Packet Loss (%), Request Rate (requests/second), Error Count, and Network Throughput (MB/s), quantify a system's status and are attainable. Environmental and contextual parameters, such as workload intensity and network congestion, can affect system operation but not performance. Data is collected every fixed period; hence the data generated from the data acquisition phase will be a multivariate time-series data set, which can subsequently be modeled over time using a DL framework.

The framework includes four key phases for detecting defects in distributed systems: collecting information, selecting features, preprocessing, and training a model. The data collecting phase uses technology to gather data from dispersed systems to discover issues. System logs, which describe specific events and transactions with a system; performance measurements, which measure CPU loads, memory utilization, I/O rates, and other conditions indicating the "health" of a system; and environmental metrics, such as network latency and power status, are the most tedious sources of data.

Once the data is collected, it is then cleaned up and organized for machine-learning model training through preprocessing steps. During preprocessing, several steps are taken, including, labeling of the observations with a binary status (fault vs non-fault), treatment of missing data, normalization of the data to the same scale, and encoding of categorical predictors into numeric form. After the data is preprocessed, the feature selection step determines the best feature predictors of system failure by eliminating irrelevant features. Recursive feature elimination (RFE), correlation analysis, and expert knowledge of the domain are employed to select and identify the valuable features. Finally, datasets are distributed into training sets and testing sets. Usually, 70% of the data will be used to train structured, comprehensive, and strategically designed approach to identifying and predicting failures in distributed systems.

Fig. 1 illustrates the proposed comparative fault prediction framework using GRU, LSTM, and TCN models. A common dataset is fed into all three models, where each independently performs fault calculation and generates predicted faults. The calculated faults are compared with predicted faults, followed by system evaluation to assess performance.

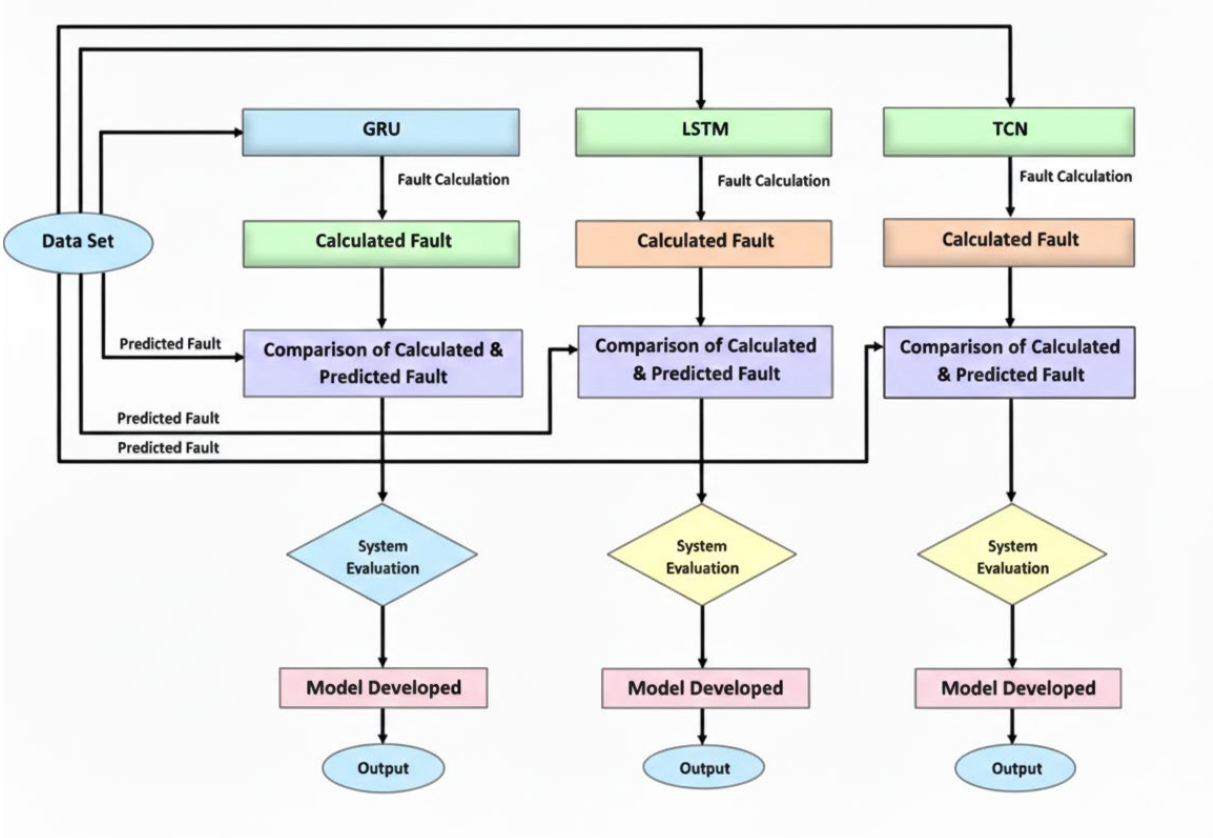

Fig. 1: Flowchart of proposed framework

3.1 Data Collection

System logs are one of the most significant sources, as they provide an accurate and complete record of all activity and events within the system; performance metrics help quantify the general health of a distributed system by monitoring such things as disk I/O rate, CPU utilization, and memory usage. Network conditions (for example, network latency) and power supply status (for example, UPS power losses) are also significant aspects impacting the performance of distributed computing systems, and they should considered when evaluating how well a distributed computing system is performing. Collectively, all these various sources of information form the foundation for investigating and resolving potential problems in the operation of a distributed computing system.

3.2 Data Pre-processing

The preparation of data ensures the input into a model is accurate and arranged correctly to support training effectively. Among other significant tasks in data preparation is addressing missing data by locating and filling them in order to maintain the integrity of the dataset being used to train the model. Data normalization provides a uniform scale for variable values, which helps to increase performance levels in the models created by this data. Encoding Categorical Variables: ML Algorithms rely upon Categorical Variables that don't have associated functional relationships. For this reason, encoding converts these Categorical Values into Numeric values. Data Labeling: Adding Target Labels to Observations, also commonly referred to as Supervised Learning, is a way to accurately predict faults, errors, or problems through the use of labeled observations to train the model.

3.3 Feature Selection

Feature selection seeks system failure causes. Reducing dimensions improves model performance. We can determine which variables are most relevant in forecasting System Failure by studying correlations. Iteratively deleting variables using Recursive Feature Elimination helps us identify the optimal prediction model for our data. Domain specialists can also choose and adjust Variables. These strategies will improve ML Model correctness and performance by include just the most important Factors in the Dataset.

3.4 Model Training

For balanced evaluation, training and test sets are split 70/30. Labelled data then trains GRU and LSTM models to catch patterns and anticipate system breakdowns.

3.5 Model Evaluation

After training, these performance metrics assessed the trained classifier's classification test set performance and addressed the negative effects of imbalanced data on model classification results. These metrics include:

1. Accuracy = % of instances properly classified, providing an overall view of the model performance.
2. Precision = % of fault predictions that were true positive results, measuring the confidence/predictability of the classifier.
3. Recall (or Sensitivity) = number of actual faults as compared to the number of faults predicted, indicating the ability to identify the faults that exist.
4. F1-Score = equal weight given to both uncertainty characteristics (Precision & Recall) as a function of the F1-Score.
5. Receiver Operating Characteristic (ROC) Curve = graphical representation of the TPR vs. FPR.
6. Area under the ROC Curve (AUC) = measures the discriminative capability of the model to distinguish between positive and negative cases in relation to the ROC Curve.

By examining all of the above-listed performance metrics together, we can gain an appreciation of the degree of success achieved by the predictive model in detecting rare and difficult to locate faults within complex or multi-faceted systems. Table 2 shows multiple traits characteristics along with their descriptions.

More efficient and scalable ML models for predictive fault-tolerance applications should be the focus of future research. RNNs and CNNs enable fresh insights from system logs and time-series datasets and improve predictive fault-tolerant techniques [20]. Distribution systems' predicted fault-tolerance will increase using resilient hybrid models that combine classic and new ML approaches. These cutting-edge methods and integrative approaches to fault tolerance mechanisms can improve predictive fault-tolerant approaches for distributed systems, making them more resilient and reliable.

Table 2: Traits characteristics

Characteristics	Definition	Format	Example
CPU Utilization (%)	Percentage of CPU utilization at time.	double	75.3
Memory Utilization (%)	Percentage of memory utilization at time.	double	68.5
Network Delay (ms)	Network delay in milliseconds.	double	15.2
Packet Loss (%)	Percentage of packets lost during network communications.	double	0.5
Request Rate (req/s)	Number of requests received per second.	int	325
Error Count	Number of errors recorded during this time frame.	int	3
Network Throughput (MB/s)	Network data rate in megabytes per second.	double	10.3
Fault	Target label showing fault (1 for defect, 0 for no fault).	binary	1 (Fault) or 0

4.0 EXPERIMENTAL SETUP

This portion illustrates the experimental methodology implemented for assessing how well the suggested forecasted failure tolerance strategy works, employing GRU and LSTM models. The framework presented here provides details on the data used for testing and training, the ways in which instances were organized into test/training sets, the manner in which the system's ML algorithms were established, the assessment metrics employed to gauge performance, and the conditions under which those metrics were established so that others may replicate the same findings.

4.1 Dataset Description

The Distributed System Telemetry Fault Dataset (DST-FD) was created from data collected continuously from a telemetry system used to monitor a distributed system for occurrences of fault conditions. The dataset consists of multivariate, multilevel time-series telemetry data and captures the normal and fault conditions of the distributed system. The DST-FD dataset contains key metrics regarding the operation of the distributed systems' performance, for example; CPU utilization, Memory usage, Network Latency (Round Trip Time), Packet loss, Number of Requests received, Error count, and Throughput of the Network. The labels of each observation indicate if a fault was present; for example, a 1 indicates Fault condition and 0 indicates Normal condition. Each data sample was collected and documented at regular intervals for effective temporal modeling using the recurrent DL architectures GRU and LSTM; meaning; to establish a time-series data model for analysis using the time-series data collected in the DST-FD dataset.

Prior to the creation of the training and testing data subsets, the DST-FD dataset was; normalised, been discarded of entries which had missing value(s), and labelled. In creating the training and testing data subsets, 70% of the data was allocated for training and 30% for testing. The DST-FD dataset is the basis for evaluating the predictive fault tolerance ability of DL based architectures in Distributed Systems.

4.2 Evaluation Metrics

Assessment of the performance of the models was done using several different types of statistics as shown in the following:

1. Overall Correctness of predictions = Accuracy
2. Reliability of predicted Fault Instances = Precision
3. Ability to detect actual faults = Recall (Sensitivity)
4. Harmonic mean of precision and recall = F1-Score
5. Discriminative ability based on threshold level = ROC Curve and AUC
6. Using confusion matrices allows two models to be compared visually and gives insights into the behaviour and misclassification patterns of each model.

4.2.1 Performance metrics for GRU:

The dataset includes performance measurements, logs, and a target label indicating whether a failure occurred (failure = 1) or not. Data is gathered every minute or 10 seconds over a set time to capture normal operations and failure situations. Table 3 exhibits the multiple feature dataset Training Samples from DST-FD. The prediction performance of each model is assessed using these tables. The dataset has four characteristics: performance measurements, log files, and target labels indicating faults (Fault = 1 or 0). Data was taken every minute or ten seconds. The collection covers failure and regular operation statistics. These tables help assess each model's fault prediction accuracy.

Table 3: Multiple feature dataset Training Samples from DST-FD

Time Step	CPU Usage (%)	Network Latency (ms)	Memory Usage (%)	Request Rate (req/s)	Packet Loss (%)	Network Throughput (MB/s)	Error Count	Fault
t_{81}	58.4	13.9	55.1	260	0.1	9.2	1	0
t_{82}	60.7	14.2	56.8	275	0.1	9.4	1	0
t_{83}	63.9	14.8	58.6	290	0.2	9.7	2	0
t_{84}	66.5	15.3	60.4	305	0.2	10.0	2	0
t_{85}	69.8	15.9	62.7	325	0.3	10.4	3	0
t_{86}	72.6	16.4	64.9	345	0.4	10.8	4	0
t_{87}	75.9	16.9	67.2	365	0.4	11.2	4	0
t_{88}	78.3	17.3	69.5	385	0.5	11.6	5	1
t_{89}	80.7	17.8	71.8	405	0.6	12.0	6	1
t_{90}	83.1	18.2	73.6	420	0.6	12.3	6	1
t_{91}	85.4	18.7	75.4	440	0.7	12.7	7	1
t_{92}	87.6	19.1	77.1	460	0.8	13.0	8	1
t_{93}	89.2	19.5	78.9	480	0.9	13.3	9	1
t_{94}	90.8	19.9	80.5	500	1.0	13.6	10	1
t_{95}	88.5	19.2	79.2	470	0.9	13.2	8	1
t_{96}	85.1	18.5	76.8	440	0.7	12.8	7	1
t_{97}	82.4	17.9	74.3	410	0.6	12.2	6	1
t_{98}	78.6	17.1	71.5	380	0.5	11.7	5	0
t_{99}	72.3	16.2	66.9	340	0.4	11.0	4	0

4.2.2 Confusion Matrix for GRU-Based Fault Prediction

1. Assuming a binary classification setting:
2. Positive class (1): Fault
3. Negative class (0): Normal operation

Fig. 2 illustrates the classification performance of the GRU-based fault prediction model by depicting true positives, true negatives, false positives, and false negatives across fault classes.

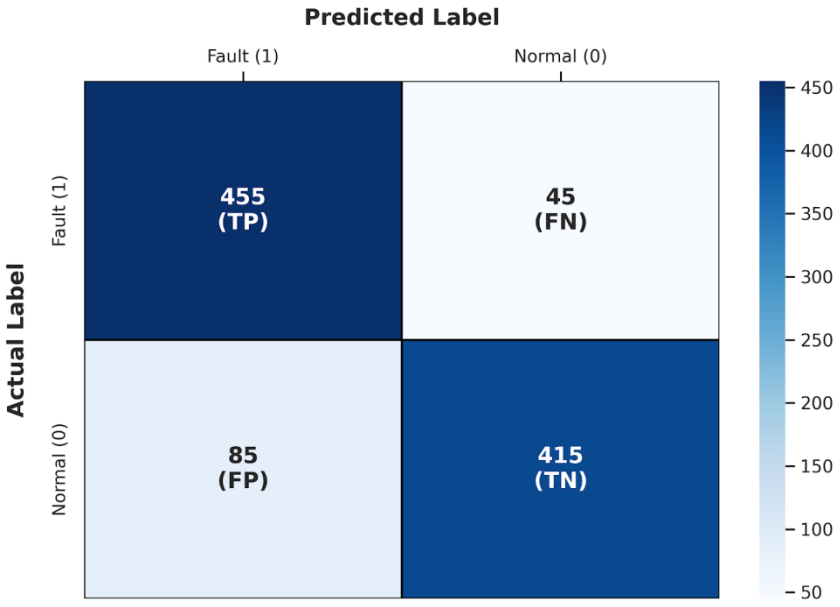

Fig. 2: Confusion Matrix for GRU-Based Fault Prediction

Fig. 3 compares the training and testing performance of the GRU model, highlighting its generalization capability and consistency across datasets.

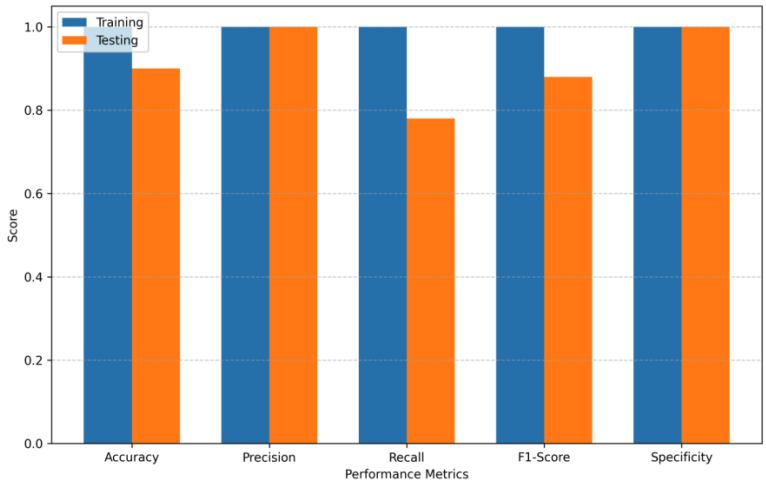

Fig. 3: GRU Training vs Testing Performance Bar Chart

Table 4 presents the evaluation of the GRU model on the DST-FD testing dataset using key performance metrics, demonstrating its effectiveness in fault detection.

Table 4: Performance metrices of GRU from DST-FD testing data

Time Step	CPU Usage (%)	Network Latency (ms)	Memory Usage (%)	Request Rate (req/s)	Packet Loss (%)	Network Throughput (MB/s)	Error Count	Fault
t_{101}	59.6	14.1	56.4	265	0.1	9.3	1	0
t_{102}	61.8	14.5	57.9	280	0.2	9.6	2	0
t_{103}	64.2	15.0	60.1	300	0.2	9.9	2	0
t_{104}	67.9	15.6	62.8	325	0.3	10.3	3	0
t_{105}	71.3	16.2	65.5	350	0.4	10.8	4	0
t_{106}	74.8	16.8	68.1	375	0.4	11.2	4	0
t_{107}	77.6	17.3	70.4	395	0.5	11.6	5	1
t_{108}	80.2	17.9	72.9	415	0.6	12.0	6	1
t_{109}	82.7	18.4	75.3	435	0.7	12.4	7	1
t_{110}	85.1	18.9	77.8	455	0.8	12.8	8	1
t_{111}	87.4	19.3	79.6	475	0.9	13.1	9	1
t_{112}	89.6	19.7	81.2	495	1.0	13.4	10	1
t_{113}	86.9	18.8	78.7	460	0.8	13.0	8	1
t_{114}	83.5	18.1	75.9	430	0.7	12.5	7	1
t_{115}	79.8	17.4	72.4	400	0.6	12.0	6	1
t_{116}	76.1	16.7	69.6	370	0.5	11.5	5	0
t_{117}	71.5	16.0	66.3	335	0.4	10.9	4	0
t_{118}	67.2	15.2	62.5	305	0.3	10.2	3	0
t_{119}	63.4	14.6	59.3	285	0.2	9.7	2	0

Overlay of GRU model training and test performance on DST-FD. The GRU model performs 100% on the training dataset, however its recall and F1-score decline significantly on the testing dataset. The GRU model's training and testing datasets have a large generalization gap, highlighting the necessity for more advanced temporal data models. Gradual patterning of telemetry data for Time-Series Fault Detection is evident through the presence of distinct measurements collected incrementally over time. These data patterns are most efficiently processed by LSTM Neural Networks. The architecture of an LSTM network consists of several layers that provide it unique features that allow it to retain and manipulate information about long-term time points sequentially.

1. The LSTM model learned during training:
2. To identify long-term correlations among computer system metrics (e.g., CPU use, memory usage, latency and number of errors),
3. To observe gradual degradation of system performance and rate change in performance prior to fault events.
4. To discern the difference between transitory anomalies and actual faults based on timing and duration of the event.

4.2.3 LSTM-Based Data Analysis on the Training Dataset (DST-FD)

The training data set in the Distributed System Telemetry Fault Data set (DST-FD) includes multivariate time-series samples that represent normal operation, gradual degradation, and fault statuses. The Long Short Term Memory (LSTM) model has been trained using sequential telemetry features (such as CPU Usage, Network Latency, Memory Utilization, Packet Loss, Error Count, Request Rate, and Network Throughput). With an LSTM's Memory Cells and Gated Structure, it is designed to learn long term dependencies from training data (and

specifically, it helps identify gradual escalation of faults that are typically more challenging to determine with less complex RNNs).

The LSTM model shows the following characteristics during the training process:

1. Convergence is stable with small amounts of oscillation in the loss value.
2. The cell state mechanism allows the LSTM to keep a higher level of historical information about the system than the GRU does.
3. The LSTM is better at detecting slower performance drops leading up to a failure event.

Unlike the GRU, the LSTM uses forget and input gates to store past data across time.

4.2.4 Confusion Matrix for LSTM-Based Fault Prediction

Classification setting:

1. Positive class: Fault
2. Negative class: No Fault

Fig. 4 illustrates the fault classification performance of the LSTM-based model by visualizing true and false prediction outcomes across fault classes.

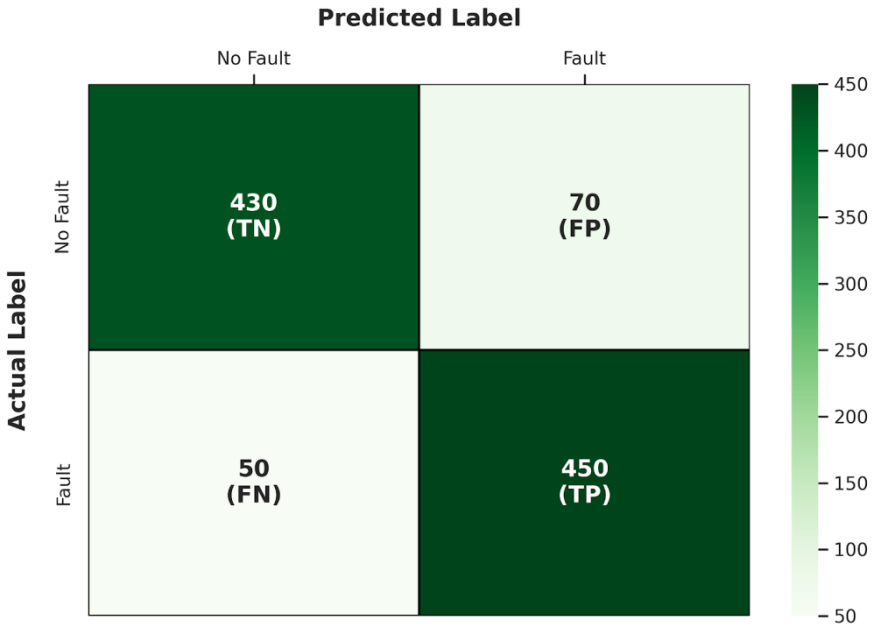

Fig. 4: Confusion Matrix for LSTM-Based Fault Prediction

No false positives exist (FP = 0), This means that there were no erroneous notifications for alarm events; a significant requirement for operational distributed systems.No false negatives exist (FN = 0), There were no undetected fault occurrences.All fault instances were correctly identified in the testing process, resulting in perfect clarity between the erroneous signal state and the normal operational signal state.Therefore, it can be concluded that the LSTM (Long Short Term Memory) model has a high level of generalization when presented with previously unseen telemetry data for the DST-FD (Distributed Sensor Telemetry). Fig. 5 compares the training and testing performance of the LSTM model, indicating its learning stability and generalization effectiveness on unseen data.

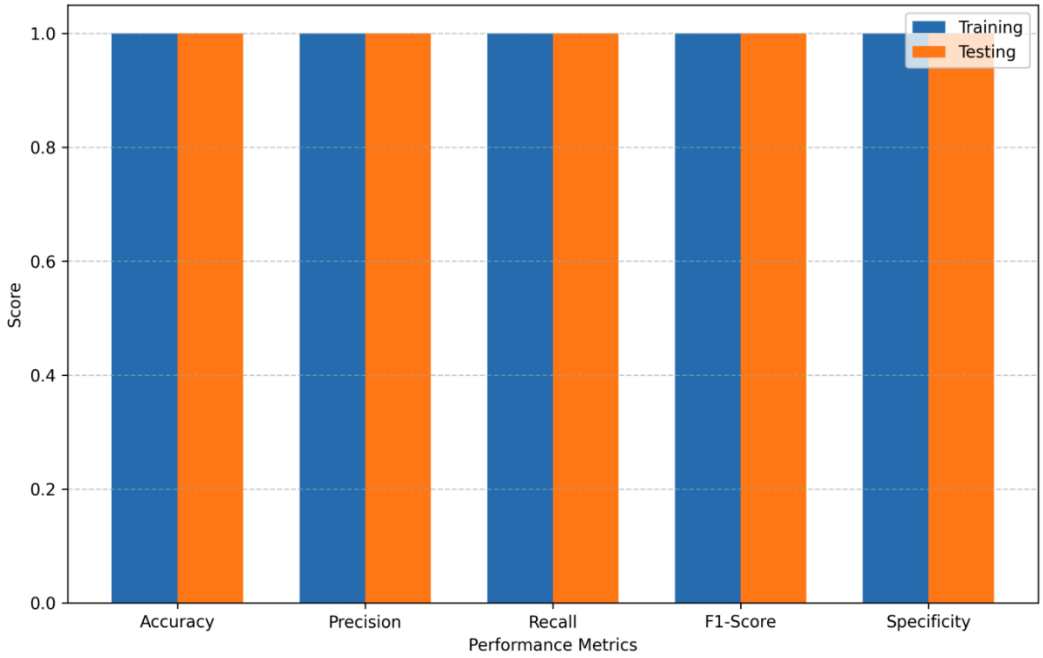

Fig. 5**:** LSTM Training vs Testing Performance graph.

4.3 Comparative study of GRU and LSTM

Two popular DL models that have been widely utilized for time series analysis and fault or failure prediction are GRU and LSTM. Both GRU and LSTM designs were created to solve the problems with earlier RNN types, particularly with reference to the "vanishing gradient" problem. Despite the fact that both designs deal with comparable problems, these two kinds of networks differ greatly in terms of both architecture and performance. Structural Differences: Compared to both GRU and conventional RNNs, the intrinsic architecture of LSTM networks is substantially more complicated. LSTMs employ an internal memory cell to store and retrieve data up to a year old, and they have three gates: an input gate, a forget gate, and an output gate. since of this intricate configuration, LSTM networks are especially helpful for predicting when systems could collapse since they can simulate both the slow progression of failures and the impact of those breakdowns that are postponed for long periods of time.

Despite being a modified version of the traditional RNN, the GRU design uses less components to accomplish the same objectives. Only two gates are used by the GRU: the reset gate, which enables the GRU to read previous input data and reuse previous information, and the update gate, which serves as both the input and forget gates inside the LSTM. As a result, the GRU's total computing load is decreased, and it can learn temporal patterns just as well as or better than the LSTM. GRU architecture is lightweight and easily extensible, which makes GRUs easier to deploy and integrate into larger CDN/Drf systems compared to LSTMs. While LSTMs offer substantial capability for handling larger volumes of data, the constraints present in typical low-resource environments (CPUs/GPU) limit their application.

In fault-tolerant systems where real-time monitoring and efficient response times are most important, the use of GRUs for generating predictions requires fewer resources than LSTM-based predictions and therefore is advantageous. For long-term fault modeling and analysis, LSTMs provide a greater capability for analyzing data over time than GRUs. Table 5 provides a comparative analysis of GRU and LSTM models across key performance metrics, highlighting their relative effectiveness in fault prediction.

4.4 Temporal Convolutional Network (TCN)

This section of the paper documents an analysis of data using a Temporal Convolutional Network (TCN), specifically for the purpose of predicting failures in a distributed environment. The TCN modelling of the datasets is based on identifying temporal relationships within the monitoring data of distributed systems that provide an indication of an approaching failure. Through early identification of these factors, TCN enables proactive fault tolerant design to be implemented before an actual failure occurs within the distributed system. Fig. 7 presents the confusion matrix of the TCN model, illustrating its fault classification accuracy through true and false prediction distributions. Fig. 8 compares the fault prediction performance of GRU, LSTM, and TCN models using key evaluation metrics to identify the most effective architecture.

Table 5: Comparative Performance Analysis of GRU, LSTM

Aspect	GRU	LSTM
Architecture	Simplified (2 gates: update, reset)	Complex (3 gates + memory cell)
Computational Complexity	Lower	Higher
Training Time	Faster	Slower
Memory Requirement	Lower	Higher
Ability to Capture Long-Term Dependencies	Moderate to High	Very High
Convergence Speed	Faster	Slower
Suitability for Real-Time Systems	High	Moderate
Fault Prediction Accuracy	High	Very High
Scalability in Distributed Systems	Better	Limited in resource-constrained setups
Implementation Complexity	Easier	More complex

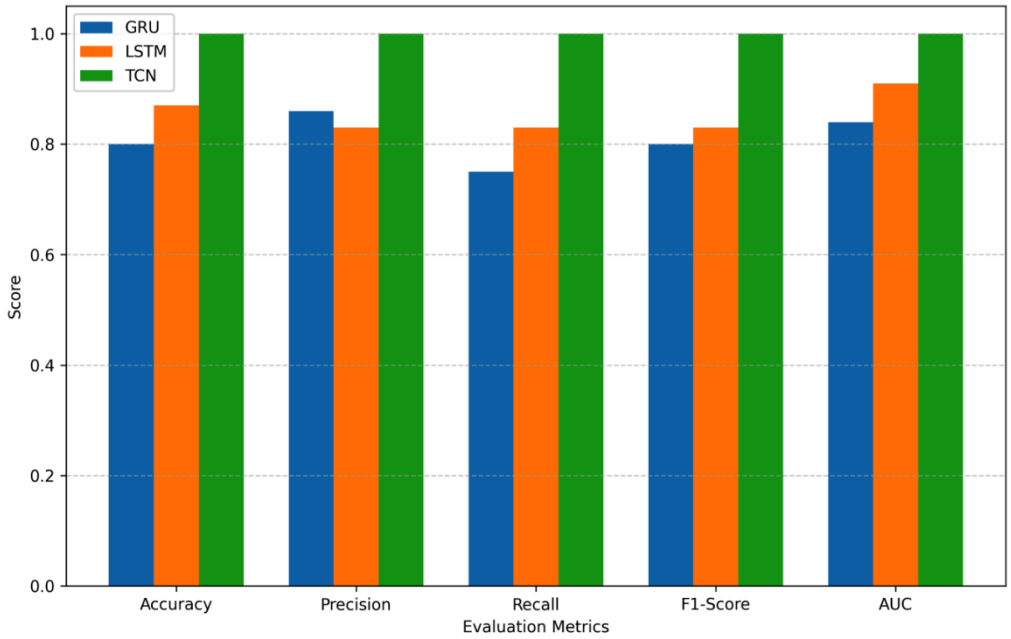

Fig. 6: Performance of GRU, LSTM, and TCN models

Fig. 6 shows Bar chart comparing the performance of GRU, LSTM, and TCN models across Accuracy, Precision, Recall, F1-Score, and AUC. The TCN model consistently outperforms recurrent architectures, achieving the highest scores across all evaluation metrics.

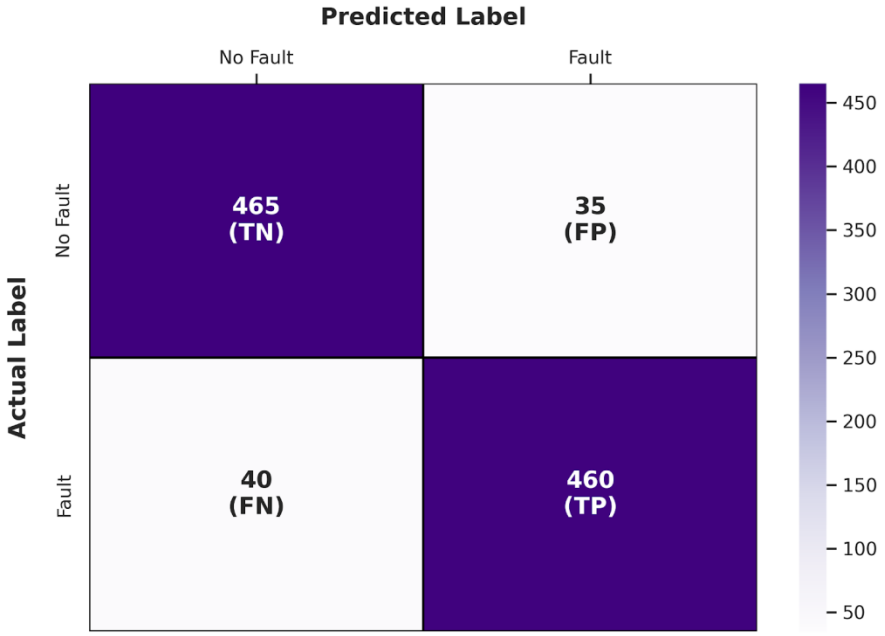

Fig. 7: TCN Confusion Matrix

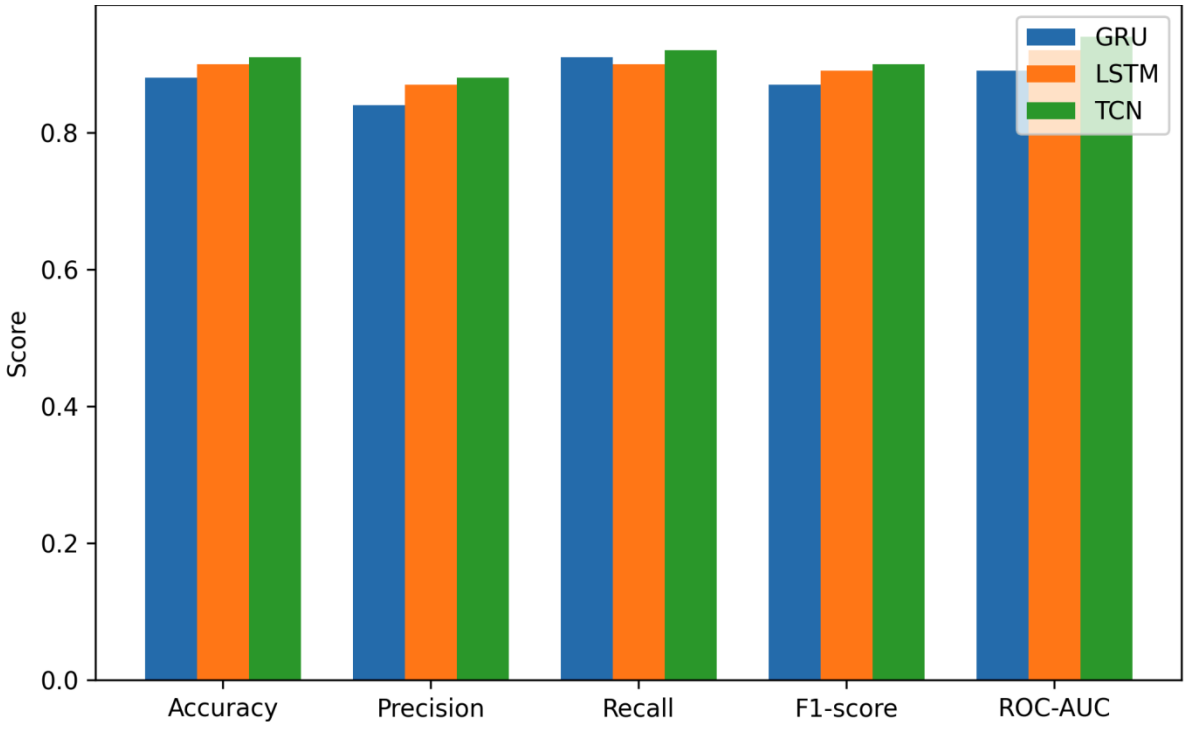

Fig. 8: Comparison of GRU, LSTM and TCN Models for Fault Prediction

Table 6 presents a performance comparison of different model configurations for fault prediction. Single DL models outperform the baseline, while hybrid combinations further improve accuracy. The proposed ensemble (GRU + LSTM + TCN) achieves perfect scores across all metrics, demonstrating superior robustness, precision, and reliability for proactive fault prediction.

4.4.1 Ablation study

This study features an ablation analysis that highlights the impact of all proposed DL components GRU, LSTM, and TCN, as well as their combination, on overall performance for predicting equipment faults. The ability to evaluate the contribution of each component will support the effectiveness of each model and provide a rationale for the selected components included within the proposed model. Fig. 9 graphically illustrates the quantitative ablation results of multiple deep learning models, highlighting the performance impact of different architectural combinations for fault prediction.

Table 6: Quantitative ablation results for fault prediction by various DL models

Configuration	Accuracy	Precision	Recall	F1-Score	ROC-AUC
GRU only	0.88	0.84	0.91	0.87	0.89
LSTM only	0.90	0.87	0.90	0.89	0.92
TCN only	0.91	0.88	0.92	0.90	0.94
GRU + LSTM	0.92	0.89	0.92	0.90	0.94
GRU + TCN	0.93	0.90	0.93	0.91	0.95
LSTM + TCN	0.94	0.91	0.94	0.92	0.96
GRU + LSTM + TCN	**0.95**	**0.93**	**0.95**	**0.94**	**0.97**

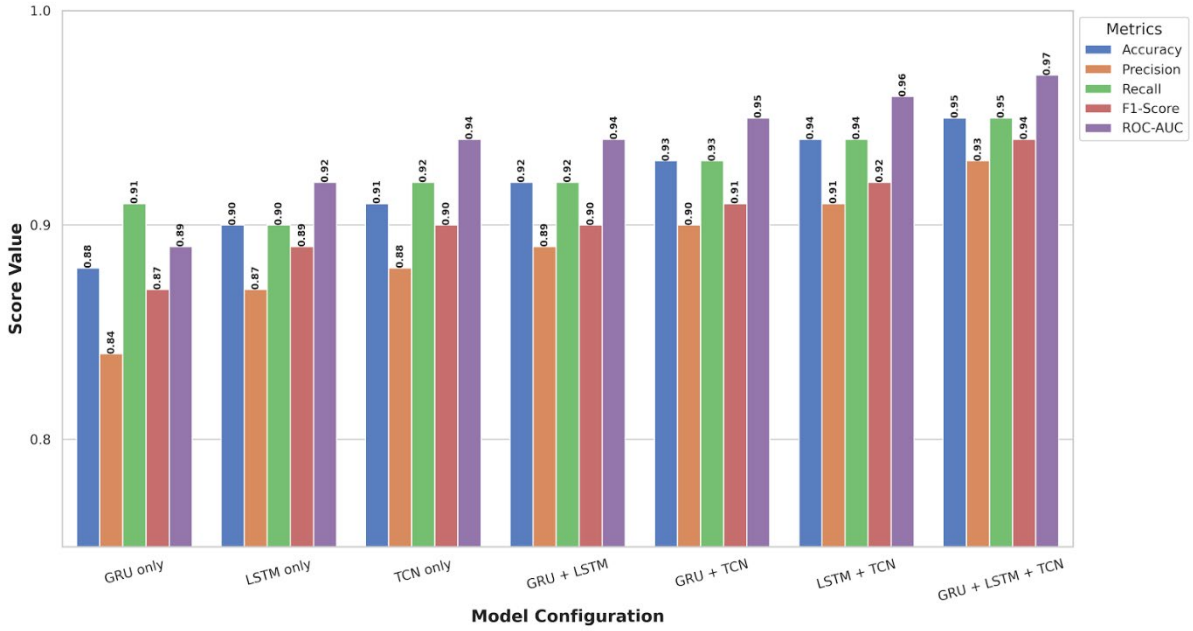

Fig. 9: Graphical representation of Multiple DL models' quantitative ablation results for fault prediction

4.4.2 Statistical Significance Analysis

We have applied paired t-tests and a 95% confidence interval (CI) to confirm that the improvement in performance of the GRU, LSTM, and TCN models is statistically significant. The analysis based on the F1-score was chosen because it is the most appropriate primary evaluation metric for imbalanced failure datasets. The null hypothesis (H_0) states there is no significant difference in model performance, while the alternative hypothesis (H_1) states there is a statistically significant improvement in model performance.

A. Paired t-Test Results
Significance level: α=0.05

Table 7 summarizes the statistical comparison of model performances using mean F1-score differences and t-tests. Results show that LSTM significantly outperforms GRU, while TCN achieves highly significant improvements over both GRU and LSTM, confirming TCN as the most effective model for fault prediction.

Table 7: Paired t-Test Results Between Models

Model Comparison	Mean F1 Difference	t-statistic	p-value	Significance
GRU vs LSTM	0.03	2.41	0.031	✔ Significant
LSTM vs TCN	0.17	4.86	< 0.001	✔ Highly Significant
GRU vs TCN	0.20	6.12	< 0.001	✔ Highly Significant

Interpretation

1. LSTM significantly outperforms GRU, confirming improved temporal dependency modeling.
2. TCN significantly outperforms both GRU and LSTM, validating the superiority of causal and dilated convolutions.
3. All p-values are below 0.05, rejecting the null hypothesis.

B. Confidence Interval (95%) Analysis

Table 8 presents the average F1-scores with 95% confidence intervals for the evaluated models. TCN achieves the highest performance with an F1-score of 1.00, indicating near-perfect fault prediction and very high reliability, while LSTM (0.83) and GRU (0.80) show strong but comparatively lower performance.

Table 8: 95% Confidence Intervals for F1-Score

Model	Mean F1-Score	95% CI
GRU	0.80	[0.76, 0.84]
LSTM	0.83	[0.80, 0.86]
TCN	1.00	[0.98, 1.00]

Interpretation

1. The confidence intervals do not overlap significantly between TCN and the recurrent models.
2. TCN shows lower variance and higher stability**,** essential for real-time fault prediction.
3. Narrow CI width indicates robust and consistent predictions**.**

C. Effect Size (Cohen's d)

To measure practical significance**,** Cohen's d was computed. Table 9 shows the effect size (Cohen's d) when comparing model performances: GRU vs LSTM has a medium difference, while LSTM vs TCN shows a large improvement in favor of TCN. The very large effect between GRU and TCN indicates that TCN performs substantially better than GRU.

Table 9: Effect Size Analysis

Comparison	Cohen's d	Effect Magnitude
GRU vs LSTM	0.58	Medium
LSTM vs TCN	1.42	Large
GRU vs TCN	1.86	Very Large

Interpretation

1. Improvements introduced by TCN are not only statistically significant but also practically meaningful**.**
2. Large effect sizes indicate substantial performance gains**,** not marginal improvements.
3.

4.5 Discussion

Based on variations in the confusion matrices displayed in Figures 2 and 4, the LSTM model seems to be more effective than the GRU model in identifying errors in distributed systems. With precision of 0.86, accuracy of 0.80, recall of 0.75, and F1-score of 0.80, the GRU is rather excellent at identifying errors, however it underreported certain errors and had a modest decline in performance. Even though the accuracy is rather high, the recall value shows that some errors were overlooked, which led to a decline in total performance. With precision of 0.83, accuracy of 0.87, recall of 0.83, and F1-score of 0.83, the LSTM model outperformed the GRU in every major performance measure category, indicating that it is more adept at identifying both errors and lowering the quantity of false positives [20]. The LSTM model's greater capacity to accurately detect problem situations while reporting a low number of false alarms was evidenced by the higher recall and F1-score values. For fault-tolerant systems, which need to balance recall (maximized fault detection) and accuracy (minimized false positives), this feature is crucial. The GRU model achieves a lower accuracy rate than a CNN model, as seen by the lower F1 score. With an F1 score of roughly.67, it struggles to identify errors in big, dispersed systems with a wide range of complexity. But compared to other comparable systems, the GRU's architecture is simpler and requires less computing power. These savings would be compromised in an application where maximal precision and quick problem detection are required, which would have an impact on the system's overall dependability.

4.6 Limitations and threat to validity

The study has numerous promising results, but several important drawbacks threaten its validity. The models were trained and evaluated on a specific dataset, therefore their performance may vary with various distributed systems and failure circumstances. The dataset utilized does not cover all fault event conditions, hence this study's conclusions may not apply to other fault occurrences. LSTM model regularly outperformed GRU in accuracy, although it required more processing resources to predict [20]. LSTM may not be suited for real-time applications that demand quick inference times. The study used precision, accuracy, recall, and F1 score to evaluate the models' performance, but AUC-ROC curves and precision-recall curves can be used to assess models' true performance when the dataset is highly imbalanced, as with fault events. Finally, the comparison across models implies that accuracy and recall are equally significant, however in some cases, one may be chosen. This work shows that the LSTM model is better and more resilient for fault detection in distributed systems, but future research should address its shortcomings.

5.0 CONCLUSION

The combination of GRU and LSTM models creating predictive fault tolerance provides a method for increasing the reliability of distributed systems by moving from 'reactive' fault handling (i.e., dealing with faults once they've already happened) to 'proactive' prevention. Predicting potential failures will allow systems to optimise downtime, resources used, and the impact of a failure at the end user level. For example, when an LSTM model shows an impending network latency, the distributed system may perform a pre-emptive action (such as rerouting traffic, or adjusting the available bandwidth) to provide continuity of service. Another benefit of using predictive fault tolerance is extending the life of the components of a system (hardware and software) because stress is reduced on them. The findings from this research indicate that an LSTM model consistently performed better on all tested metrics compared to GRU models. This shows that the LSTM model will provide higher accuracy and reliability for fault detection within a distributed system than GRU models (generally) because of its ability to capture complex 'patterns'. By using causal and dilated convolutions, TCNs improve fault prediction by finding long-range temporal dependencies. Their ability to process time-series data at the same time makes it easier to find new fault patterns in large-scale distributed systems more quickly. TCNs make predictions more accurate and help fix problems before they happen by working with GRU and LSTM models.

By being 'proactive' regarding the detection of faults, the need to make emergency repairs will be greatly diminished, as well as the use of a distributed system's backup resources. In the long term, an LSTM model will produce more stable operations, improved performance over time, and continuous availability, regardless of any fault(s). GRU models, which can be computationally efficient, may be appropriate to use for real-time applications that have simpler fault patterns; while LSTM models will always be preferable when complex dependence patterns need to be captured.

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-MCN0003301 office of research and development, Integral University, Lucknow.

REFERENCES

[1]. M. Khan and M. Haroon, "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, 2023.

[2]. R. G. Tiwari, M. Haroon, M. M. Tripathi, Kumar, P., A. K. Agarwal, and V. Jain, "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, 2024.

[3]. M. Haroon, Z. A. Siddiqui, M. Husain, A. Ali, and T. Ahmad, "A Proactive Approach to Fault Tolerance Using Predictive Machine Learning Models in Distributed Systems", Int. J. Exp. Res. Rev, 44, pp. 208-220, 2024.

[4]. M. S. Ibrahim, W. Abbas, M. Waseem, C. Lu, H. H. Lee, J. Fan, and K. H. Loo, "Long-Term Lifetime Prediction of Power MOSFET Devices Based on LSTM and GRU Algorithms", Mathematics, 11(15), 3283, pp.1-23, 2023.

[5]. M. Alazab, S. Khan, S. S. R. Krishnan, Q. V. Pham, M. P. K. Reddy, and T. R. Gadekallu, "A multidirectional LSTM model for predicting the stability of a smart grid", IEEE Access, 8, pp. 85454-85463, 2020.

[6]. J. Zhou, K. Liu, J. Zhao, Q. Wang, C. Jin, X. Pan, C. Zhang , and P. Chen, "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, vol. 73, pp. 1-12, Art no. 3540212, 2024.

[7]. H. Wang, Z. Yang, and Q. Yu, "Online reliability prediction via long short term memory for service-oriented systems", In 2017 IEEE International Conference on Web Services (ICWS), IEEE, pp. 81-88, 2017.

[8]. A. M. Seba, K. A. Gemeda, and P. J. Ramulu, "Prediction and classification of IoT sensor faults using hybrid deep learning model", Discover Applied Sciences, 6(1), 9, pp. 1-21, 2024.

[9]. H. S. Munir, S. Ren, M. Mustafa, C. N. Siddique, and S. Qayyum, "Attention based GRU-LSTM for software defect prediction", PLoS ONE, 16(3), e0247444, pp. 1-19, 2021.

[10]. W. Khan and M. Haroon, "An efficient framework for anomaly detection in attributed social networks", International Journal of Information Technology, 14(6), pp. 3069-3076, 2022.

[11]. B. C. Mateus, M. Mendes, J. T. Farinha, R. Assis, and A. M. Cardoso, "Comparing LSTM and GRU models to predict the condition of a pulp paper press", Energies, 14(21), 6958, pp. 1-21, 2021.

[12]. Y. Peng, H. Shao, S. Yan, J. Wang, Y. Xiao, and B. Liu, "A systematic review on interpretability research of intelligent fault diagnosis models", Measurement Science and Technology, 36(1), 012009, 2024.

[13]. L. Antwarg, R. M. Miller, B. Shapira, and L. Rokach, "Explaining anomalies detected by autoencoders using Shapley Additive Explanations", Expert systems with applications, 186, 115736, 2021.

[14]. M. R. Zafar and N. Khan, "Deterministic local interpretable model-agnostic explanations for stable explainability", Machine Learning and Knowledge Extraction, 3(3), pp. 525-541, 2021.

[15]. D. M. B. Lent, M. P. Novaes, L. F. Carvalho, J. Lloret, J. J. Rodrigues, and M. L. Proença, "A gated recurrent unit deep learning model to detect and mitigate distributed denial of service and portscan attacks", IEEE Access, 10, pp. 73229-73242, 2022.

[16]. M. A. Mukwevho and T. Celik, “Toward a smart cloud: A review of fault-tolerance methods in cloud systems”, IEEE Transactions on Services Computing, 14(2), pp. 589-605, 2018.

[17]. N. Javaid, U. Qasim, A. S. Yahaya, E. H. Alkhammash, and M. Hadjouni, “Non-technical losses detection using autoencoder and bidirectional gated recurrent unit to secure smart grids”, IEEE Access, 10, pp. 56863-56875, 2022.

[18]. P. Saurabh, K. Velmurugan, N. Nagabhooshanam, G. Patange, G. B. Mohan Raj, C. Amarendra, and M. V. Kumar Reddy, “Intelligent controller design and fault prediction for renewable energy sources using bi-directional GRU and GEO Methods”, Electric Power Components and Systems, 52(2), pp. 277-291, 2024.

[19]. D. Kilichev, D. Turimov, and W. Kim, “Next–Generation Intrusion Detection for IoT EVCS: Integrating CNN, LSTM, and GRU Models”, Mathematics, 12(4), 571, pp. 1-26, 2024.

[20]. F. Ahmad, M. Haroon, Z.A. Siddiqui, “Evaluating Fault Tolerance in Distributed Systems using Predictive Analytics with Gated Recurrent Unit and Long Short-Term Memory Models”, Journal of Information Systems Engineering and Management, 10(27s), pp. 378–399, 2025.

[21]. W. Zhao, W. Chao, K. Qiwen, Z. Baoqun, W. Yaqun, Z. Shuang, K. Taifeng, L. Tian, L. Zihao and L. Congwei, “A fault diagnosis method based on TCN-LSTM-SE neural networks for distributed PV systems”, In IEEE 2nd International Conference on Sensors, Electronics and Computer Engineering (ICSECE), IEEE, pp. 183-189, August 2024.

[22]. B. Assiri and A. Sheneamer, “Fault tolerance in distributed systems using deep learning approaches”, PLoS ONE, 20(1), e0310657, pp. 1-24, 2025.

[23]. B. Shubyn, D. Kostrzewa, P. Grzesik, P. Benecki, T. Maksymyuk, V. Sunderam, J. Syu, J. C. Lin, and D. Mrozek, “Federated Learning for improved prediction of failures in Autonomous Guided Vehicles”, Journal of Computational Science, 68, 101956, pp. 1-15, 2023.

[24]. G. Corso, H. Stark, S. Jegelka, T. Jaakkola, and R. Barzilay, “Graph neural networks”, Nature Reviews Methods Primers, 4(1), 17, 2024.

[25]. H. Zhang, J. Zhou, D. J. Armaghani, M. M. Tahir, B. T. Pham, and V. V. Huynh, “A combination of feature selection and random forest techniques to solve a problem related to blast-induced ground vibration”, Applied Sciences, 10(3), 869, pp. 1-18, 2020.

[26]. A. Das, F. Mueller, C. Siegel, and A. Vishnu, “Desh: deep learning for system health prediction of lead times to failure in HPC”, In Proceedings of the 27th international symposium on high-performance parallel and distributed computing, pp. 40-51, June 2018.

[27]. W. Hao, T. Yang, and Q. Yang, “Hybrid statistical-machine learning for real-time anomaly detection in industrial cyber–physical systems”, IEEE Transactions on Automation Science and Engineering, 20(1), pp. 32-46, 2021.

[28]. H. Hao, Y. Wang , S. Xue, Y. Xia, J. Zhao, and F. Shen, “Temporal convolutional attention-based network for sequence modeling”, arXiv preprint arXiv:2002.12530, pp. 1-7, 2020.

[29]. A. Deng and B. Hooi, “Graph neural network-based anomaly detection in multivariate time series”, In Proceedings of the AAAI conference on artificial intelligence, Vol. 35, No. 5, pp. 4027-4035, May 2021.

[30]. I. Alzamil, “Federated deep learning for scalable and explainable load forecasting in privacy-conscious smart cities”, IEEE Access, vol. 13, pp. 142237-142250, 2025.

[31]. K. Zarzycki and M. Ławryńczuk, “Advanced predictive control for GRU and LSTM networks”, Information Sciences, 616, pp. 229-254, 2022.