ESP-JETA
ESP Journal of Engineering & Technology Advancements
ISSN: 2583-2646 / Volume 3 Issue 4 December 2023 / Page No: 138-148
Paper Id: JETA-V3I8P116 / Doi: 10.56472/25832646/JETA-V3I8P116

Orignai Article

Artificial Intelligence-Powered Optimization of Industrial IoT Networks Using Python-Based Machine Learning

Ruchi Patel
Independent Researcher.

Received Date: 26 November 2023 Revised Date: 09 December 2023 Accepted Date: 24 December 2023

Abstract: *Through sophisticated monitoring and real-time data sharing, industrial Internet of Things (IIoT) networks are revolutionizing smart manufacturing. With the increasing complexity of IIoT infrastructures, traditional methods often fail to deliver fault-resilient and scalable performance. With the goal of improving system dependability and operational safety, this paper presents a method to sensor system defect detection and classification based on deep learning (DL), for the purpose of improving IIoT network defect prediction. Using PyTorch and Scikit-Learn in a high-throughput Python industrial simulation environment. The Long Short-Term Memory (LSTM) model had a better fault detection accuracy of 99.33% than the Random Forest (RF) (99%), Multilayer Perceptron (MLP) (96.6%), and FuzHD++ (92%). According to the confusion matrix, the model showed minimal false positives (FP) and false negatives (FN) with 99.38% accuracy, 99.87% recall, and 99.66% F1-score. The robustness and generalizability of the model were demonstrated by the loss plots and training and validation accuracy, which demonstrated strong convergence with minimal overfitting. This method improves operational uptime and system dependability in IIoT scenarios while simultaneously improving real-time issue detection. Overall, the findings illustrate the efficacy of LSTM-based deep learning in enhancing defect resilience and predictive maintenance in intelligent industrial networks.*

Keywords: *Industrial Internet of Things (IIoT), Fault Detection, Predictive Maintenance, Sensor Networks, Network Optimization, Python, Machine Learning, Intel Berkeley Research Lab Sensor Data.*

I. INTRODUCTION

In the era of digital transformation, industries are increasingly embracing automation, connectivity, and data-driven intelligence to enhance efficiency, productivity, and innovation. As operational complexity and global competition continue to grow, traditional industrial management methods are proving inadequate [1]. To address these challenges, organizations are adopting advanced technologies capable of delivering smarter, more responsive systems that adapt to dynamic environments and enable real-time decision-making. The IIoT is a paradigm that brings physical assets, control systems and digital technologies together through sensor-rich networks and is a central enabler of this transformation. These IIoT networks provide a continuous, low-latency [2], low-bandwidth medium for data acquisition and communication across industrial environments, giving deep insight into system behavior and supporting predictive maintenance as well as process automation [3][4] and energy optimization. But modern IIoT deployments are complex and scaled and the data source is unreliable, response time is slow and system is not intelligent enough to be optimized.

In order to reach the maximum potential of IIoT, there is a growing need for intelligent optimization techniques to manage sensor data, detect faults proactively and adapt system parameters autonomously [5]. In general, traditional rule base methods which rely on pre-defined thresholds and static configurations, fail to cope with dynamic industrial conditions and large volumes of sensor data. As a result, they tend to generate high FP or miss sublet system anomalies which sometimes causes inefficiencies and failures [6]. To fill this gap, AI and ML are used that provide robust resources for pattern recognition, anomaly detection and predictive analytics. Because AI systems can use past and present sensor data to learn to recognize normal and abnormal behaviour, they can improve system performance and they can make better decisions [7][8]. High system uptime, reliability and responsiveness for success in IIoT environments makes these capabilities particularly important.

The Python programming language forms to be central to the practical implementation of these AI-driven solutions, for it offers a robust ecosystem of open source libraries and frameworks written specifically for data science and ML [9]. Scikit-learn, TensorFlow, Keras, XGBoost, as well as Panda's support end to end building of intelligent systems (data preprocessing, visualization, model training, deployment, etc.). Because of its simplicity, versatility and ability to integrate with other systems [10], Python is a good choice to build and implement AI applications in the IIoT infrastructures. The intent of this research is to build on Python's development ecosystem and data driven modelling to create smart systems that enhance operational resilience, reduce downtime and maximize sensor network performance. This work shows that AI is revolutionary in manufacturing and how useful Python is for building intelligent, scalable IIoT applications.

A. Motivation and Contributions of the Study

This study's motivation stems from current IIoT developments. Which tend to be increasingly massive and complex and play an increasingly important role in modern day industrial operations. The problem is to ensure reliability, efficiency and fault tolerance, as these networks grow. Traditional network management approaches frequently fail because of the dynamic and data-intensive nature of IIoT settings, which results in system inefficiencies, unexpected failures and operational downtime. AI particularly ML integration into the IIoT networks offers a transformative possibility for IIoT network optimization to achieve predictable faults prediction and proactive system management. This research contributes the following key contributions:

- The data set that was used for this work was a real time sensor data from the Intel Berkeley Lab Research, where defective sensors were identified using measurements of temperature, humidity, light, and voltage.
- A robust preprocessing pipeline that includes data cleaning, correlation analysis, SMOTE-based class balancing and Min Max normalization was developed.
- Applied time-series feature engineering techniques, including rolling statistics, to extract time-related trends from the sensor data.
- Proposed a neural network model with LSTM to efficiently learn temporal connections and identify problematic sensor behavior.
- Visualization of training and validation loss curves demonstrating effective model convergence with minimal overfitting, alongside confusion matrix plots highlighting the robustness of the model in accurately detecting sensor states that are both healthy and malfunctioning with little FP and FN.
- Demonstrated model effectiveness using a standard matrix that include recall, accuracy, precision, and f1-score.

B. Structure of the Paper

The structure of this paper is as follows: Section II reviews related work on AI-based optimization in IIoT networks. Model creation and data pretreatment are covered in Section III clarification of the approach. The study findings are covered in Section IV. Key findings and recommendations for future research to increase IIoT dependability using AI are presented in Section V.

II. LITERATURE REVIEW

This section reviews IIoT networks using ML, focusing on fault detection, predictive maintenance, and efficient resource management to enhance reliability and system performance. Some of the previous works are:

Praveenchandar et al. (2022) invention focuses on Industrial accidents can be avoided by keeping an eye on and managing dangerous poisonous gases and environmental factors. The system's core microcontroller is an Arduino UNO R3 board, which is linked to the cloud by a number of sensors. For intelligent prediction, AI and ML are employed. Using hybrid hidden Markov and AI models, the system outperforms current methods in mistake detection. The hybrid ANN and HMM defect detection techniques have a FPR of 0.01% [11].

Kumar et al. (2022) Describe a low-cost multi-sensor DAQ system that can detect defects in 3D printed products. The apparatus collects Real-time multi-sensor data is acquired from sound, vibration, and current sensors using an Arduino microcontroller. The authors use the CNN model and the chi-square approach to examine how fault circumstances affect the recorded sensor data. The CNN model showed the potential of FDM-based 3D printing in Industry 4.0 by classifying 94% accuracy in both normal and fault state data [12].

Hojabri et al. (2022) assesses how well ML models perform in detecting faults in PV modules. They discuss eight different faults and their impacts on output, using literature review and simulation. The results show indicates for seven chosen classes, NN achieves 93% classification accuracy, highlighting the promise of inexpensive sensors and ML models for enhanced defect detection [13].

Dang et al. (2021) uses spatial-temporal correlation to identify anomalous sensory information. They create an MSC technique that improves the accuracy and FPR of online detection. Furthermore, for both temporary and permanent artificial abnormalities, they develop a general anomaly model. In comparison to current systems, MSC delivers a 5% lower FPR and an 8% higher ACC [14].

Marathe et al. (2021) introduces Current Sense is a fingerprint for sensors that records the electrical properties of hardware parts. Without context, this fingerprint assesses the health of the sensors, corrects for drift, and identifies catastrophic flaws. Numerous environmental sensors may be used using this non-intrusive method. Current Sense surpasses conventional anomaly detectors and accounts for sensor drift problems by 86%, according to the study, which uses 51 sensors spread across many cities [15].

Viktoros, Michael and Polycarpou (2020) explores the development of a small fault dictionary for IoT-enabled CPS that uses Multiple sensor failure isolation and detection using a real-time model. Because of its tremendous scalability, the method requires less memory and enables fault isolation in linear time. Because ZDD are used in the study as a fault dictionary, the examined systems' fault isolation procedure takes 0.002 to 0.012 seconds [16].

Table I presents a comparative summary of prior studies on Industrial IoT networks using ML, focusing on techniques, performance, limitations, and future research directions.

Table 1 : Summary of Reviewed Works Industrial IOT Networks Based Machine Learning Techniques

Author	Methodology	Data	Key Findings	Limitation	Future Work
Praveenchandar et al. (2022)	Hybrid HMM + ANN for fault detection; Arduino-based real-time monitoring system using multiple gas sensors	Real-time toxic gas sensor data (NO2, CO, SO2, etc.), Arduino + ESP8266 IoT system	Achieved 0.01% false positive rate in sensor fault detection; real-time toxic gas monitoring	Limited scalability, specific to toxic gas environments	Extend to scalable multi-environment IIoT networks with advanced AI models
Kumar et al. (2022)	CNN model trained with chi-square feature selection; K-means clustering; Arduino-based DAQ	Multi-sensor (vibration, sound, current) data from FDM 3D printing	94% classification accuracy for fault diagnosis in 3D printing	Focused on offline data analysis and printing-specific faults	Integrate real-time ML for broader industrial machinery diagnostics
Hojabri et al. (2022)	ML classification (including NN) on labelled panel-level electrical data	Real and simulated PV module data with 8 fault types	93% accuracy in PV fault classification using low-cost sensors	Model complexity may limit deployment in resource-constrained hardware	Develop lightweight models optimized for embedded IIoT devices
Dang et al. (2021)	Monotone Subtrend Correlation (MSC); SPE control limit estimation	The dataset from the Intel Berkeley Research Lab (IBRL)	8% higher accuracy, 5% lower FPR; detects short and long-term anomalies in real time with minimal data	Needs artificial anomaly generation for evaluation; specific tuning for different environments	Extend MSC for multi-source sensor fusion and Python-embedded IIoT deployments
Marathe et al. (2021)	Sensor fingerprinting based on electrical characteristics (Current Sense)	51 air pollution sensors in real-world over 8 months	98% F1 score for fault detection; 86% drift compensation	Requires access to sensor internals; limited to pollution monitoring	Generalize Current Sense for heterogeneous industrial sensors
Viktoros, Michael and Polycarpou (2020)	Model-based multiple sensor fault detection using ZDD (Boolean encoding)	Simulated fault data for CPS/IoT environments	Fault isolation within 0.002s–0.012s; reduced memory overheads	Still theoretical; lacks implementati on in real IIoT networks	Combine ZDD with live ML-based anomaly prediction in Python-based platforms

A. Research gaps/ problems

Existing literature on AI-powered optimization in Industrial IoT (IIoT) systems highlights diverse innovations in sensor-based fault detection using machine learning. Several studies have developed real-time monitoring systems leveraging Arduino-based multi-sensor data acquisition units that collect environmental, acoustic, and electrical signals. These setups often employ hybrid AI models such as ANN-HMM, CNNs, and spatial-temporal correlation techniques to improve the accuracy of anomaly detection and minimize false positive rates. Techniques like Current Sense use electrical fingerprinting for non-intrusive fault diagnosis and sensor drift correction, demonstrating high robustness across distributed environments. According to existing work, numerous challenges persist in the optimization of Industrial IoT (IIoT) networks, particularly related to data quality, preprocessing inconsistency, class imbalance, insufficient use of advanced deep learning models, and limited performance evaluation frameworks. Prior studies often lack standardized pipelines for noise reduction, missing value handling, and feature engineering, while many overlook the severe impact of class imbalance on fault detection. Additionally, the use of shallow or non-sequential models fails to capture the temporal dependencies inherent in IIoT time-series data. To overcome the limitations of prior studies, the proposed work introduces a scalable AI-driven approach using Research Lab Sensor Data. It includes thorough preprocessing—data cleaning, SMOTE for class balancing, time-based feature extraction, and Min-Max scaling. An LSTM model is trained to capture temporal patterns for fault detection. Performance is evaluated using precision, recall, F1-score, and accuracy, ensuring robust and balanced results. This streamlined pipeline addresses key gaps in data handling, model training, and performance evaluation for IIoT network optimization.

III. METHODOLOGY

The methodology for optimizing IIoT networks is outlined in the provided in The process starts with gathering information from the Intel Berkeley Research Lab, after which it goes through many preparatory steps. Preprocessing steps included data cleaning (removing duplicates, handling missing values), class balancing using SMOTE to address imbalance between "Normal" and "Faulty" classes, and feature engineering involving time-based features and rolling statistics. Feature scaling was applied using Min-Max normalization to ensure consistent ranges across variables. The dataset was partitioned 20% are testing sets and 80% are training sets to facilitate predictive modelling. The training data is used to build models, specifically leveraging LSTM networks are ideal for IIoT systems' problem detection and time-series prediction. The model's performance is assessed using industry-standard criteria including F1-score, recall, accuracy, and precision. This comprehensive pipeline ensures effective fault detection and optimization of IIoT networks through AI-driven predictive analytics.

A. Data Collection

The Intel Berkeley Research Lab's 54 sensors are used to gather the data. Mica2Dot sensors equipped with weatherboards recorded temperature, light, voltage, humidity, and time-stamped topological data every 31 seconds. Built on the TinyOS platform, The TinyDB in-network query processing technology was used to acquire the data. Figure 2 depicts the compacted shape of the sensors.

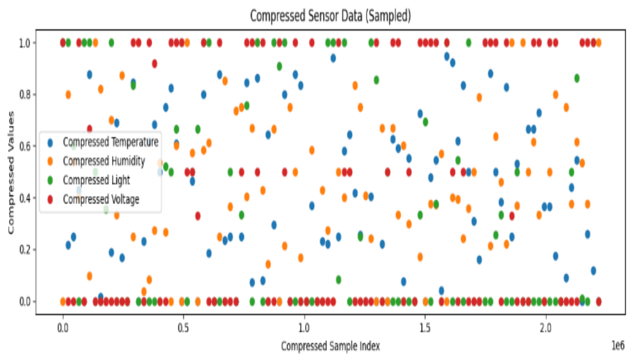

Figure 1 : Scatter Plot for Compressed Data

Figure 1 visualizes the distribution of four different types of sensor readings Compressed Temperature, Compressed Humidity, Compressed Light, and Compressed Voltage against a "Compressed Sample Index." Each data point represents a sampled sensor reading, with different colors distinguishing the sensor types. The y-axis, "Compressed Values," ranges from 0.0 to 1.0, indicating that the sensor data has likely been normalized or scaled. The x-axis, "Compressed Sample Index," spans from 0 to approximately 2.0 x 106, suggesting a large dataset of sampled and compressed sensor readings over time or a sequence of measurements. The scattered plot shows variability and patterns in each sensor's compressed values, helping to visually assess their behavior and trends.

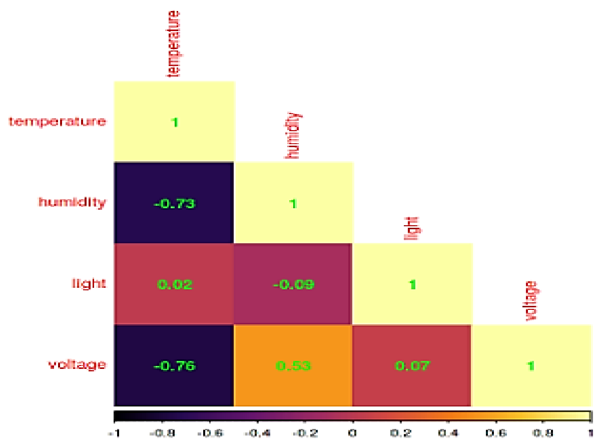

Figure 2: Correlation Matrix of Features

Figure 2 is a correlation matrix that uses color intensity and numerical values to depict the strength and direction of these correlations, falling between -1 and +1. There's a significant inverse relationship between temperature and humidity (-0.73), while voltage and humidity have a moderate positive correlation (0.53). Light exhibits weak linear relationships with the other measured environmental parameters.

B. Data Preprocessing

Data preparation is an essential process that converts raw sensor data into a format that machine learning models can use, preparing it for analysis. I want to make the model more efficient and perform better by eliminating noise and making it more consistent. Standard preprocessing procedures often involve:

C. Data Cleaning

The process of collecting and analyzing data cannot happen without data cleaning. This is because it guarantees the accuracy and quality of data which is vital for making well thought through decision and taking action. Below are given the key data cleaning steps:

- Remove duplicates: Data deduplication is the process of eliminating duplicate data from a dataset since sensors may gather data more than once.
- Handle missing values: Determining and dealing with missing data using methods like removal, interpolation, or imputation, depending on the kind and amount of the missing variables.

D. Class Balanced Using SMOTE

An imbalance in data distribution occurs when there are more instances from the majority class than the minority class. Here, the sampling-level technique is the way to go for accurate classification by ensuring that classes are distributed evenly [17]. A common technique for oversampling is the SMOTE. The linear interpolation technique is used between Synthetic minority class instances are created using the original minority class instances. Balanced data distribution depicts in below:

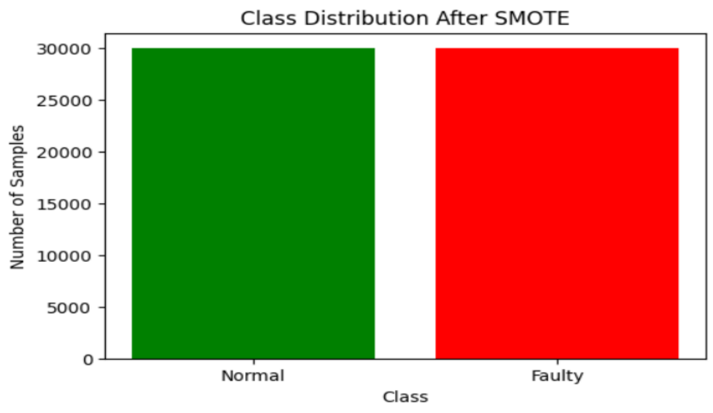

Figure 3 : Class Distribution After Smote

Figure 3 illustrates the balance between two classes, "Normal" and "Faulty," after the application of the SMOTE algorithm. Each class now contains approximately 30,000 samples, indicating that by oversampling ensure that the minority group receives an equal number of samples as the dominant group ("Normal"), SMOTE has effectively solved the problem of class imbalance. Since properly training ML models necessitates avoiding the issue of the model's bias towards the majority class and ensuring that instances of both classes are categorized correctly, it is imperative that data be dispersed evenly across all classes.

E. Feature Engineering

The feature engineering stages designed to enhance predictive accuracy, followed the feature engineering process. To first extract the local temporal patterns inherent in the sensor data, time based and rolling window features were created, chiefly consisting of statistical measures such as rolling mean and standard deviation. Next, a feature selection method was applied to select and eliminate the most relevant features thereby improving the model efficiency and substituting to minimize the noise. Finally, the reformulated the time series data into a supervisory learning problem using the Keras Time Series Generator API for efficient training of the proposed DL architectures on sequential sensor measurements.

F. Feature Scaling

Scaling techniques play a critical role in domain data preprocessing for ML, to make a predictive model more robust and effective [18]. It ensures that each feature contributes evenly to the model and works especially well when features have varying scales. According to the Min-Max Scaler, the converted data is normally distributed with a mean of 0 and a variable of 1. Equation (1) is used to scale the characteristics to a certain range, usually between 0 and 1:

$$Scaled\ Feature = \frac{Feature - \min\,(Feature)}{\max(Feature) - \min\,(Feature)} \quad (1)$$

Where, min = dataset's minimum feature value; max = dataset's maximum feature value.

Data preprocessing plays a critical role in data analysis and machine learning projects. In this study, we carried out data transformation involving handling missing or damaged data and converting data into a suitable format for machine learning algorithms. Missing values were carefully imputed to avoid bias and maintain prediction accuracy, while categorical variables were label-encoded to convert them into numerical values. Additionally, continuous numerical features (Total Charges, Monthly Charges, Tenure Months) were normalized using Min-Max Scaler to fit within a predefined range, typically 0-1.

These preprocessing steps ensure that the data is appropriately prepared for the machine learning algorithms used in this study Data preprocessing plays a critical role in data analysis and machine learning projects. In this study, we carried out data transformation involving handling missing or damaged data and converting data into a suitable format for machine learning algorithms. Missing values were carefully imputed to avoid bias and maintain prediction accuracy, while categorical variables were label-encoded to convert them into numerical values. Additionally, continuous numerical features (Total Charges, Monthly Charges, Tenure Months) were normalized using Min-Max Scaler to fit within a predefined range, typically 0-1.

These preprocessing steps ensure that the data is appropriately prepared for the machine learning algorithms used in this study Data preprocessing plays a critical role in data analysis and machine learning projects. In this study, we carried out data transformation involving handling missing or damaged data and converting data into a suitable format for machine learning algorithms. Missing values were carefully imputed to avoid bias and maintain prediction accuracy, while categorical variables were label-encoded to convert them into numerical values. Additionally, continuous numerical features (Total Charges, Monthly Charges, Tenure Months) were normalized using Min-Max Scaler to fit within a predefined range, typically 0-1.

These preprocessing steps ensure that the data is appropriately prepared for the machine learning algorithms used in this study.

Data preprocessing plays a critical role in data analysis and machine learning projects. In this study, we carried out data transformation involving handling missing or damaged data and converting data into a suitable format for machine learning algorithms. Missing values were carefully imputed to avoid bias and maintain prediction accuracy, while categorical variables were label-encoded to convert them into numerical values. Additionally, continuous numerical features (Total Charges, Monthly Charges, Tenure Months) were normalized using Min-Max Scaler to fit within a predefined range, typically 0-1.

These preprocessing steps ensure that the data is appropriately prepared for the machine learning algorithms used in this study.

Data preprocessing plays a critical role in data analysis and machine learning projects. In this study, we carried out data transformation involving handling missing or damaged data and converting data into a suitable format for machine learning algorithms. Missing values were carefully imputed to avoid bias and maintain prediction accuracy, while categorical variables were label-encoded to convert them into numerical values. Additionally, continuous numerical features (Total Charges, Monthly Charges, Tenure Months) were normalized using Min-Max Scaler to fit within a predefined range, typically 0-1.

These preprocessing steps ensure that the data is appropriately prepared for the machine learning algorithms used in this study.

G. Data Partitioning

In order to do predictive analysis, two subsets are created from the dataset. 20% of the data from the training set is used to develop and train the ML model, while 80% is reserved for testing purposes. and its ability to generalize within the context of Industrial IoT network optimization.

H. Proposed Long-Short-Term Memory (LSTM)

LSTM networks employ a gating technique that selectively stores certain data in memory while discarding other data by adjusting the weights linked to each gate [19]. The gates, which regulate the flow of information, are neural networks [20]. These consist of Equation (2-4) a forget gate that determines what data from the past should be forgotten, Equation (5-6) an input gate that determines what data should be stored in the present memory, and Equation (7-8) an output gate.

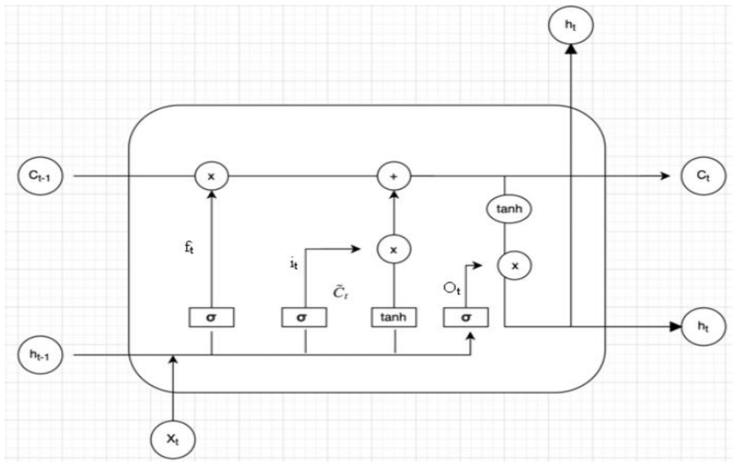

Figure 4 : Long-Short-Term Memory Structure

The memory cell, which is represented by the top line in the LSTM diagram, is the weight = 1 feature in the LSTM cell seen in Figure 4. LSTM makes it simple to add or delete data from the memory cell by utilizing structures known as gates. These gates determine what data to output 34 by combining a pointwise multiplication operation with a sigmoid neural net layer. The gates are calculated in this way:

$$f^{(t)} = \sigma\left(W_{hf}\left[h^{t-1}, x^{(t)}\right] + bf\right) \quad (2)$$

$$i^{(t)} = \sigma\left(W_{hi}\left[h^{t-1}, x^{(t)}\right] + bi\right) \quad (3)$$

$$o^{(t)} = \sigma\left(W_{ho}\left[h^{t-1}, x^{(t)}\right] + bo\right) \quad (4)$$

The gates employ the sigmoid activation function and to limit the values to [0, 1]. A gate value of 0 totally erases the information, whereas a value of 1 fully keeps it. The gates are used to compute an intermediate memory that is calculated before the current memory.

$$\hat{C}^{(t)} = tanh\left(W_{hc}\left[h^{t-1}, x^{(t)}\right] + bc\right)(4) \quad (5)$$

$$C^{(t)} = f^{(t)} \otimes C^{(t-1)} + i^{(t)} \otimes \tilde{C}^{(t)}) \quad (6)$$

Where, stands for multiplication by elements. Thus, the LSTM cell's output and current state may be calculated as follows:

$$h^{(t)} = o^{(t)} \otimes tanh(C^{(t)} \quad (7)$$

$$y^{(t)} = g\left(W_y h^t + by\right) \quad (8)$$

The output activation function, denoted by function g, is using either linear or SoftMax for regression tasks or classification tasks, respectively. Key hyperparameters for LSTM include the number of hidden units (neurons), which controls the depth and learning complexity are influenced by the model's capacity and layer count. Additionally, the learning rate, batch size, dropout rate, and sequence length (timesteps) are important hyperparameters.

I. Performance Metrics

The suggested strategy is evaluated using many performance measures to determine its efficacy. Some examples of these metrics include recall and loss, accuracy and precision, and F-measure. The purpose of these measures is to separate sensor data that is defective from data that is not wrong.

a) *Accuracy*

It is the most popular and maybe the first option for assessing how well an algorithm performs in classification scenarios [21]. It is defined by Equation (9) as the ratio of accurately recognized data items to all observations.

$$Accuracy = \frac{TP+TN}{TP+FN+FP+TN} \quad (9)$$

b) *Precision*

The accuracy of an algorithm is the percentage of observations that it predicts will be positive. Equation (10) states that the accuracy may be calculated by dividing the number of TP by the total of FP and TP:

$$Precision = \frac{TP}{TP+FP} \quad (10)$$

c) *Recall*

Recall is the percentage of observations that turn out to be positive. As per Equation (11), recall is calculated by dividing the number of TP by the total of FN and TP:

$$Recall(Rc) = \frac{TP}{TP+FN} \quad (11)$$

d) *F1-score*

The F1-score evaluates an algorithm's performance by taking into consideration both accuracy and recall. Equation (12) in mathematics represents it as the accuracy and memory harmonic mean:

$$F1\ score(F1) = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (12)$$

The numbers TP, T, N, FP, and FN represent the number of TP, TN, and FN samples, respectively.

e) *loss*

The loss function was used to quantify the discrepancy between the predicted and actual outcomes. In Equation (13), the numbers C and S represent the number of classes and samples, respectively, provides an illustration of this calculation, $(S \in C)$ is the samples associated with each class.

$$Loss = -\log p(S \in C) \quad (13)$$

The performance metrics are applied to model’s output on the test dataset in order to assess the reliability of the model.

IV. RESULT ANALYSIS AND DISCUSSION

In this study, the effectiveness of the proposed machine learning approach for enhancing and optimizing IIoT network resilience is examined. Their studies are carried out using PyTorch and Scikit, two Python-based ML packages. Use a 64-bit industrial simulation environment based on Ubuntu. The simulation system used an Intel Xeon processor with 64GB of RAM to simulate high-throughput industrial IoT scenarios. Fault prediction in IIoT networks was amazingly achieved by the LSTM model based on Table II. It achieved an accuracy of 99.33%, indicating a high level of precision in identifying system anomalies. The accuracy of 99.38% suggests a moderate FPR, but the recall of 99.87% shows how well the model can identify actual issues. A 99.66% F1-score verifies a steady and well-balanced performance. These results substantiate the LSTM model’s suitability for AI-powered optimization and fault resilience in Industrial IoT environments.

Table 2 : Experiment Results of Proposed Model for IoT Network Optimization

Performance Metrics	**Long Short-Term Memory (LSTM)**
Accuracy	99.33
Precision	99.38
Recall	99.87
F1-score	99.66

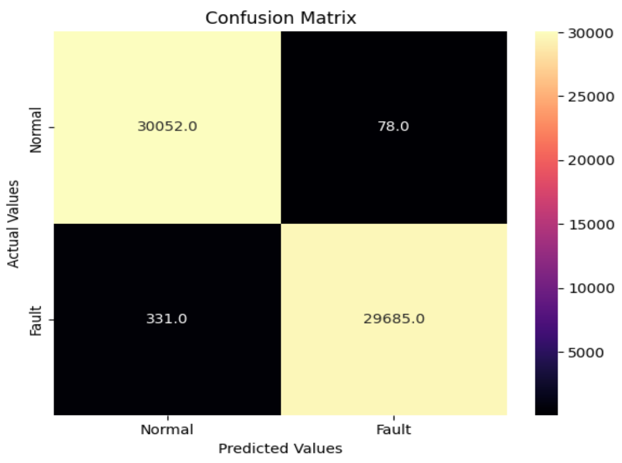

Figure 5: Confusion Matrix of the LSTM Model

A matrix of perplexity assessments. The performance of the LSTM model is shown in Figure 6. For two classes, it displays the numbers of accurate and inaccurate forecasts. The matrix reveals a substantial number of TN (30052) and TP (29685), alongside a relatively small number of FP (78) and FN (331). Color intensity, from black to light yellow-orange, visually represents sample counts; lighter shades denote higher values, quantified by the side color bar, confirming strong performance.

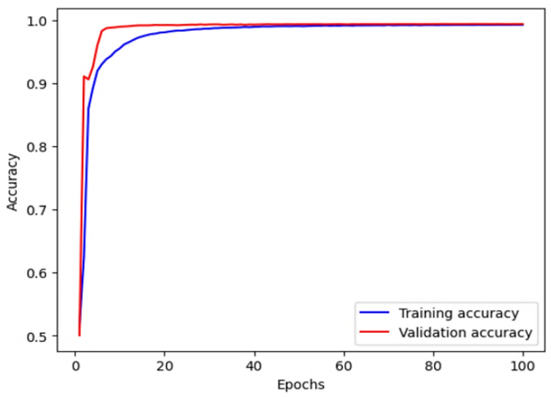

Figure 6 : Plot Accuracy Graph of the LSTM Model

An LSTM model's training and validation accuracy across 100 epochs is shown in Figure 6. The plot reveals a swift increase in both training (blue line) and validation (red line) accuracy during the initial epochs. Both accuracies reach and stabilize at a high level, indicating the model's effectiveness in learning and generalizing from the data used for optimizing Industrial IoT networks.

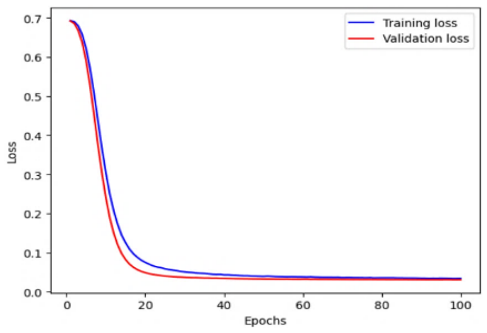

Figure 7 : Plot Loss Curve of the LSTM Model

Figure 7 plotted the loss of a trained LSTM model over 100 epochs. In the plot, training (blue line) and validation (red line) loss decrease rapidly in the first epochs, proving effective learning. It is found that both losses converge and plateau at a low value, indicating that the model learned well without overfitting the specific dataset used to optimize Industrial IoT networks.   Table III: Accuracy comparison of ml Models for IIot Fault Detection

A. Comparison and Discussion

This section compares ML models for fault prediction in IIoT networks. The FuzHD++ model had an accuracy of 92% (see Table III) and was followed by MLP at 96.6%. With 99% accuracy, the RF model improved the performance. Furthermore, the LSTM model presented the best result with accuracy of 99.33%, showing the best performance in modeling temporal dependencies. These results highlight the effectiveness and use of LSTM for the reliable fault detection in IIoT environments.

Table 3 : Accuracy Comparison of Ml Models for IIOT Fault Detection

Model	Accuracy
Random Forest (RF)[22]	99
Multilayer Perceptron (MLP)[23]	96.6
FuzHD++[24]	92
Long Short-Term Memory	99.33

The approach suggested works well in predicting faults early on in Industrial IoT networks, achieving an accuracy of 99.33% with an LSTM model. This approach performs better than traditional techniques because it reveals temporal features in data and helps clean, design, normalize and deal with imbalanced classes using SMOTE. It shows good generalization abilities and minimal overfitting which support dependable fault detection, the best network results and cut IIoT downtime.

V. CONCLUSION AND FUTURE SCOPE

A robust and secure IoT system is heavily dependent on accurate and reliable sensor data. Despite the usefulness of resource-constrained sensors, variables such as electrical noise and ambient conditions frequently lead to inaccurate and erroneous readings. Among the numerous areas they may impact is a higher danger to the safety of important systems. The primary objective of this study was to use predictive methods, such as the LSTM model, to identify IIoT issues. The model achieved more than 99% in recall, F1-score, precision, and accuracy, with an accuracy of 99.33%. Visual evaluations, such as accuracy/loss curves and a confusion matrix, provided additional evidence of the model's efficacy by confirming its excellent learning capability, generalisability, and low overfitting. Although these findings show promise, there are certain limitations to the study. Sensor noise, hardware failures, and changing network conditions are examples of real-world industrial circumstances that would have been absent from the experimental setup due to its reliance on a simulated environment. Despite its high accuracy, the proposed LSTM-based framework has limitations such as sensitivity to hyperparameter tuning and higher computational requirements for training on large-scale IIoT data. Additionally, the model's performance may vary with different sensor environments or unseen fault types. Future work will focus on incorporating real-time adaptive learning, exploring lightweight deep learning models for edge deployment, and extending the framework to multi-class fault classification across diverse IIoT applications. Deploying AI models in real IIoT environments faces key challenges such as limited computational resources on edge devices, the need for low-latency real-time processing, and variability in sensor data. Ensuring data privacy, integrating with legacy industrial systems, and maintaining model performance across diverse conditions further complicate deployment. These issues require efficient model optimization, robust system integration, and secure data handling strategies. Future work will focus on optimizing deep learning models like LSTM for edge deployment through model compression and quantization techniques to reduce resource usage. Additionally, implementing real-time adaptive learning will help models adjust to evolving sensor data and fault patterns. Expanding the framework to support multi-class fault detection, integrating with industrial protocols for seamless deployment, and incorporating federated learning for privacy-preserving analytics are also key directions. These advancements aim to enhance scalability, responsiveness, and security in diverse IIoT environments. The goal of future research is to use various, real-world IIoT datasets to deploy the model in industrial settings where it may be used in real-time. Important improvements include using adaptive learning to address concept drift, expanding to multi-class fault classification, and using edge computing to detect faults in real-time and at scale.

VI. REFERENCES

[1] N. Patel, "Sustainable Smart Cities : Leveraging IoT and Data Analytics for Energy Efficiency and Urban Development," J. Emerg. Technol. Innov. Res., vol. 8, no. 3, 2021.

[2] S. Pandya, "Predictive Analytics in Smart Grids : Leveraging Machine Learning for Renewable Energy Sources," Int. J. Curr. Eng. Technol., vol. 11, no. 6, pp. 677–683, 2021, doi: 10.14741/ijcet/v.11.6.12.

[3] Q. Zhou, T. Zhao, X. Chen, Y. Zhong, and H. Luo, "A Fault-Tolerant Transmission Scheme in SDN-Based Industrial IoT (IIoT) over Fiber-Wireless Networks," Entropy, vol. 24, no. 2, 2022, doi: 10.3390/e24020157.

[4] P. Pathak, A. Shrivastava, and S. Gupta, "A survey on various security issues in delay tolerant networks," J Adv Shell Program., vol. 2, no. 2, pp. 12–18, 2015.

[5] Z. Li, F. Fei, and G. Zhang, "Edge-to-Cloud IIoT for Condition Monitoring in Manufacturing Systems with Ubiquitous Smart Sensors," Sensors, vol. 22, no. 15, Aug. 2022, doi: 10.3390/s22155901.

[6] Y. Liu, Y. Yang, X. Lv, and L. Wang, "A Self-Learning Sensor Fault Detection Framework for Industry Monitoring IoT," Math. Probl. Eng., vol. 12, 2013, doi: 10.1155/2013/712028.

[7] C. Kan, H. Yang, and S. Kumara, "Parallel computing and network analytics for fast Industrial Internet-of-Things (IIoT) machine information processing and condition monitoring," J. Manuf. Syst., vol. 46, pp. 282–293, 2018.

[8] H. S. Chandu, "A Survey of Memory Controller Architectures: Design Trends and Performance Trade-offs," Int. J. Res. Anal. Rev., vol. 9, no. 4, pp. 930–936, 2022.

[9] A. Polleri, R. Kumar, M. M. Bron, G. Chen, S. Agrawal, and R. S. Buchheim, "Identifying a Classification Hierarchy Using a Trained Machine Learning Pipeline," U.S. Patent Application No. 17/303,918, 2022

[10] W. Yu, T. Dillon, F. Mostafa, W. Rahayu, and Y. Liu, "A global manufacturing big data ecosystem for fault detection in predictive maintenance," IEEE Trans. Ind. Informatics, 2020, doi: 10.1109/TII.2019.2915846.

[11] J. Praveenchandar et al., "IoT-Based Harmful Toxic Gases Monitoring and Fault Detection on the Sensor Dataset Using Deep Learning Techniques," Sci. Program., vol. 2022, 2022, doi: 10.1155/2022/7516328.

[12] S. Kumar et al., "A Low-Cost Multi-Sensor Data Acquisition System for Fault Detection in Fused Deposition Modelling," Sensors, vol. 22, no. 2, 2022, doi: 10.3390/s22020517.

[13] M. Hojabri, S. Kellerhals, G. Upadhyay, and B. Bowler, "IoT-Based PV Array Fault Detection and Classification Using Embedded Supervised Learning Methods," Energies, vol. 15, no. 6, 2022, doi: 10.3390/en15062097.

[14] T.-B. Dang, D.-T. Le, T.-D. Nguyen, M. Kim, and H. Choo, "Monotone Split and Conquer for Anomaly Detection in IoT Sensory Data," IEEE Internet Things J., vol. 8, no. 20, pp. 15468–15485, 2021, doi: 10.1109/JIOT.2021.3073705.

[15] S. Marathe, A. Nambi, M. Swaminathan, and R. Sutaria, "CurrentSense: A Novel Approach for Fault and Drift Detection in Environmental IoT Sensors," in Proceedings of the International Conference on Internet-of-Things Design and Implementation, New York, NY, USA: ACM, May 2021, pp. 93–105. doi: 10.1145/3450268.3453535.

[16] S. A. Viktoros, M. K. Michael, and M. M. Polycarpou, "Compact Fault Dictionaries for Efficient Sensor Fault Diagnosis in IoT-enabled CPSs," in 2020 IEEE International Conference on Smart Internet of Things (SmartIoT), 2020, pp. 236–243. doi: 10.1109/SmartIoT49966.2020.00042.

[17] A. Gaddam and V. Govender, "Detecting Sensor Faults and Outliers in Industrial Internet of Things," in International Conference on Sensing Technology, 2022, pp. 183–200.

[18] S. U. Jan, Y.-D. Lee, J. Shin, and I. Koo, "Sensor fault classification based on support vector machine and statistical time-domain features," IEEE Access, vol. 5, pp. 8682–8690, 2017.

[19] R. Tarafdar and Y. Han, "Finding Majority for Integer Elements," J. Comput. Sci. Coll., vol. 33, no. 5, pp. 187–191, 2018.

[20] L.-S. Lin, Z.-Y. Chen, Y. Wang, and L.-W. Jiang, "Improving Anomaly Detection in IoT-Based Solar Energy System Using SMOTE-PSO and SVM Model," in Frontiers in Artificial Intelligence and Applications, vol. 360, 2022, pp. 123–131. doi: 10.3233/FAIA220434.

[21] M. Vakili, M. Ghamsari, and M. Rezaei, "Performance Analysis and Comparison of Machine and Deep Learning Algorithms for IoT Data Classification," 2020.

[22] M. Safaei, M. Driss, W. Boulila, E. A. Sundararajan, and M. Safaei, "Global outliers detection in wireless sensor networks: A novel approach integrating time-series analysis, entropy, and random forest-based classification.," Softw. - Pract. Exp., 2022, doi: 10.1002/spe.3020.

[23] M. A. P. Putra, D. S. Kim, and J. M. Lee, "DB-BiLSTM: Euclidean Distance-Based Sensor Data Prediction for IoT Applications," Int. Conf. ICT Converg., vol. 2021-Octob, no. December, pp. 814–817, 2021, doi: 10.1109/ICTC52510.2021.9620877.

[24] N. Berjab, H. H. Le, and H. Yokota, "Recovering Missing Data via Top-k Repeated Patterns for Fuzzy-Based Abnormal Node Detection in Sensor Networks," IEEE Access, vol. 10, pp. 61046–61064, 2022, doi: 10.1109/ACCESS.2022.3181742.