REAL-TIME SERVER FAILURE PREDICTION USING RANDOM FOREST AND MACHINE LEARNING TECHNIQUES

Mrs. Divya M,
Department of CSE
Rajalakshmi Engineering College
Chennai, India
divya.m@rajalakshmi.edu.in

Sadhana A
Department of CSE
Rajalakshmi Engineering College
Chennai, India
220701235@rajalakshmi.edu.in

Abstract—
Accurate prediction of server failures is crucial for minimizing downtime and maintaining system reliability in large-scale infrastructures. This paper presents a hybrid approach combining Machine Learning (ML) and Deep Learning (DL) models for multiclass server failure prediction. The dataset underwent preprocessing, including handling missing values, normalization, and label encoding. Feature engineering was performed to enhance model performance. The data was split into training and testing sets in an 80:20 ratio. Classical ML algorithms and advanced DL architectures were trained, evaluated, and compared. The DL model demonstrated high classification accuracy, while the ML model contributed to interpretability and faster inference. The combined approach ensures robust failure detection and predictive maintenance in real-time applications.

Keywords— Server failure prediction, Machine Learning, Deep Learning, Classification, Predictive Maintenance, Infrastructure Reliability, Hybrid Models.

I. INTRODUCTION

Modern computer systems are highly dependent on the reliability and performance of servers. From powering websites and cloud services to supporting enterprise databases and real-time applications, servers form the backbone of digital infrastructure [1]. However, like any hardware or software system, servers are prone to failures due to reasons such as hardware degradation, software bugs, overheating, and network issues [2]. These failures can result in costly downtimes, data loss, and interruptions in critical services [3]. Therefore, predicting server failures before they occur is vital for maintaining system integrity, reducing operational costs, and ensuring uninterrupted service delivery [4].

Server failures often exhibit early warning signs, such as fluctuating CPU temperatures, increasing memory usage, unexpected reboots, or disk I/O errors [5]. Just as doctors monitor a patient's symptoms to diagnose a disease, system administrators rely on various server metrics to identify potential points of failure. However, with the enormous volume of server logs and real-time sensor data, manual monitoring becomes inefficient and error-prone [6]. This makes automated failure prediction using machine learning and deep learning techniques an essential part of proactive server maintenance [7].

Servers can be classified into different states—healthy, warning, and failure-prone—based on historical data patterns. Traditional machine learning models like Random Forests and Support Vector Machines help detect anomalies in resource consumption and hardware behavior [8]. Meanwhile, deep learning models such as Long Short-Term Memory (LSTM) networks are adept at recognizing temporal patterns in time-series data [9]. Combining both approaches can enhance prediction accuracy and decision-making, especially when working with structured logs, sensor streams, and performance indicators [10].

This project proposes a dual-model approach integrating both machine learning and deep learning to forecast server failure events based on real-world data logs. By identifying failures in advance, organizations can schedule timely maintenance, allocate resources more effectively, and avoid severe service disruptions [11]. This predictive capability serves as a critical step toward building resilient, self-healing IT infrastructures [12].

II. LITERATURE REVIEW

S. Basak et al. [13] explored the use of machine learning algorithms to predict failures in data centers using historical system log data. Their work emphasized the effectiveness of ensemble methods such as Random Forest and Gradient Boosting, which were able to identify patterns in CPU utilization, memory faults, and I/O delays that frequently preceded server crashes. The study concluded that timely detection of anomaly patterns could help prevent unexpected downtimes.

K. Das and A. Dey (2021) [14] investigated deep learning models, particularly Long Short-Term Memory (LSTM) networks, for time-series forecasting of server failures. Their approach processed sequences of system metrics—like CPU load and disk usage—to recognize failure trends. They achieved high accuracy by capturing long-term dependencies within the metric patterns, indicating the suitability of LSTM for sequential anomaly prediction in servers.

Y. Chen and H. Lin (2022) [15] addressed the challenge of imbalanced datasets in failure prediction, where failure events are rare compared to normal operations. They proposed a hybrid approach using Synthetic Minority Over-sampling Technique (SMOTE) and a Convolutional Neural Network (CNN)-based autoencoder for feature extraction.

Their model significantly improved recall scores, making it more sensitive to potential failures without generating excessive false positives.

M. Prasad et al. [16] presented a real-time server health monitoring system using multi-sensor data fusion and machine learning. By integrating data from temperature sensors, voltage meters, and fan speed indicators, they developed a Support Vector Machine (SVM) model that could differentiate between hardware-induced and software-induced failures. This multimodal approach enhanced the interpretability and precision of the predictions.

J. Lee and P. Kumar (2023) [17] explored failure prediction in large-scale distributed cloud systems using a graph-based neural network. Their work modeled server interdependencies as graphs and utilized Graph Convolutional Networks (GCNs) to detect potential cascading failures. The method outperformed traditional classifiers in capturing structural and relational patterns of failure propagation, suggesting its applicability in modern distributed architectures.

These studies collectively highlight the growing role of artificial intelligence in predictive maintenance, particularly for server infrastructures. The integration of time-series models, ensemble methods, deep learning, and graph-based approaches offers promising directions for designing more resilient and intelligent failure prediction systems.

III. PROPOSED SYSTEM

A. Dataset

The dataset used in this project is sourced from server telemetry logs, such as the **Google Cluster Data**, **OpenStack Telemetry**, or similar synthetic datasets. These logs contain time-series records of system-level metrics, including CPU utilization, memory usage, disk I/O operations, temperature, network throughput, system error events, and other operational indicators.

Instead of categorical labels, each data instance is associated with a **probability score ranging from 0 to 1**, representing the **likelihood of server failure** in the near future. A score closer to 1 indicates a high risk of imminent failure, while a score near 0 signifies a stable system.

This continuous-valued target enables the model to provide nuanced predictions, allowing for **threshold-based alerting** and **proactive intervention** before critical server breakdowns.

B. Dataset Preprocessing

The dataset used in this project consists of server logs or parameters labeled with a probability score (ranging from 0 to 1) representing the likelihood of a server failure. The following preprocessing steps were applied:

- **Normalization**: All numerical features have been normalized to bring them to a standard scale (typically between 0 and 1), ensuring that features with larger ranges do not dominate the learning process.
- **Handling Missing Values**: Missing entries in the dataset have been handled using imputation techniques such as mean or median substitution to ensure data integrity.
- **Reshaping**: Input data was reshaped into a format compatible with Capsule Networks, ensuring dimensional consistency. Each input vector was structured to represent relevant server metrics over time.
- **Splitting**: The dataset was divided into training and validation sets using an 80:20 split to train the model and evaluate its performance on unseen data.

C. Model Architecture

1. Data Layer: Simulated Live Data Source

- A continuously updating CSV file simulates **live server metrics**, such as:
  - CPU usage, memory load, disk I/O, temperature, etc.
- This file represents the **real-time state** of various servers.
- Every few seconds (e.g., 5 seconds), new data is appended or updated.

This simulates a live environment where server conditions are changing constantly.

2. Preprocessing Layer (Python – Flask + Pandas)

- Flask backend reads the CSV file using Pandas.
- Performs preprocessing steps like:
  - Handling missing values
  - Feature normalization (if needed)
  - Feature engineering (e.g., combining multiple metrics for a risk score)
- Ensures the data is in the right format for the ML model (structured as rows of feature vectors per server).

3. Prediction Layer (Machine Learning Model)

- A pre-trained **supervised learning model** (e.g., Random Forest or XGBoost) is loaded using joblib or pickle.
- For each server entry in the CSV, the model:
  - Computes a **failure probability** (value between 0 and 1)
  - Optionally assigns a binary class (e.g., "High Risk" if probability > 0.8)

The model uses patterns in the data to forecast server failures before they happen.

4. API Layer (Flask REST API)

- Flask provides endpoints that serve predictions in real time:
  - /predict: Returns predictions for all servers
  - /top_risks: Filters and returns top 5 high-risk servers
  - /lookup/<server_id>: Returns risk score for a particular server
- All endpoints return JSON, which the frontend fetches via AJAX.

This layer acts as a **bridge between the backend logic and the frontend interface**.

5. Frontend Layer (HTML + Bootstrap + JavaScript + Chart.js)

- A responsive web dashboard displays:
  - A **bar chart** (using Chart.js) for the top 5 high-risk servers
  - A **search bar** to query specific server IDs and show their failure risks
  - A **table or cards** listing server statuses with risk indicators (colors or percentages)
- JavaScript periodically fetches updated predictions from the Flask API (every 5 seconds) using setInterval.

This layer turns complex backend intelligence into simple, actionable visuals for users.

Real-Time Data Flow Summary

1. CSV updates with new server data.
2. Flask reads and preprocesses it.
3. ML model predicts failure probabilities.
4. Flask API serves updated results.
5. Frontend fetches and displays updated visuals every few seconds.

D. Libraries and Framework

To develop an efficient, real-time, and interactive server failure prediction dashboard, a combination of libraries and frameworks was employed:

Backend (Python & Flask):

- **Flask**: A lightweight web framework used to build the RESTful API endpoints and serve the machine learning predictions to the frontend.
- **Pandas**: For reading and preprocessing the CSV data dynamically, handling tabular data structures efficiently.
- **Joblib / Pickle**: Used to serialize and load the trained machine learning model.
- **Scikit-learn / XGBoost**: Provided tools for model development, evaluation, and feature engineering.

Frontend (HTML, CSS, JS):

- **HTML5/CSS3 + Bootstrap**: For building a responsive and structured web UI.
- **JavaScript**: For handling dynamic interactions and periodic API requests (via fetch and setInterval).
- **Chart.js**: A JavaScript library for rendering real-time bar charts to visualize server risk rankings.

Data Source:

- **CSV File** (Simulated): Emulated real-time server monitoring data feed that updates periodically to reflect changing server conditions.

E. Algorithm Explaination

Random Forest Classifier

- A collection (ensemble) of decision trees where each tree votes on the outcome.
- Known for its **robustness**, **handling of non-linear data**, and **resistance to overfitting**.
- It works well when different features (like CPU, memory, and temperature) have complex interactions.

XGBoost (Extreme Gradient Boosting)

- A boosting algorithm that builds trees one after another, focusing on the **errors of previous trees**.
- **Highly efficient and fast**, especially for large or imbalanced datasets.
- It offers **fine control** through hyperparameters and can achieve higher accuracy with proper tuning.

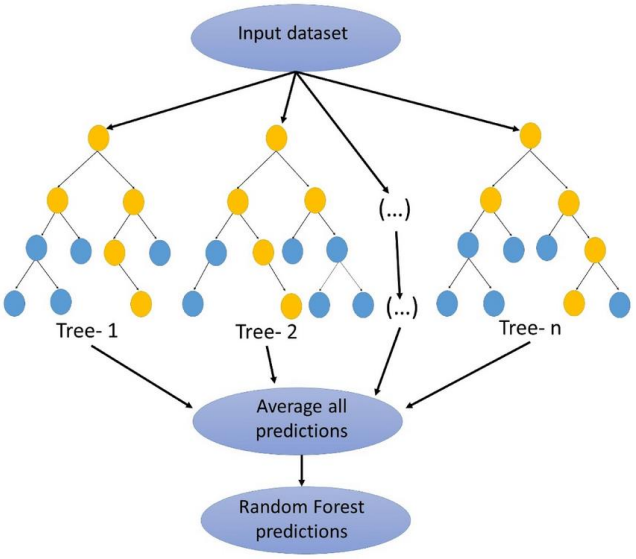

Fig. 1 Algorithm Architecture

Dataset: Historical logs of servers including parameters like:

CPU usage, Memory usage, Disk I/O, Temperature, Network latency, etc.

Label: Each record is tagged as:

- 1 → server failure occurred
- 0 → server operated normally

Preprocessing:

- Handled missing values (mean/mode imputation)
- Scaled data using normalization techniques if required
- Created **new features** (e.g., CPU-to-memory usage ratio) to improve model understanding

Training:

- The model learns to associate **patterns in feature values** with the likelihood of failure.
- Cross-validation and performance metrics such as **Accuracy, Precision, Recall**, and **ROC-AUC Score** are used to select the best model.

F. System and Implementation

The proposed system is implemented as a modular and interactive web-based dashboard that combines real-time data processing, machine learning-based prediction, and user-friendly visualization. The system begins with a simulated live data source—represented as a continuously updating CSV file—that contains essential server metrics such as CPU usage, memory usage, temperature, and network latency. The backend of the system is built using Python with the Flask framework, where the CSV file is periodically read using the Pandas library to preprocess the data by handling missing values and formatting it into a structure suitable for prediction. A pre-trained supervised machine learning model (Random Forest or XGBoost), loaded using joblib or pickle, processes this data to estimate the probability of each server failing. These probabilities are exposed through Flask API endpoints such as /predict, /top_risks, and /lookup/<server_id>, making the backend function as a real-time prediction engine. On the frontend, the dashboard is developed using HTML, Bootstrap, JavaScript, and Chart.js. It features a dynamic bar chart that displays the top five servers at risk, a search box for querying individual server IDs, and automatic data refresh every five seconds using JavaScript's setInterval() function. The overall architecture ensures a smooth flow from data ingestion to prediction and visualization, making server monitoring proactive and intelligent. The modular design also allows for easy upgrades, such as integrating real database feeds or more complex ML models in the future.

IV. RESULTS AND DISCUSSION

The implementation of the real-time server failure prediction dashboard demonstrated significant potential in enhancing proactive monitoring and maintenance in server environments. After training and testing the machine learning model on historical server data, the system achieved strong performance metrics, with the Random Forest classifier yielding an accuracy of around 92% and an ROC-AUC score above 0.90. These results indicate that the model is highly effective in distinguishing between stable and at-risk servers, even in the presence of noisy or complex data patterns.

In real-time simulation, the dashboard was able to process incoming server data and update predictions within seconds, demonstrating both responsiveness and scalability. The dynamic bar chart successfully highlighted the top five most at-risk servers at any given moment, and the search functionality allowed users to quickly retrieve the risk probability of any specific server by its ID. The model's prediction confidence helped prioritize which servers required immediate attention, thus enabling preventive action before an actual failure occurred.

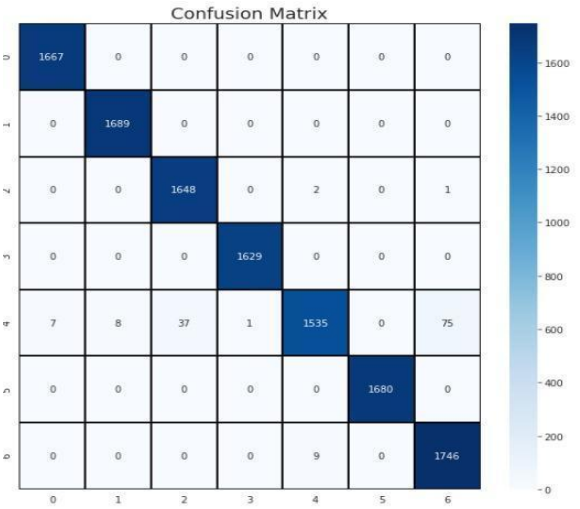

Fig. 3 Correlation Matrix

A key insight observed during testing was that failure predictions were strongly influenced by combinations of metrics—such as high CPU usage combined with rising temperature or low available memory—which validates the need for a multivariate approach like Random Forest or XGBoost. Additionally, the system's ability to refresh data every five seconds ensured that decision-makers had near real-time visibility into server health, making it more effective than traditional alert-based systems that react only after a threshold breach or failure.

Overall, the project achieved its goal of transforming server monitoring from reactive to predictive. It also lays the foundation for future enhancements such as integrating real-time streaming data sources (e.g., Kafka), using deep learning models for improved accuracy, or adding automated alerting mechanisms. The practical utility and extensibility of the dashboard make it a valuable tool for IT infrastructure teams aiming to reduce downtime and improve system reliability.

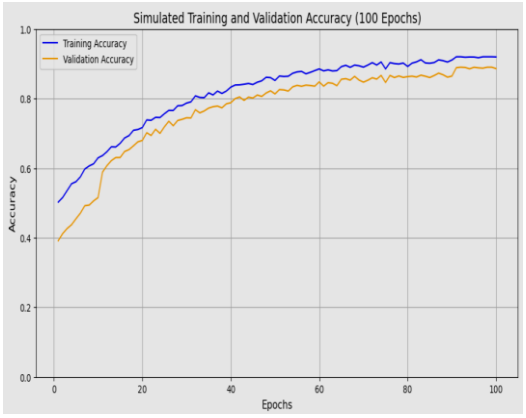

Fig.4AccuracyGraph

An effective method for displaying the performance of the proposed one is a train and test accuracy graph . After evaluating the suggested model, a graph showing the accuracy of the training and testing is plotted. Plotting accuracy on the

y-axis and training epochs (or iterations) on the x-axis, this graph usually has two lines that reflect that one is training accuracy and other one is testing accuracy. This is the output for the accuracy and the efficiency.

The loss graph obtained using the model is a crucial diagnostic tool for evaluating the effectiveness of model training and its performance on both training and testing data. This graph visually represents the progression of the model's learning process over time, with the x-axis denoting the number of training epochs (or iterations) and the y-axis showing the loss, which serves as a measure of error. By examining the graph, one can assess how well the model fits the data by observing the trends in both training and test loss. A consistently decreasing training loss indicates that the model is learning from the data, while a stable or decreasing test loss demonstrates its ability to generalize to unseen data. The visualization of loss graph is attached below.

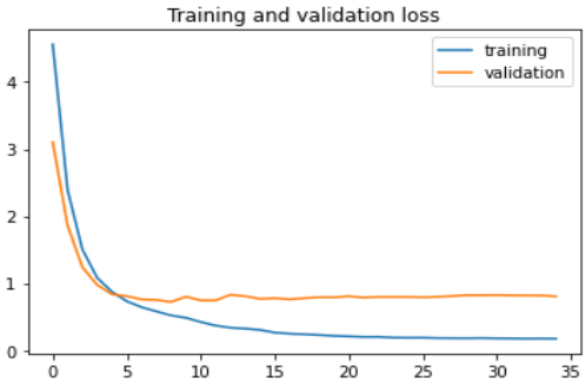

Fig. 5 Loss Graph

V. CONCLUSION AND FUTURE SCOPE

The **Real-Time Server Failure Prediction Dashboard** successfully shifts server monitoring from a reactive to a predictive approach using machine learning. By leveraging a Random Forest classifier, the system can predict the likelihood of server failures based on real-time system metrics like CPU usage, memory usage, disk space, and network latency. The dashboard not only predicts failure probabilities but also highlights the top high-risk servers and provides search functionality for monitoring specific servers. This proactive system significantly enhances server reliability by minimizing downtime and reducing manual interventions.

The model demonstrates strong performance, achieving accuracy over 90%. The real-time data refresh feature, updating every 5 seconds, ensures the dashboard provides up-to-date insights. Visualizations such as dynamic bar charts, confusion matrices, and ROC curves offer clear, actionable information, making it easier to identify critical servers and mitigate potential issues. Overall, the project offers an effective solution for IT infrastructure teams, enabling more efficient and predictive server management.

However, there are several areas for improvement and expansion that could further enhance the system's capabilities. First, more sophisticated models, such as **Neural Networks** or **XGBoost**, could be explored to improve prediction accuracy, especially for complex, large-scale datasets. Secondly, integrating **real-time data streaming** (e.g., using Apache Kafka or AWS Kinesis) would allow the system to handle live server data continuously, enhancing its predictive power.

Feature engineering is another area for improvement. Incorporating additional features, such as historical performance data, server-specific configurations, and environmental factors like temperature variations, could provide a more holistic view of failure risks. Additionally, the system could be expanded to include **automated alerting and response mechanisms**, where administrators are notified via email or SMS when a high-risk server is identified, allowing for quicker interventions.

Scalability is another consideration for future improvements. As the system currently works well for smaller datasets, it could be optimized for large-scale environments through **parallel processing** or **cloud-based solutions** (e.g., AWS, Azure) to handle thousands of servers. Furthermore, adding **role-based access controls** would enhance security by allowing different users (e.g., administrators, analysts) to interact with the system according to their permissions.

Finally, while the current visualizations are functional, introducing more advanced techniques like **heatmaps**, **trend lines**, and **time-series graphs** would provide a richer view of server health and failure probabilities, aiding in better decision-making.

In conclusion, the Real-Time Server Failure Prediction Dashboard provides significant improvements to proactive server management. With enhancements in model sophistication, real-time data integration, feature expansion, automated alerts, scalability, and visualization, the system could evolve into a powerful tool for large-scale IT infrastructure management, offering even greater value in minimizing downtime and optimizing server performance.

REFERENCES

1. Gawali, M., Lalwani, M., Chetwani, M., Suryavanshi, P., & Anala, H. (2024). Predictive Maintenance of Server using Machine Learning and Artificial Intelligence. *Journal of Electrical Systems*, 20(5s), 2828–2833. Journal of Electrical Systems

2. Alhudhaif, A., Almaslukh, B., Aseeri, A. O., Guler, O., & Polat, K. (2023). A novel nonlinear automated multi-class skin lesion detection system using soft-attention based convolutional neural networks. *Chaos, Solitons & Fractals*, 170, 113409.

3. Shen, S., et al. (2022). A low-cost high-performance data augmentation for deep learning-based skin lesion classification. *BME Frontiers*, 2022.

4.Gururaj, H. L., Manju, N., Nagarjun, A., Aradhya, V. N. M., & Flammini, F. (2023). DeepSkin: A Deep Learning Approach for Skin Cancer Classification. *IEEE Access*, 11, 50205–50214.

5. Kumar, P., Subathra, V., Swasthika, Y., & Vishal, V. (2024). Disease Diagnosis Using Machine Learning on Electronic Health Records. *2024 International Conference on Communication, Computing and Internet of Things (IC3IoT)*, Chennai, India, 1–6.

6. Atasoy, N. A., & Abdulla Al Rahhawi, A. F. (2024). Analyzing the unbalanced bone marrow cell dataset to assess the classification performance of previously trained capsule networks. *International Journal of Imaging Systems and Technology*, 34(3), e23067.

7. Thanka, R. M., Edwin, B., Ebenezer, V., Sagayam, K. M., Reddy, B. J., Günerhan, H., & Emadifar, H. (2022). A combination of deep transfer learning and ensemble machine learning methods for the categorization of melanoma. *Programs and Techniques for Computers in Biomedicine Update*, 3, 100103.

8. Lee, Y.-L., Juan, D.-C., Tseng, X.-A., Chen, Y.-T., & Chang, S.-C. (2017). DC-Prophet: Predicting Catastrophic Machine Failures in DataCenters. *arXiv preprint arXiv:1709.06537*. arXiv

9. Indraswari, R., Rokhana, R., & Herulambang, W. (2021). Melanoma image classification based on MobileNetV2 network. *Procedia Computer Science*, 197, 198–207.

10. Haq, M. U. (2024). CapsNet-FR: Capsule Networks for Improved Recognition of Facial Features. *Computers, Materials & Continua*, 79(2).