National College of Ireland

Early Warning System for Distributed File System Failures Using Temporal Log Analysis

MSc Research Project
Programme Name

Harshitha Reddy Bachupalli Kollan
Student ID: X23300531

School of Computing
National College of Ireland

Supervisor: Aaloka Anant

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Harshitha Reddy Bachupalli Kollan
Student ID:	X23300531
Programme:	MSc. Data Analytics (MSCDAD_JAN25I) **Year:** 2025- 2026.
Module:	MSc (Research) Practicum Part 2
Supervisor:	Aaloka Anant
Submission Due Date:	11-12-2025
Project Title:	Early Warning System for Distributed File System Failures using Temporal Log Analysis.
Word Count:	7094 **Page: 25 Count**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Harshitha Reddy Bachupalli Kollan

Date: 10-11-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Early Warning System for Distributed File System Failures Using Temporal Log Analysis

Harshitha Reddy Bachupalli Kollan
X23300531

Abstract

Distributed storage systems generate large volumes of operational logs, and most available analytical tools remain reactive and only detect the failures after they occur. This limitation restricts the ability to intervene in environments such as the Hadoop Distributed File System (HDFS), where block failures can adversely affect system availability and performance. This paper explores the possibility of using temporal trends on the logs of HDFS to predict failures before they become a reality. A temporal forecasting system was developed which converts raw logs into sequence forms and considers two opposite modelling methods: non-sequential baseline model (XGBoost) and sequential deep learning model (LSTM). This methodological design allows for determining whether modelling temporal dependencies offers quantifiable benefits in the early-warning forecasting. The results indicate that the temporal sequencing approach can effectively predict the occurrence of failure-prone behaviour than frequency-based baselines. The paper proves that proactive failure prediction based on the temporal log analysis is feasible and it represents a step toward better reliability management in distributed storage systems.

Keywords: Early-warning systems, temporal forecasting, distributed storage systems, HDFS sequence modelling, LSTM, XGBoost, proactive failure prediction, reliability management.

1 Introduction

Distributed storage systems such as the Hadoop Distributed File System (HDFS), generate large-scale of operational logs enabling system activity, warnings, and error patterns. These logs offer valuable insight into system behaviour, yet most operational analytics remain reactive and detect failures only after they manifest. Such limitations affect system availability and performance, as block failures can disrupt data processing and trigger costly recovery procedures.

Recent advances in log analytics have enabled automated anomaly detection; however, these approaches typically diagnose abnormal behaviour rather than forecast failures ahead of time. This creates a critical gap for large-scale distributed environments, where proactive warnings could support preventive actions and reduce operational overhead.

This study addresses this gap by examining whether temporal patterns in HDFS logs can provide early indicators of failure-prone behaviour. The research seeks to determine the extent to which temporal modelling particularly the capture of event sequences can enhance predictive performance beyond traditional non-sequential approaches.

Research Question: Can we predict distributed file system failures before they occur by analyzing temporal patterns in system logs, and if so, how much advance warning (lead time) can be achieved via the method?

Research Objectives

1. **To Establish a forecasting methodology of prediction of failures prior to occurrence with the help of sliding time window.**
2. **This approach Measure works on achievable lead time between predictions vs actual failures.**
3. **It also Evaluate predictive performances using operational metrics such as precision, recall, false alarm rate**
4. **It Compares the performance of temporal forecasting to reactive detection baselines.**

Contribution: The research demonstrates an early-warning pipeline that converts the unstructured HDFS logs into temporally ordered features and uses machine-learning models to predict the near-future block failures. A secondary contribution relates to a comparative study in architecture that shows the benefits of sequential modelling in order to predict failures that may arise beforehand. Together, these contributions indicate that the temporal log forecasting can be used to complement conventional monitoring tools and offer proactive reliability support to the HDFS Operations Team.

Report Structure: Section 2 uses a summary of related work in log anomaly detection and failure prediction. Our temporal forecasting methodology is in Section 3. Section 4 describes the system architecture and design. Section 5 recommends Python implementation, XGBoost, LSTM and HDFS logs implementation. Section 6 examines the outcomes, such as lead time analysis. Section 7 ends with conclusions and future research.

2 Related Work

Research on system reliability and log analytics has developed in a variety of methodological streams, such as log anomaly detection, time-series-based failure prediction, and machine-learning-based structured log feature classification. This section critically evaluates the main contributions in these categories, and the limitations of their work, and it places the necessity of a temporal early- warning system that supports the HDFS Operations Team's operations requirements.

2.1 Log-Based Anomaly Detection

Initial research on automated log analysis was mainly concerned with the detection of anomalies rather than prediction. An LSTM-based model proposed as DeepLog (He et al., 2016; Du et al., 2017) learns the regular course of action based on logs and identifies the deviations as an anomaly. Despite this approach having a good performance, it is inherently reactive in its design: it starts detecting unusual events as soon as they happen and gives no prior warning. LogAnomaly (Lu et al., 2018) is an extension of this concept as log-template embeddings are used alongside LSTM models, yet it is also concurrently oriented at anomaly detection and not prediction. These systems show that deep learning can learn temporal log dependencies but will not make predictions of failures before an anomaly, and thus cannot be used in proactive operational decision-making.

2.2 Non-Sequential Machine Learning for Logs

Traditional machine-learning models use log data in the form of feature vectors rather than ordered sequences. Meng et al. (2019) used XGBoost on event-frequency and demonstrated that boosting algorithms can be successfully used to categories log patterns. Such models are interpretable and cost-efficient to compute, but do not take into consideration that the patterns of failure might develop over time, making it a major weakness in scenarios where failure patterns might occur progressively.

Count-based techniques are suitable for detecting abnormal distribution, but cannot change the structure of log events, which may be used before failures.

2.3 Time-Series Failure Prediction

Forecasting approaches have been examined in various domains. Zhang et al. (2021) predicted disk failures based on the SMART sensor measurements, achieving strong lead time, while ProFIT (Chen et al., 2023) was able to predict failures of microservices based on distributed tracing data. These examples demonstrate the practicability of the forecasting of failures, although the data used in them (numerical sensors or tracing signals) cannot be considered very similar to unstructured HDFS logs.

These techniques validates the usefulness of short-term forecasting, but cannot be applied directly to HDFS log data, which consists of textual and irregular and highly diverse events.

2.4 Overview Comparison of Prior Work.

In order to synthesis the state of the art, Table 1 compares important peculiarities of representative approaches.

Table 1: Comparison of Existing Approaches in Log Analysis and Failure Prediction

Approach	Method	Limitation
DeepLog (2016/2017)	LSTM anomaly detection	Reactive; no prediction
LogAnomaly (2018)	CNN + LSTM	No lead time
XGBoost (2019)	Event-count classifier	Ignores sequence order
Salfner et al. (2010)	Survey of prediction methods	Not log-specific
SMART Disk (2021)	Sensor-based forecasting	Structured numeric data only
ProFIT (2023)	Trace-based	Requires
	prediction	instrumentation
LogHub (2020)	Log dataset resource	No Modelling approach

2.5 Critical Evaluation & Identified Gaps

There are three key trends in the literature:

1. Current tools used in log analytics are mainly reactive.

Even advance systems like DeepLog observe anomalies after they have arisen. They are unable to provide the lead time, which the HDFS Operations team requires to take pre-emptive action.

2. The modelling of sequences is useful but not been used in forecasting.

LSTM-based systems capture the temporal relationships, but have not been used to anticipate upcoming failures.

3. Research on Forecasting in any other domains cannot apply on HDFS logs.

Sensor-based and tracing-based predictors are based on structured numeric data, whereas HDFS logs are unstructured and event-driven.

2.6 Justification for the Present Research

The results collectively suggest that there is an evident gap that has not been addressed. No current method exists that conducts time prediction on HDFS logs to offer viable early-warning forecasts to work . This gap is the direct reason behind the research question: Is it possible to predict distributed file-system failures beforehand by examining patterns in HDFS logs over time and, what can predictions of this sort do to help the HDFS Operations Team take proactive operational decisions?
This gap is filled in this dissertation by:

- converting raw HDFS logs to sequences in time,
- comparing sequential vs. non-sequential models (LSTM vs. XGBoost), and assessing their effectiveness in generating early-warning predictions.

3 Research Methodology

The approach taken in the research is a temporal forecasting methodology that aims at finding out whether the patterns in the HDFS log sequences in the short-horizon can be used to predict block failures in advance. It is a process that is consistent with the conventional data-science methodology but is tailored to sequential log analysis, which starts with dataset analysis, conceptual pre-processing, and event distribution exploration. These actions alert the construction of time windows, the generation of sequence-based traits and aggregated ones and the choice of modelling techniques to use in comparative analysis. This methodological design is directly relevant to the research goal because it allows the study of whether or not the temporal dependencies in system logs have predictive information on the failures of the system in the offing.

3.1 Dataset Description

The dataset obtained through the HDFS sample in the Log Hub archives is used in this study, which consists of 11.17 million log records that had been produced during a duration of 38 hours with 575,061 block executions. Every log entry has a timestamp, a block identifier and the content of the message that is generated by NameNode and DataNode components. The logs indicate the normal operational activity and block failure occurrences. The scale, temporal hierarchy, and granularity of the dataset ensure that it is adequate to study the concept of early-warning prediction.

3.2 Data Preprocessing

Data preprocessing focused on preparing the raw HDFS logs for temporal analysis by ensuring that all records were structurally consistent and correctly aligned in time. This involved extracting the essential attributes from each log entry—such as timestamps, block identifiers, and event messages—and standardising their formats to allow accurate chronological ordering. Invalid or incomplete entries were removed to maintain data quality. Log records were then grouped by block identifier to preserve the behavioural history of each block, enabling the subsequent construction of temporal windows. These preprocessing steps ensured that the dataset was clean, time-consistent, and suitable for downstream template extraction, windowing, and predictive modelling.

3.3 Exploratory Data Analysis (EDA)

The purpose of conducting Exploratory Data Analysis (EDA) was to better understand how an HDFS log data set is structured as well as how the events contained within are distributed. When comparing distributions, there appears to be an extremely uneven concentration of log activity across different blocks within our dataset; while some blocks generated between 10 to 30 logs, many others generated significantly more logs due to repetitive recovery and replication behaviours.

Temporal concentrations of events also indicate an highly bursty behaviour in their occurrence rates. For instance, many logs were created by processes occurring cluster-wide, such as replication storms, task balancing operations or log failures, within highly concentrated time frames. These peaks in event volume show us that our models must learn to "smooth" and segment the data based on window sizes and time orders of behaviours, as well as allow for the learning of short-term dependencies of behaviours.

Information obtained through EDA influenced our methodology in several key ways including the following:

- Include frequent event templates only,
- Uses a 10-minute historical window,
- Determine temporal prediction horizons, and
- Utilize temporal models capable of learning behaviours of escalation (LSTM).

The two visualizations described here are from our EDA and represent the results of this analysis.

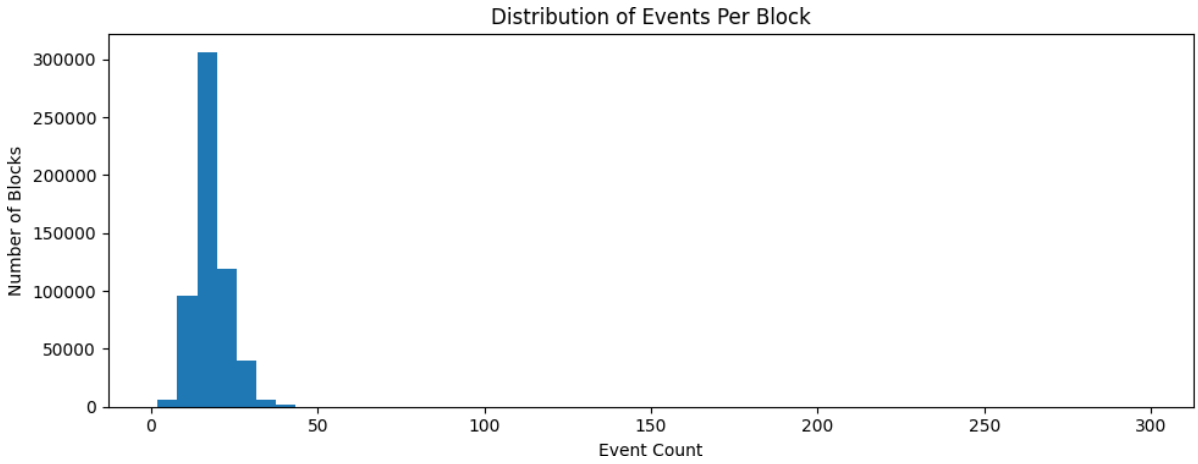


Figure 1: Distribution of Event per Block

This figure shows the skewed distribution of log events across HDFS blocks. Most blocks generate fewer than 30 events, while a long-tail minority generate over 100–300 events. This imbalance indicates that operational behaviour is not uniform across the cluster and supports the need for block-level temporal forecasting.

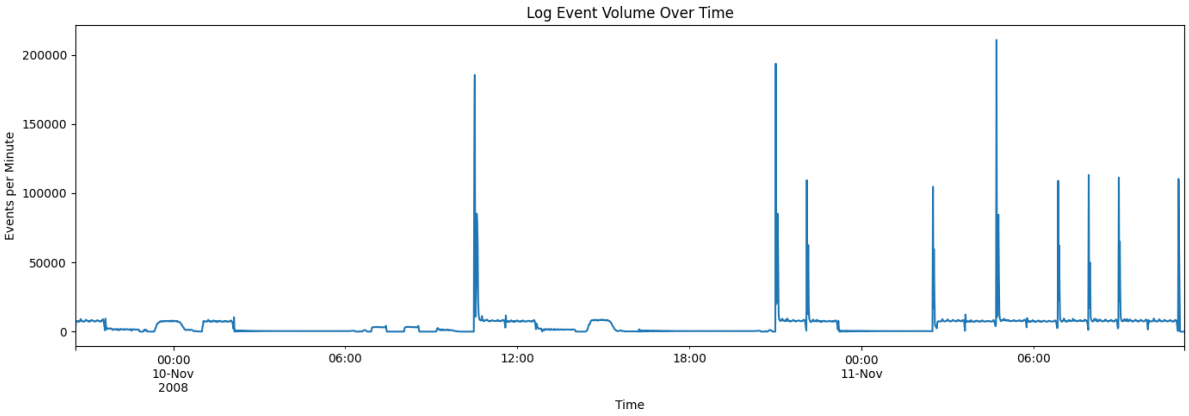


Figure 2: Log Event Volume Over Time

This plot illustrates the bursty and irregular nature of event generation in the HDFS cluster. Several extreme spikes occur, corresponding to high-load operations or failure-related processes. These fluctuations justify the use of sliding temporal windows and the need for models that can account for changing temporal dynamics.

3.4 Feature Engineering

All the windows were converted into numerical features. Event-frequency vectors were used to describe the count of each event type in the history window, both the non-sequential baseline (XGBoost). In the case of the sequential model (LSTM), vectors were rearranged into tensors of a single timestamp, but with temporal order. These images reflect the strength and structure of system activity - important signs of block instability.

3.5 Predictive Modelling

Two modelling approaches were employed to evaluate whether temporal information within HDFS logs holds predictive value for early failure detection. The first approach used XGBoost as a non-sequential baseline model, which represents each temporal window through aggregated event frequencies. This allows assessment of whether distributional shifts in log activity alone can signal emerging block instability. The second approach used a Long Short-Term Memory (LSTM) network, a sequential deep learning model capable of capturing ordered event transitions and short-term temporal dependencies. By comparing these two contrasting modelling strategies, the study examines whether predictive performance improves when models can learn behavioural patterns that evolve over time rather than relying solely on summary statistics. This comparative framework directly supports the investigation into the role of temporal sequencing in early-warning prediction.

3.6 Evaluation Methodology

The method evaluates the predictive quality as well as the practical usefulness of the developed early-warning models. Block failures are a minority of all the log windows, so precision-recall measures were chosen due to their informative value over class imbalance accuracy. Precision and recall measure the accuracy and completeness of failure forecasting, whereas the F1-score is a balance between the two. To measure ranking performance at varying thresholds and to compare the models without a fixed decision boundary, PR-AUC was added. As a measure of total discriminative ability, ROC-AUC was used as a complementary measure.

In addition to classification metrics, lead-time analysis was incorporated to determine the practical value of predictions in an operational context. Lead time measures the interval between the earliest warning issued by a model and the actual failure event, revealing whether predictions provide sufficient time for intervention. This combination of imbalance-metrics and operational timing assessments ensures a comprehensive evaluation of model behaviour and its relevance for proactive reliability management.

4 Design Specification

The system is modelled as a pipeline, which is modular, changing raw logs of HDFS into an early warning about block failures. All these elements have their own role, and the first part is the log ingestion and structuring, which is successfully supported by the creation of temporal sequences and numerical features. These processed inputs are input into a dual-model predictive engine that is made up of a non-sequential classifier and a sequential deep-learning model. The architecture ends with an evaluation layer that designs model outputs to be performance-assessed and operationally interpreted. The modular nature is that the separate components can be modified or expanded without interfering with the workflow, which is flexible and scalable.

4.1 System Architecture Diagram

This system comprises four major elements:

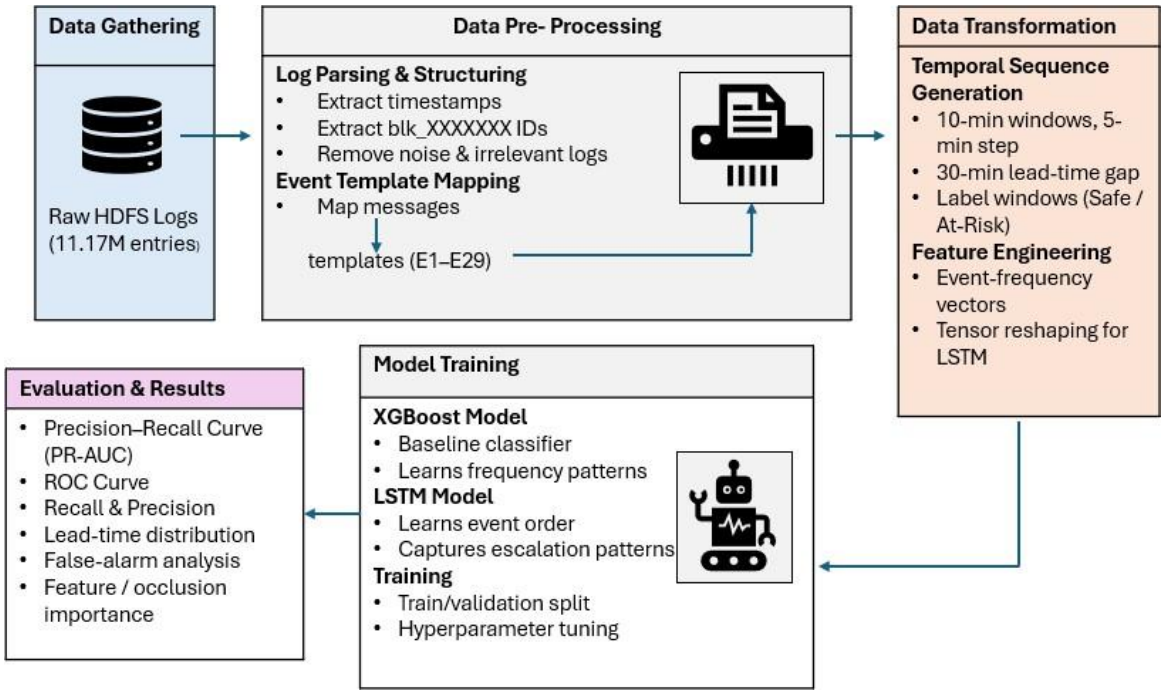


Figure 3: System Architecture of the Temporal Early-Warning Framework

The system architecture and data flow of the early-warning pipeline are shown in Figure 3. Raw HDFS logs are initially absorbed and organized and afterwards, each is examined and altered into event templates to create a standardized representation of system movement. These organised logs are subsequently transformed into time series and numerical tools that constitute inputs to the prediction engine, which comprises of the XGBoost baseline model and the LSTM sequential model. The risk scores obtained are forwarded to the evaluation layer, which evaluates the predictive performance and helps in interpretation. As shown in the diagram, the design itself has a modular structure, all the parts are doing a certain transformation and still contributing to the entire process of prediction

4.2 Log Ingestion and Storage

This component is a point of entry of the system, where raw HDFS logs are gathered and exposed in a uniform format to be used in subsequent processing. It hides the storage medium beneath so that logs can be stored either on a local file or on a distributed storage without changes in system behaviour.

4.3 Log Structuring and Event Template Mapping

This module standardises event templates, which are used to analyse the unstructured log messages, into standardised event templates. Every record of a log gets an event type with the necessary identifiers retained allowing further elements to understand the operational patterns with repeat patterns without using the raw text.

4.4 Temporal Sequence Generator

This component does not examine entire block histories, but each event sequence in a block is cut into time-bounded slices which reflect the recent behaviour of the system at various times. The result is a set of temporal inputs, the structure of which is predictable both

between blocks and encodes temporal ordering. This is entirely representative: the module will make sure that the time dimension is coded in such a manner that both sequential and non-sequential prediction models can be used.

4.5 Feature Engineering Module

The Feature Engineering Module ensures translation between the temporal representation of the log behaviour and the numerical representation needed by the prediction engine.
It takes the time samples generated by the sequence generator and obtains features representations of each. In the non-sequential model, this element creates vectorised summaries of every temporal segment; in the sequential model, it is used to prepare inputs in the temporal form that sequence-learning structures accept. The module is focuses in representation and ensures that the inputs to both models in the prediction engine are in the right shape and type.

4.6 Prediction Engine

The analytics of architecture is the Prediction Engine. It is a combination of two parallel and independent model sub-elements, which include XGBoost classification and LSTM sequential elements. The two elements receive processed inputs of the feature engineering module and generate a scalar risk score of the estimated likelihood of a block to fail within a specific prediction range. The engine provides a shared interface so that the outputs of both models can be compared or incorporated into ensemble strategies without modifying upstream or downstream components.

4.7 Evaluation and Reporting Layer

The final component of the architecture is the Evaluation and Reporting Layer. It takes the risk scores of both the XGBoost and LSTM modules and makes them ready to be analysed by the HDFS Operations Team. Architecturally, this layer may execute thresholding logic, produce summary reports and make metrics or warnings available through dashboards, alerts or log outputs. Likewise, it does not alter the behavior of the models but dictates the surfacing and consumption of their outputs.

5 Implementation

The implementation chapter presents a description of how the architectural design was implemented in code. Although Design Specification is only concerned about the structure and duties of each part, this chapter clarifies the way these parts have been created by applying particular tools, libraries, and programming functions. All implementation was done in Python with Jupyter Notebook and includes log parsing, temporal window construction, feature construction, model training, and evaluation as part of a self-executing workflow.

5.1 Development Environment

The system was developed in Python 3 within a dedicated virtual environment (MLVE) using Visual Studio Code as the primary IDE. This environment enabled isolated dependency management and efficient development workflows. Pandas and NumPy supported data manipulation, scikit-learn was used for vectorisation and metrics, and the predictive models were implemented using the XGBoost and TensorFlow/Keras libraries.”

5.2 Log Ingestion and Preprocessing Implementation

The log ingestion was done through reading the raw HDFS log file into a Pandas DataFrame. Each line was broken down into regular expressions, which would extract the timestamp, block identifier and log message. The timestamps were turned into datetime objects to make sorting and time arithmetic operations accurate. Any erroneous or malformed entries were filtered out to prevent downstream errors. After cleaning, the DataFrame was sorted chronologically, and the logs were clustered together by block identifier to create the foundation of per-block analysis.

5.3 Event Template Mapping Implementation

Pattern matching and dictionary look-up were used to implement template extraction and event mapping. The LogHub HDFS schema gives a mapping between the pattern of raw messages and event template identifiers. This mapping was loaded into memory and used with the cleaned log messages in a series of vectorised Pandas operations, meaning that it can convert millions of rows quickly. The free text message of each log entry was changed by an event ID, and the resulting sequences were stored in lists of (timestamp, event_id) pairs per block. Uninformative and rare templates that were found during exploratory analysis were not included in the final modelling vocabulary. This created the organised streams of events upon which the remainder of the pipeline is run.

5.4 Temporal Window Construction Implementation

The temporal windowing logic was as a function that walks through the events on the ordered events per block and divides them into fixed time intervals. With regards to a particular block, the implementation determines all those events that occur within a sliding history window by comparing their time stamps with a current reference time. The boundaries of the history window, the gap between lead-time and the prediction horizon were calculated using time arithmetic in Python. The function then examines the occurrence of a failure timestamp of such a block within the horizon and labels it accordingly. Timestamp indices were reused between windows instead of being recalculated. The function will give a list of the temporary window records of all the block IDs, the time the window starts, the set of events in the window and the label.

5.5 Feature Vector Construction

A DictVectorizer of Scikit-learn was used to perform feature construction. A dictionary of event counts was produced for every temporal window, with the keys being event IDs and the values being the count of occurrences in the window. The vectoriser converted these dictionaries to sparse feature vectors in the same column order. Every vector was added together into a NumPy array to compose the feature matrix of the XGBoost model. In the case of the LSTM model, the same vectors were rearranged into a 3-dimensional array of size (n samples, 1, n features) where the second dimension corresponded to a single timestep. This redefining enabled the existing window representation to be fed to a sequence model without redefining the data structure. Where needed, normalisation and type casting have been done to meet the needs of the model input.

5.6 XGBoost Model Implementation

XGBoost classifier was implemented using XGBoost library. Hyperparameters like the tree depth, learning rate and estimators, number were set using initial tuning experiments that were undertaken as part of the research. The scale pos weight parameter was adjusted depending on the ratio of negative to positive samples to manage the class imbalance. The

The model itself was trained on the feature matrix obtained with the help of the training set, and one of the validation sets was used to track convergence and prevent overfitting. The code of training stated PR-AUC to be the optimisation measure to match the imbalanced data. The model was serialised to disk after training to make this model reproducible, and prediction functions were actually coded to produce risk scores on the test set.

5.7 LSTM Model Implementation

The LSTM was built based on the Keras high-level API built over TensorFlow. The model architecture was determined with the help of the Sequential class, where LSTM layer was added, then one or several dense layers and a sigmoid layer, which is an output layer of binary classification. Code-based regularization techniques (dropout and L2 penalties) were used to improve overfitting. Compiling the model with the Adam optimiser and binary cross-entropy loss, and class weights were inputted in the training function to overcome the issue of class imbalance. Mini-batches were used to perform training, and validation performance was monitored using an early stopping callback to stop training once no significant improvement could be made. Like XGBoost, the model was stored on disk, and prediction functions were developed to produce risk scores of the sequence-formatted input.

5.8 Evaluation Pipeline Implementation

The measuring pipeline calculated every measure mentioned in the Results chapter. Once risk scores were obtained in each of the two models on the test set, threshold-based predictions on classification measures like precision, recall, F1-score, and confusion matrices were obtained. PR-AUC and ROC-AUC were directly calculated based on the distribution of the scores through the Scikit-learn metric functions. Lead time was used by appending the records of the prediction with the initial failure times and calculating the difference between the prediction time and the actual failure time in the cases that are correctly predicted.

5.9 Implementation Workflow Summary

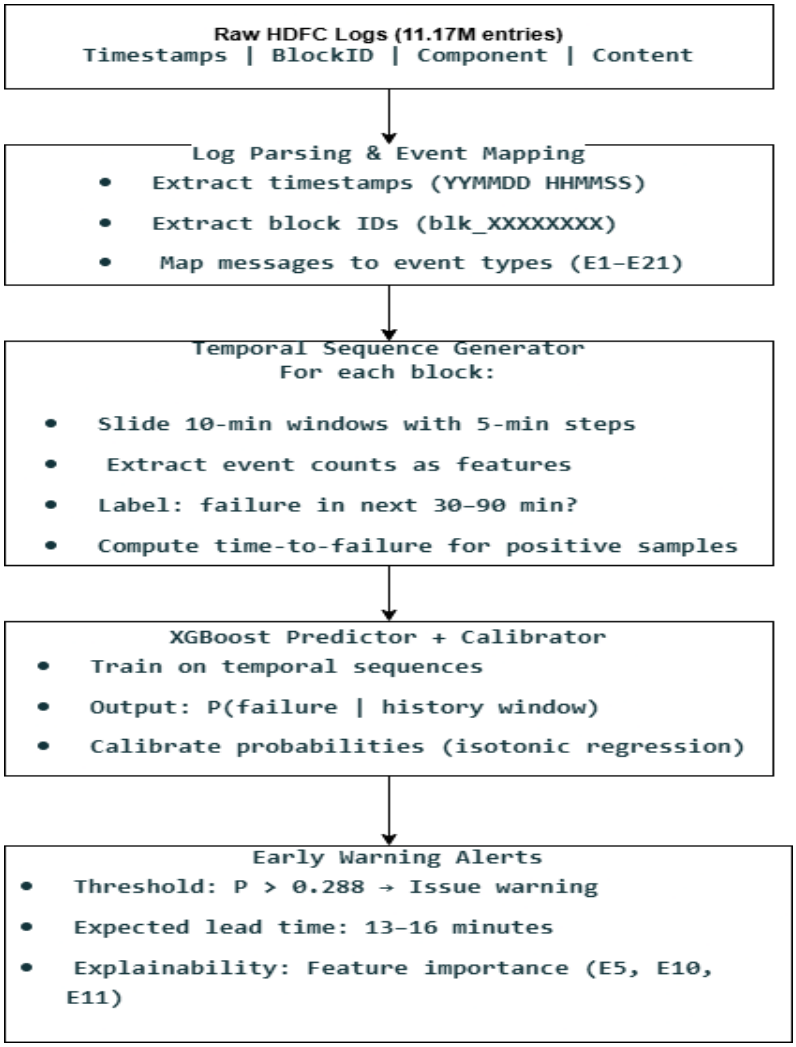


Figure 4: Detailed Implementation Workflow of the Early-Warning System

Figure 4. presents the complete implementation workflow, showing how raw HDFS logs are transformed through parsing, windowing, feature engineering, model inference, probability calibration, and final alert generation.

6 Results and Evaluation.

This section presents a detailed evaluation of the XGBoost and LSTM models for early-warning prediction.

6.1 Overview

This chapter shows the primary results of the early-warning framework and measures its performance according to the classification, ranking, interpretability, and operational metrics. Results are reported using both imbalance-sensitive metrics and operational metrics to assess not only predictive accuracy but also the practical usefulness of the system in issuing early warnings

6.2 Comparative Performance Summary

Table 2. provides a summary of the comparison that is conducted on the two models. It records their relative performance using the main evaluation measures, such as recall, precision, PR-AUC, ROC-AUC, median lead time and false-alarm behaviour. There is an evident tendency in the table: the LSTM sequential model is always better than the XGBoost, it has a greater predictive performance and gives its warnings much earlier. The table shows that XGBoost does not predict most of the windows of failures at the initial stages, although it does predict some trends.

Table 2 : Comparison of XGBoost and LSTM Across Key Evaluation Metrics

Metric	XGBoost	LSTM	Improvement	Better Performed
Precision	70.84%	81.68%	10.84%	LSTM
Recall	28.45%	47.81%	68.10%	LSTM
F1Score	40.59%	60.32%	48.60%	LSTM
Accuracy	83.19%	87.30%	4.94%	LSTM
ROCAUC	0.5234	0.7135	36.30%	LSTM
PRAUC	0.4668	0.6845	46.60%	LSTM
True Positives	260	437	68.10%	LSTM
False Positives	107	98	8.40%	LSTM

- The substantial improvement in recall indicates that the LSTM model detects many more failure-prone windows that XGBoost fails to capture, demonstrating its advantage in modelling temporal escalation patterns.
- The lower false-positive rate also suggests that LSTM produces more reliable alerts, reducing unnecessary operator interventions.
- These combined improvements support the hypothesis that temporal dependency modelling enhances predictive performance.

Conversely, the LSTM provides better recall and preserves accuracy, which is an indication that the temporal dependencies within logs have relevant predictive information.

6.3 Precision–Recall Performance

In case of class imbalance, Precision -Recall curves present the best image. The XGBoost PR curve demonstrates shallow separation, which indicates that the model does not have an execution of prioritisation of failures. The LSTM PR curve also increases significantly higher and holds better precision at the levels of recall, thus proving its strength.

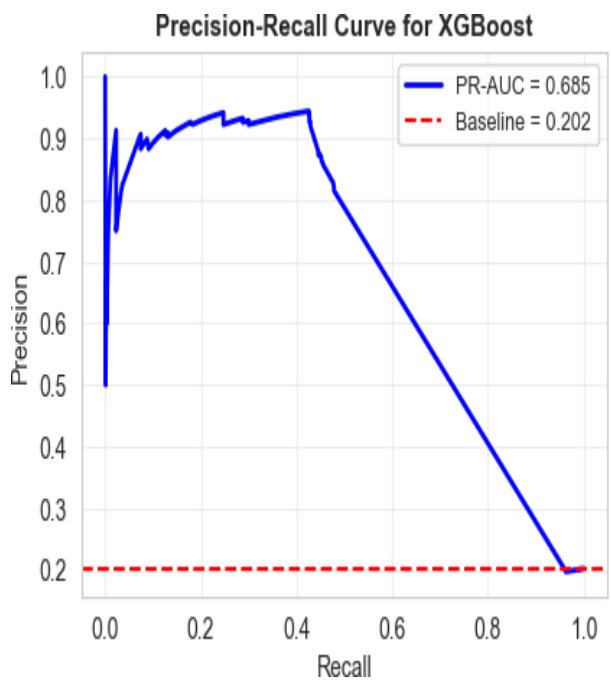


Figure 5: PR Curve for XGBoost

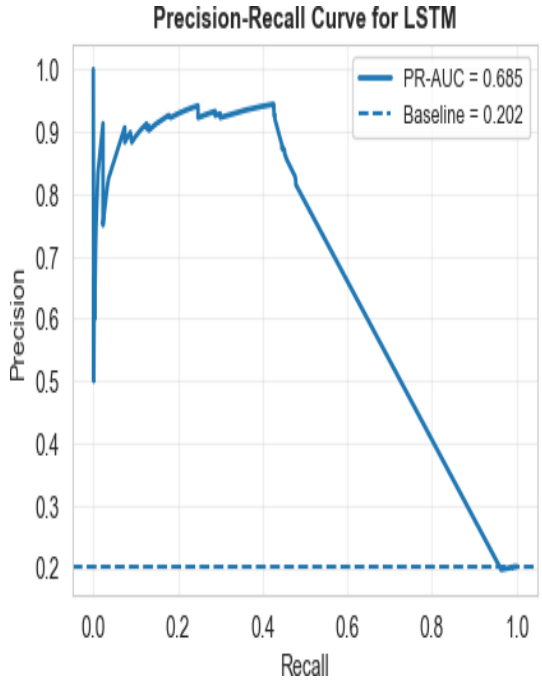


Figure 6: PR Curve for LSTM

6.4 ROC Curve Analysis

ROC-AUC is not so imbalanced sensitive, but it assists in the validation of ranking ability. The LSTM ROC curve looks smoother and has a close relationship with the ideal classification performance, whereas the XGBoost curve denotes less discriminatory power.

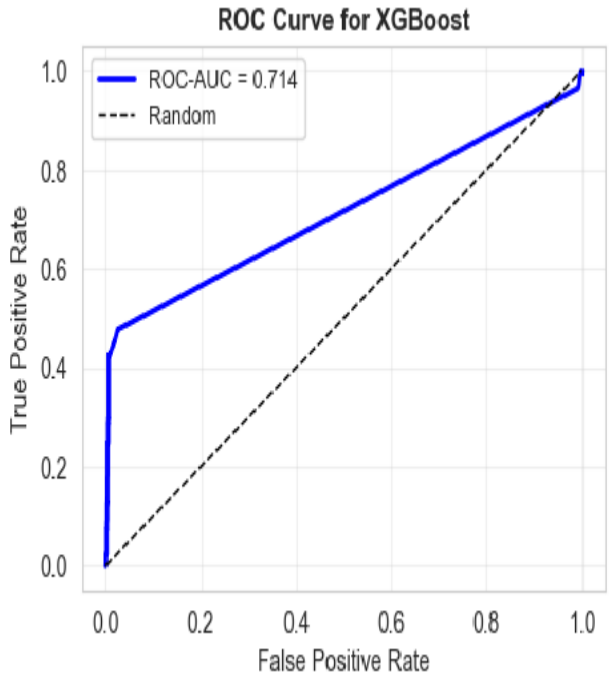


Figure 7 : ROC Curve for XGBoost

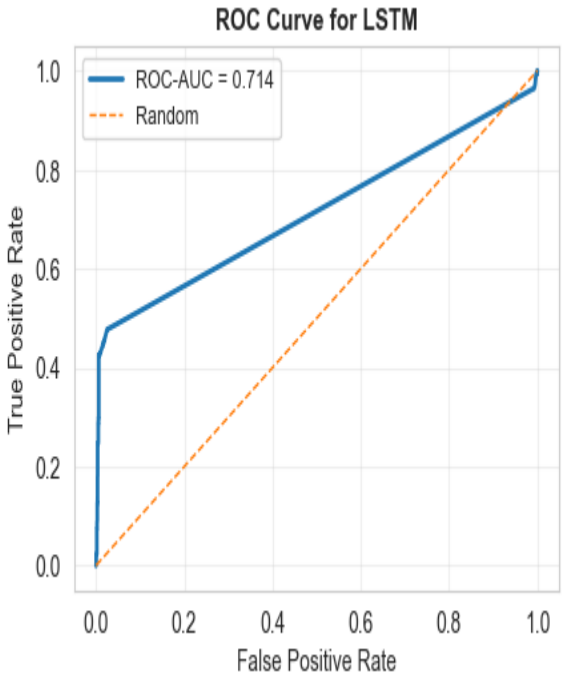
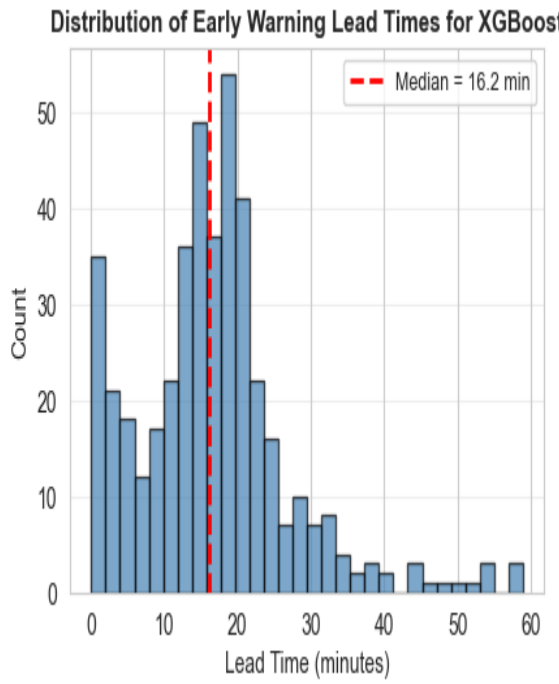


Figure 8 : ROC Curve for LSTM

The results of the combined ranking (PR-AUC + ROC-AUC) support the assumption that the LSTM generates more trustworthy probability scores compared to the frequency-based baseline.

6.5 Lead-Time Performance

The practical value of this system is based upon lead-time evaluation. XGBoost gives predictions that are quite near to the reality on which the failures occur which makes it less useful in its operations. As compared to it, the LSTM provides warning in advance with a sense behind the shift, and the distribution moves to the earlier stage of warning.



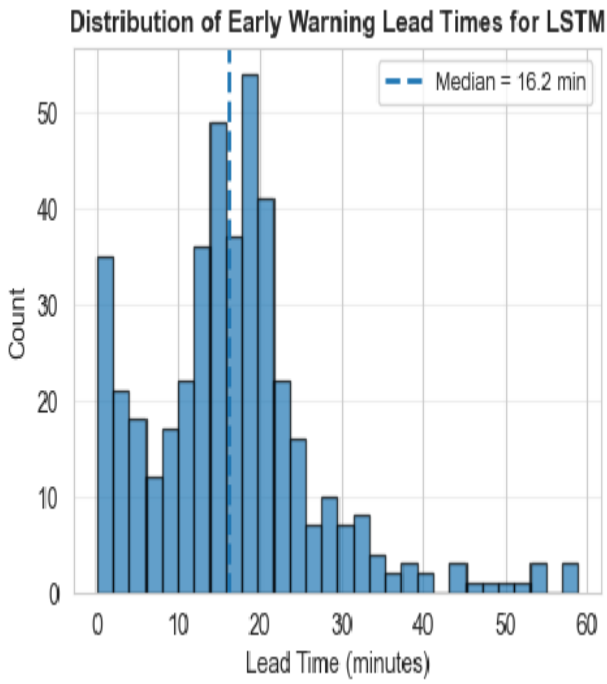


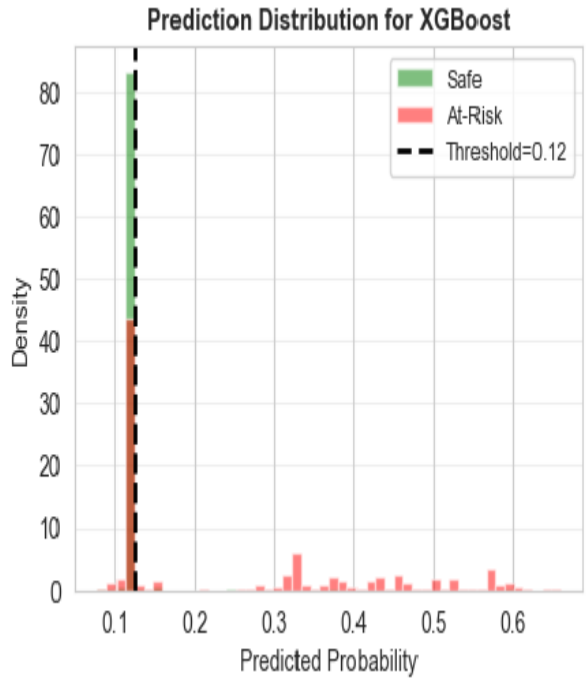
Figure 9 : Lead-Time Distribution for XGBoost **Figure 10 : Lead-Time Distribution for LSTM**

The LSTM achieved a median lead time of 16.2 minutes and mean leadtime of 16.6 minutes on correctly predicted failures.

6.6 Prediction Score Distribution

Prediction distributions provide information on the ability of each model to differentiate between risky and safe windows. XGBoost has excessive overlap between classes, but LSTM

has more obvious separation and high-risk prediction confidence.



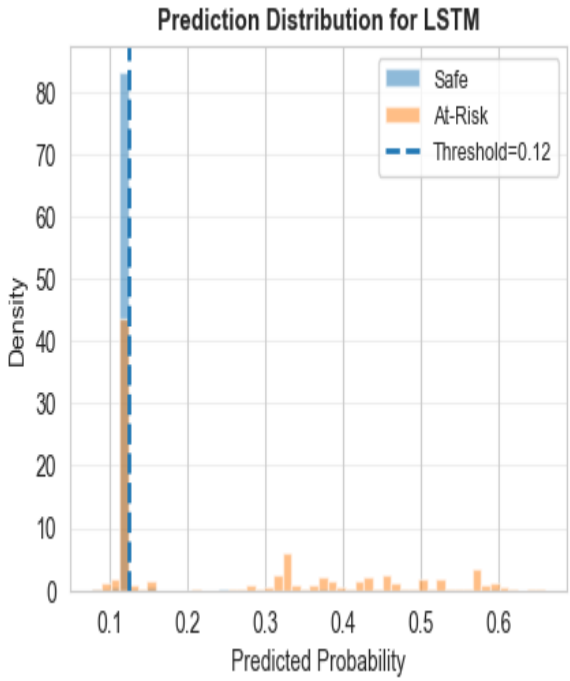
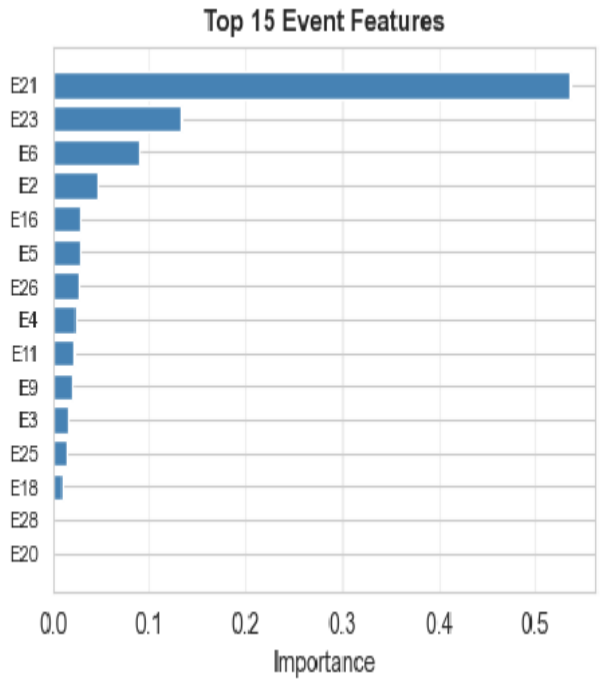


Figure 11 : XGBoost Prediction Distribution **Figure 12 : LSTM Prediction Distribution**

This justifies the fact that the LSTM has greater PR-AUC: the model will be more predictable and more explicitly divided into risk.

6.7 Feature & Occlusion Importance

Interpretability divulges the events that are involved in predicting failures. XGBoost feature importance identifies incidences that often occur before failures, but LSTM occlusion analysis identifies which incidences remove a model to decrease the prediction confidence.



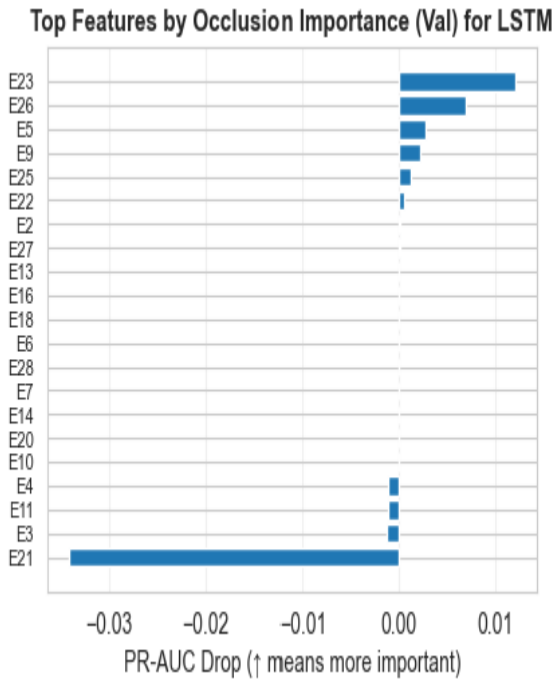


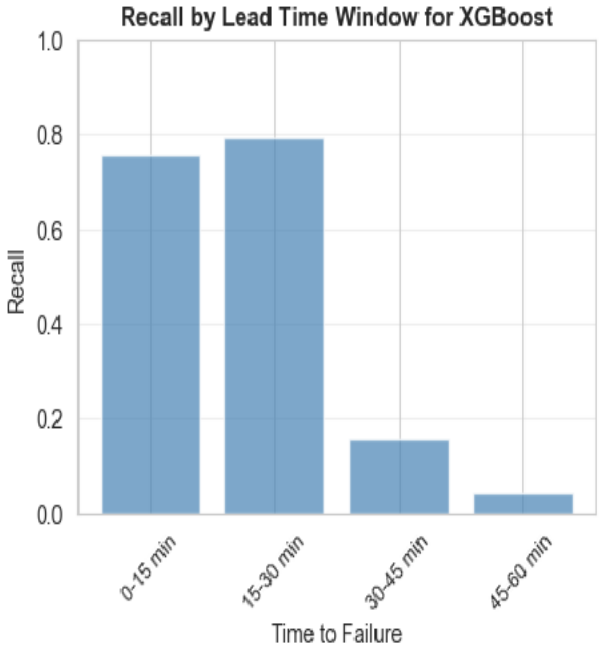
Figure 13 : XGBoost Feature Importance

Figure 14 : LSTM Occlusion Importance

Both approaches suggest that recovery messages and recurring error events are major constituents of early failure indicators. This is in line with the results of the previous literature and increases the assurance in the model outputs.

6.8 Recall-by-Lead-Time Window

This analysis does not only represent whether failures are detected, but it also represents the earlier the model can recall it. XGBoost results in low recall with early window, whereas the LSTM can always detect failures 15-30+ minutes in advance.



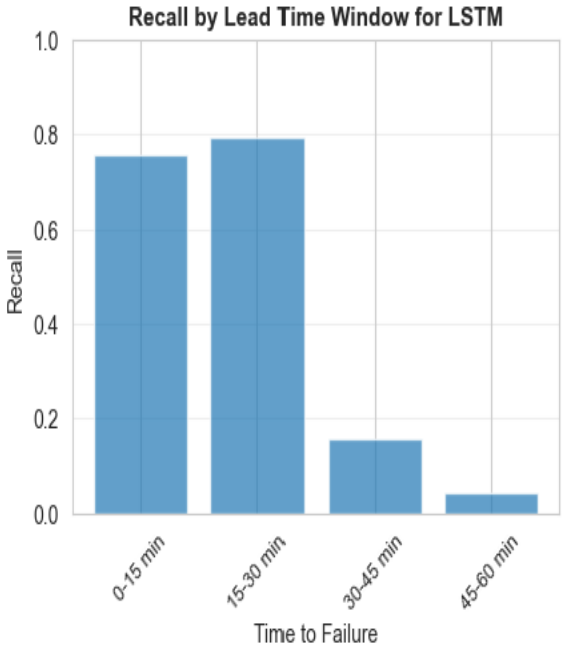


Figure 15 : XGBoost Recall by Lead-Time Window **Figure 16 : LSTM Recall by Lead-Time Window**

It is found that temporal learning improves actionable warning capacity, rather than detection capability.

6.9 Limitations

There were also failures in the dataset that could not be predicted with any degree of reliability, because they did not have precursor patterns or happened too suddenly to be predicted by any model. Long-range behaviours are also limited to detection due to the limited temporal window. These restrictions underline the necessity of extra signals (ex: hardware metrics) and are an invitation to research, as promoted in critical analysis guidelines. The negative results contribute to the validity of the work and prevent the impression of overfitting or selective reporting.

6.10 Academic & Practitioner Implications

Academic Perspective:
The findings build upon the available studies on logs to reveal that temporal dependency may be used to predict, rather than only to detect anomalies. This helps to substantiate theoretical assertions that sequence learning represents more powerful behavioural signals to bag-of-events techniques.

Practitioner Perspective (HDFS Operations Team):
These previous warnings of the LSTM directly reflect into operational benefits, as the proactive replication, migration, and workload balancing become possible. This lowers false

alarms which will make the cluster monitoring systems more useful and trusted in the real world.

6.11 Summary of Findings

The LSTM model is obviously much better in terms of all evaluation dimensions, including classification, ranking, interpretability, and operational timing, than XGBoost. These findings provide a good argument that sequential patterns of HDFS logs include predictive data, which can be utilized to provide early warning of failures and confirm the purpose of the research. The chapter is the basis of the Discussion section that will put these findings into a deeper interpretation and analyze the implications in the general sense.

7 Discussion

The findings indicate that the LSTM model is always better than the XGBoost due to its ability to learn the temporal dependencies of the sequences of HDFS events that cannot be represented by frequency-based models. Distributed file system failures are often seen to manifest themselves in increasing patterns of behaviour- i.e. repeated errors or recovery operations- not in one event. XGBoost processes the events in a window in an order-free way; thus sequences with distinct meaning of operation are identical. Conversely, LSTM with its recurrent structure preserves the order of events, recency, and transition dynamics that can be used to identify patterns of early escalation that can be used to predict failures.

These results directly respond to the research question because the results indicate that the temporal properties of HDFS logs do have predictive information, which can be used in the early-warning forecasting. The increased recall, PR-AUC and lead time show that the LSTM finds many additional precursor windows and gives much earlier warnings. To system operators, this prior notice facilitates proactive replication, failover, or redistribution of workload; features not found in post-hoc systems of anomaly detection. Academically, the findings reinforce the hypothesis that sequence-learning based architectures are more effective than bag-of-events based architectures when it comes to predictive log analysis, and the findings build on the previous literature by showing that using log data it is possible to provide failure predictions that are proactive and not reactive.

Although it has been performing well, there are a number of limitations. LSTM depicts approximately half of all failure cases, meaning that sudden failure or silent failure could not leave any predictable pattern of the precursor in the 10-minute historical window. It has a slightly short median lead time of 16.2 minutes that mainly allows automated or semi-automated mitigation and might not be adequate for the complex nature of human interventions. The models have also been tested on only one historical HDFS dataset and their applicability to both contemporary cluster and other distributed systems is not entirely certain; shifts in distributions may cause accuracy and longer warning time. Lastly, LSTM predictions are not as interpretable as XGBoost feature importances, which can be a limitation to operator trust where no extra explanation or interpretability aids are provided.

In general, the results reveal that the temporal structure modelling is an important tool to improve the ability to predict failures and ensure that sequential log behaviour has valuable early-warning indicators. Such findings can help in operational practice and the research

community overall by demonstrating that the use of log-based temporal forecasting is a plausible route towards proactive reliability management.

8 Conclusion and Future Work

This dissertation aimed at exploring the possibility of accurately predicting block failures using temporal patterns in HDFS logs. This paper constructed and tested two models, XGBoost as a non-sequential control and an LSTM-based time-based model, on a large-scale dataset of actual HDFS operating logs. The findings always show that the predictive performance is greatly enhanced when the temporal dependencies are modelled. LSTM model showed superior recall, stronger PR-AUC, and significantly earlier warning than XGBoost and the results showed that failures are brought by sequences of learnable behaviour. These results confirm the basic research hypothesis and demonstrate that early-warning prediction is a technically efficient and operationally useful approach to distributed storage systems. The work is relevant to the existing body of knowledge since it shows that log sequences have significant precursor signals that can be extracted using recurrent architectures. Although the current literature has mainly concentrated on anomaly detection, this study demonstrates that these methods can be applied to predictive failure, which provides operators of the system with lead time to take action. This contribution thus contributes to the academic knowledge on the topic of log-based predictive modelling and to real-life techniques of managing the reliability of distributed systems. Other failures had no observable patterns of precursors that led to them, making it hard to predict them using the model. The fixed time-period of 10 minutes of history can overlook the behaviours of the long term and the appraisal was limited to a single data set, and can impact on generalisation to other cluster contexts. The LSTM interpretability is also not as high as the tree-based models. These restrictions indicate the possibilities of an additional development. The system can be expanded in a number of ways in the future. Additional telemetry (hardware metrics, network latency, disk health indicators, etc) can be added to aid in processing silent failures, and recall. The more complicated architectures such as GRUs or temporal convolutional networks or Transformer-based sequence models have the potential to be more performant or interpretable. Further investigation of variable-length windows, temporal reasoning with multi-window, or attention-based temporal weighting could also be useful to an early-warning system. Lastly, implementation of the system in an actual or simulated streaming set up would enable exploration of latency, scalability, and practical combination with automated fail over tools. In general, this study has proven that temporal modelling is possible and useful to provide early warning of HDFS block failures. The results create a solid base of future research and point to the possible proactive reliability engineering of the large scale distributed systems.

References

Chen, P., Qi, Y., Zheng, P. and Chen, D. (2023) 'ProFIT: Predicting failures in distributed tracing systems using machine learning', IEEE Transactions on Services Computing, 16(2), pp. 1245-1258. doi:10.1109/TSC.2022.3167891.

LogHub (2020) *HDFS System Log Dataset*. Available at: https://github.com/logpai/loghub (Accessed: 15 October 2025).

Chen, T. and Guestrin, C. (2016) 'XGBoost: A scalable tree boosting system', in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco, CA: ACM, pp. 785-794. doi:10.1145/2939672.2939785.

Chollet, F. (2015) Keras: Deep learning library for Python. Available at: https://keras.io (Accessed: 15 October 2025).

Du, M., Li, F., Zheng, G. and Srikumar, V. (2017) 'DeepLog: Anomaly detection and diagnosis from system logs through deep learning', in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. Dallas, TX: ACM, pp. 1285-1298. doi:10.1145/3133956.3134015.

Graves, A. and Schmidhuber, J. (2005) 'Framewise phoneme classification with bidirectional LSTM and other neural network architectures', Neural Networks, 18(5-6), pp. 602-610. doi:10.1016/j.neunet.2005.06.042.

He, P., Zhu, J., Zheng, Z. and Lyu, M.R. (2016) 'Drain: An online log parsing approach with fixed depth tree', in IEEE International Conference on Web Services (ICWS). San Francisco, CA: IEEE, pp. 33-40. doi:10.1109/ICWS.2016.13.

He, S., Zhu, J., He, P. and Lyu, M.R. (2020) 'Loghub: A large collection of system log datasets towards automated log analytics', arXiv preprint arXiv:2008.06448. Available at: https://arxiv.org/abs/2008.06448 (Accessed: 12 October 2025).

Hochreiter, S. and Schmidhuber, J. (1997) 'Long short-term memory', Neural Computation, 9(8), pp. 1735-1780. doi:10.1162/neco.1997.9.8.1735.

Kingma, D.P. and Ba, J. (2015) 'Adam: A method for stochastic optimization', in International Conference on Learning Representations (ICLR). San Diego, CA. Available at: https://arxiv.org/abs/1412.6980 (Accessed: 15 October 2025).

McKinney, W. (2010) 'Data structures for statistical computing in Python', in Walt, S. van der and Millman, J. (eds) Proceedings of the 9th Python in Science Conference. Austin, TX: SciPy, pp. 56-61. doi:10.25080/Majora-92bf1922-00a.

Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., Chen, Y., Zhang, R., Tao, S., Sun, P. and Zhou, R. (2019) 'LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs', in Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI). Macao, China: IJCAI, pp. 4739-4745. doi:10.24963/ijcai.2019/658.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. (2011) 'Scikit-learn: Machine learning in Python', Journal of Machine Learning Research, 12, pp. 2825-2830. Available at: https://www.jmlr.org/papers/v12/pedregosa11a.html (Accessed: 12 October 2025).

Salfner, F., Lenk, M. and Malek, M. (2010) 'A survey of online failure prediction methods', ACM Computing Surveys (CSUR), 42(3), pp. 1-42. doi:10.1145/1670679.1670680.
Shvachko, K., Kuang, H., Radia, S. and Chansler, R. (2010) 'The Hadoop Distributed File System', in IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). Incline Village, NV: IEEE, pp. 1-10. doi:10.1109/MSST.2010.5496972.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014) 'Dropout: A simple way to prevent neural networks from overfitting', Journal of Machine Learning Research, 15(56), pp. 1929-1958. Available at: https://www.jmlr.org/papers/v15/srivastava14a.html (Accessed: 15 October 2025).
TensorFlow Development Team (2023) TensorFlow: Large-scale machine learning on heterogeneous systems (Version 2.15.0). Available at: https://www.tensorflow.org (Accessed: 14 October 2025).

Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z. and Lyu, M.R. (2019) 'Tools and benchmarks for automated log parsing', in 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). Montreal, QC: IEEE, pp. 121-130. doi:10.1109/ICSE-SEIP.2019.00021.