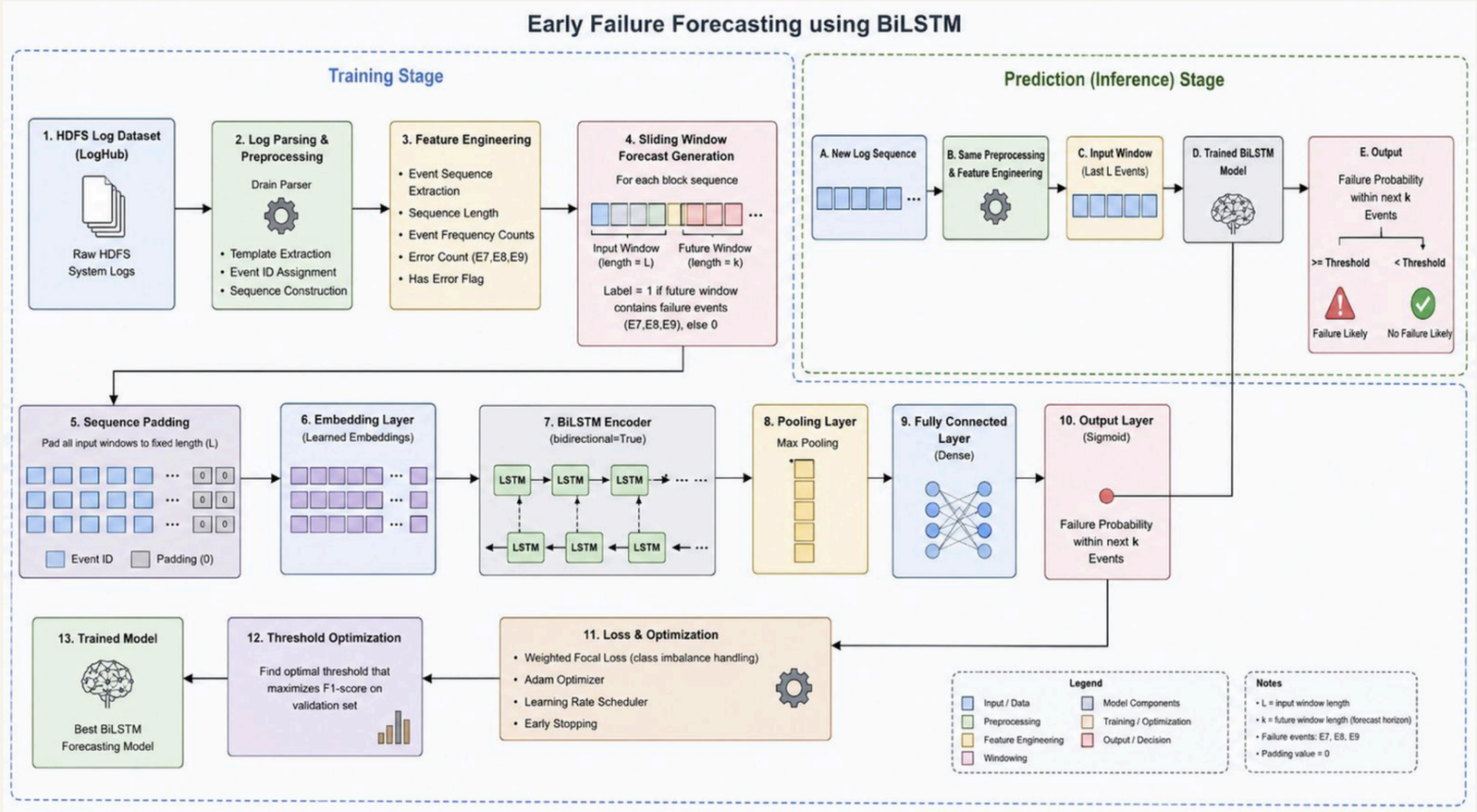
Machine Learning & Pattern Recognition

May 19, 2026

AI3011

Early Failure Prediction in Distributed Systems Using Log Event Sequences



Final Project Presentation

GitHub

Dhruv Nyati ∘ Aarav Jhawar ∘ Phani Srivatsav

PLAKSHA UNIVERSITY

Introduction

Every second, a production HDFS cluster generates thousands of log events. Engineers cannot read them. Rule-based systems cannot anticipate them. And every existing ML system waits for the anomaly to appear before raising an alert.

We asked a different question: can a model learn what failure looks like before it happens, from the sequence of events leading up to it?

Problem Statement

Modern distributed systems like HDFS generate millions of log messages describing system activity, making it difficult to monitor and anticipate failures at scale. This project aims to develop a machine learning system that analyses log event sequences to predict system failures before they occur.

Problem Statement

- Hadoop Distributed File System is a large-scale storage system that runs across clusters of machines. As it operates, it continuously generates millions of log messages recording every action, event, and state change across all components

- The volume of logs makes manual monitoring practically impossible. Traditional tools rely on hardcoded rules that only flag known error types, meaning they react after a failure has already occurred

- Treat logs as sequences of events and apply machine learning to distinguish normal from anomalous behaviour. The model learns to recognise early warning patterns in log event sequences before an actual failure occurs. This shifts monitoring from reactive to predictive, which is the core contribution of the project

DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning

Min Du, Feifei Li, Guineng Zheng, Vivek Srikumar
School of Computing, University of Utah
{mind, lifeifei, guineng, svivek}@cs.utah.edu

Benchmark

F1 Score = 96%

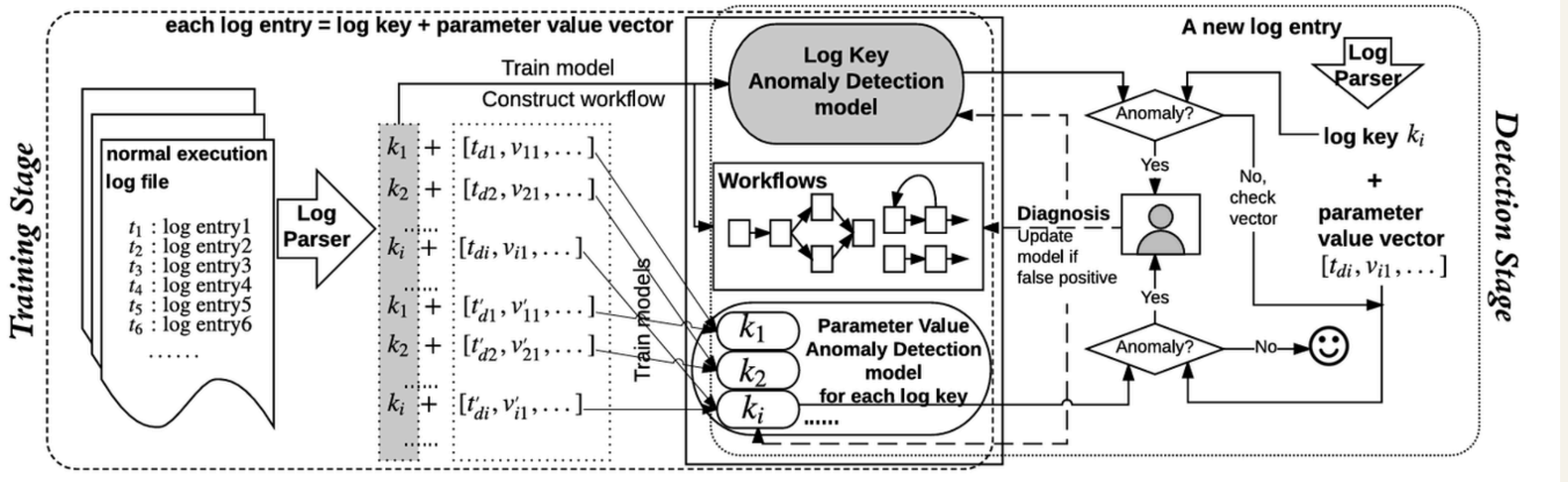
- One of the first deep learning approaches for log anomaly detection
- Models system logs as event sequences
- Uses an LSTM network to learn normal log patterns
- Predicts the next expected log event in a sequence

Gap

- Detects anomalies only after abnormal patterns appear
- Does not predict future failures

log message (log key underlined)	log key	parameter value vector
t_1 Deletion of **file1** complete	k_1	$[t_1 - t_0, \text{file1Id}]$
t_2 Took **0.61** seconds to deallocate network ...	k_2	$[t_2 - t_1, 0.61]$
t_3 VM Stopped (Lifecycle Event)	k_3	$[t_3 - t_2]$
...	...	...

Table 1: Log entries from OpenStack VM deletion task.



Du, Min & Li, Feifei & Zheng, Guineng & Srikumar, Vivek. (2017). DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. 1285-1298. 10.1145/3133956.3134015.

Systematic Evaluation of Deep Learning Models for Log-based Failure Prediction

Fatemeh Hadadi[1] · Joshua H. Dawes[4] · Donghwan Shin[5] · Domenico Lionel Briand[1,2,3]

- Conducts a systematic evaluation of deep learning models for log-based failure prediction.
- Evaluates multiple architectures:

1. CNN
2. LSTM
3. BiLSTM
4. Transformer

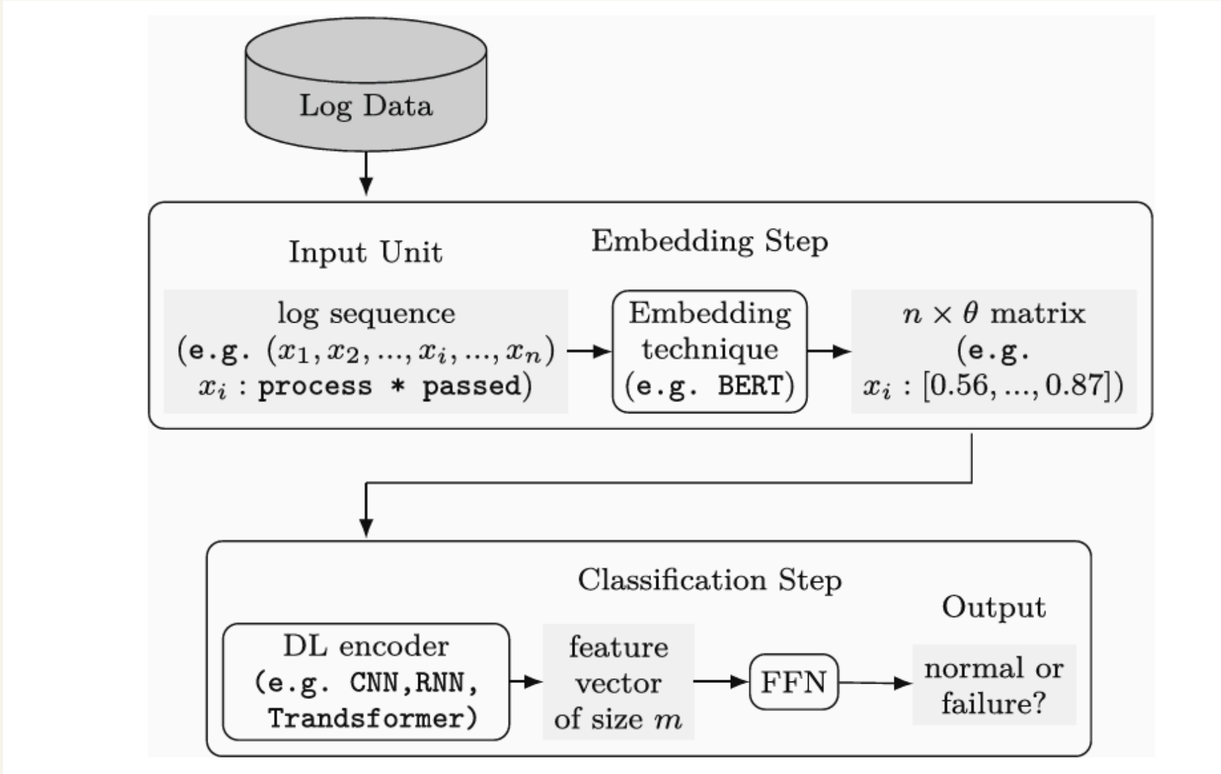
- Uses Logkey2vec embeddings to represent log events

Gap

- Failure prediction is framed as binary classification of entire log sessions
- No prediction horizon is defined, meaning the model effectively predicts at the moment of failure

Benchmark

- Best performing configuration:
  CNN + Logkey2vec
  F1-score = 0.974
- Evaluated on a large real-world industrial log dataset



Hadadi, Fatemeh & Shin, Donghwan & Bianculli, Domenico & Briand, Lionel. (2024). Systematic Evaluation of Deep Learning Models for Log-based Failure Prediction. Empirical Software Engineering. 29. 10.1007/s10664-024-10501-4.

Automated IT System Failure Prediction: A Deep Learning Approach

Ke Zhang*, Jianwu Xu†, Martin Renqiang Min†, Guofei Jiang†, Konstantinos Pelechrinis* and Hui Zhang†
* *School of Information Sciences, University of Pittsburgh*
† *NEC Laboratories America*
* {*kez11, kpele*}*@pitt.edu*, † {*jianwu, renqiang, gfj, huizhang*}*@nec-labs.com*

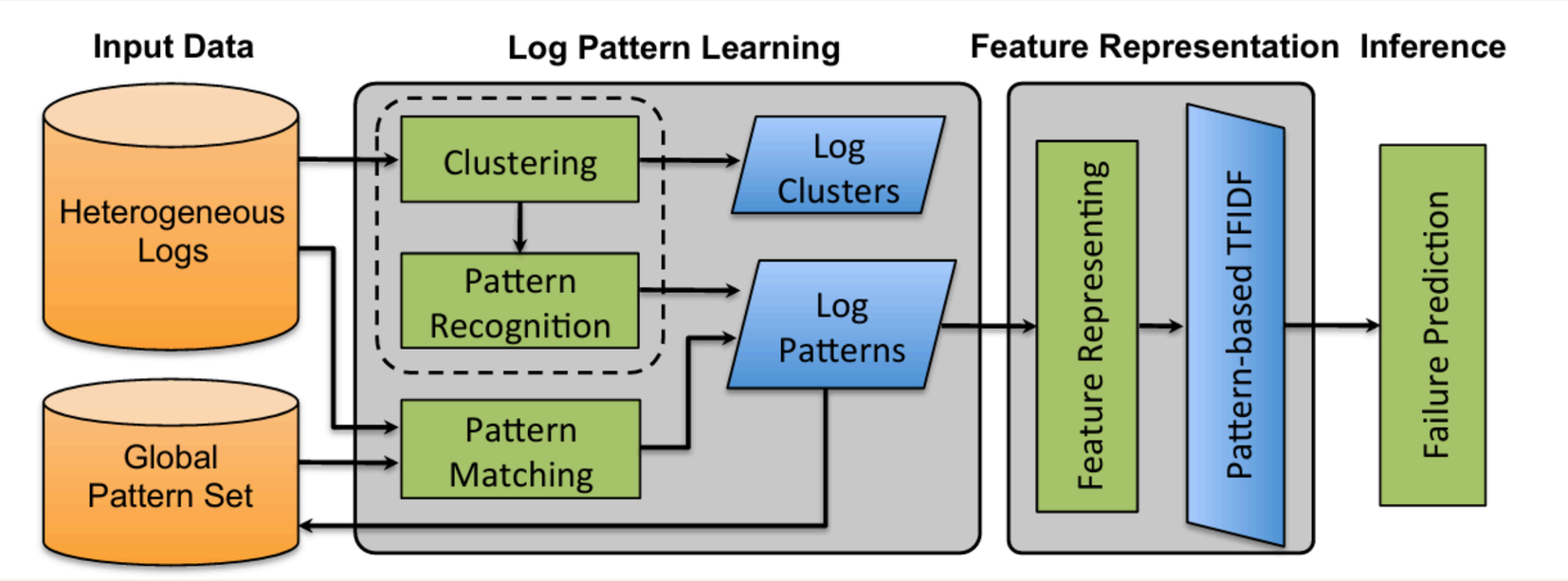
Benchmark

- Evaluated on large-scale console logs from production systems
- Recall = 63.6%

- One of the earliest works framing log analysis as failure prediction instead of anomaly detection
- Uses RNN/LSTM models to analyze streamed system console logs
- Learns patterns in log sequences that precede system failures
- Predicts the likelihood that a system failure will occur soon

Gap

- Does not define a prediction horizon (k events or time window)
- Predictions are made close to the failure moment, providing limited early warning



Zhang, Ke & Xu, Jianwu & Min, Martin & Jiang, Guofei & Pelechrinis, Konstantinos & Zhang, Hui. (2016). Automated IT system failure prediction: A deep learning approach. 1291-1300. 10.1109/BigData.2016.7840733.

Dataset

DATASET SOURCE

The dataset we are using in this project is the HDFS Log Dataset, obtained from LogHub. The original HDFS logs were collected from a production Hadoop cluster during normal system operation. We are using a preprocessed version of this dataset available on hugging face.

DATASET STRUCTURE

Logs are grouped by **block execution**. Each block contains a **sequence of log events** representing system activity during that block operation.

Feature	Description
block_id	unique identifier for block execution
tokenized_block	sequence of encoded log event IDs
label_block_status	Normal / Anomaly

Example:

block_id: blk_-1608999687919862906
tokenized_block: [5,17,22,17,44,9]
label: Anamoly

DATASET CHARACTERISTICS

Number of blocks	575,061
Number of event templates	1755
Average sequence length	239

- Each block contains a variable-length sequence of events
- Events represent system operations such as block transfers, acknowledgements, or verification failures
- Each event corresponds to a log template extracted using the Drain log parser

These labels(Normal/Anomaly) were created during dataset preparation by identifying failed block operations in the system logs.

Dataset

The original dataset contains variable-length sequences, while machine learning models require fixed-length inputs

Window Sequence Generation

Each block sequence is converted into fixed-length forecasting windows using a sliding window approach.

Example:

Original Block sequence :

[E5 E17 E22 E17 E44 E9 E12 E8 E3 E19]

Generated input windows:

[E5 E17 E22 E17 E44]
[E17 E22 E17 E44 E9]
[E22 E17 E44 E9 E12]
[E17 E44 E9 E12 E8]

Generating all windows produced extremely large datasets, so a sampling strategy was used.

Future Window Forecast Generation

From each event sequence, continuous sliding forecasting windows are generated.

Example:

Input Window:

[E5 E17 E22 E17 E44]

Future Window:

[E9 E12 E8 E3 E19]

If the future window contains failure-indicative events:

label =1

otherwise

label =0

Final dataset:

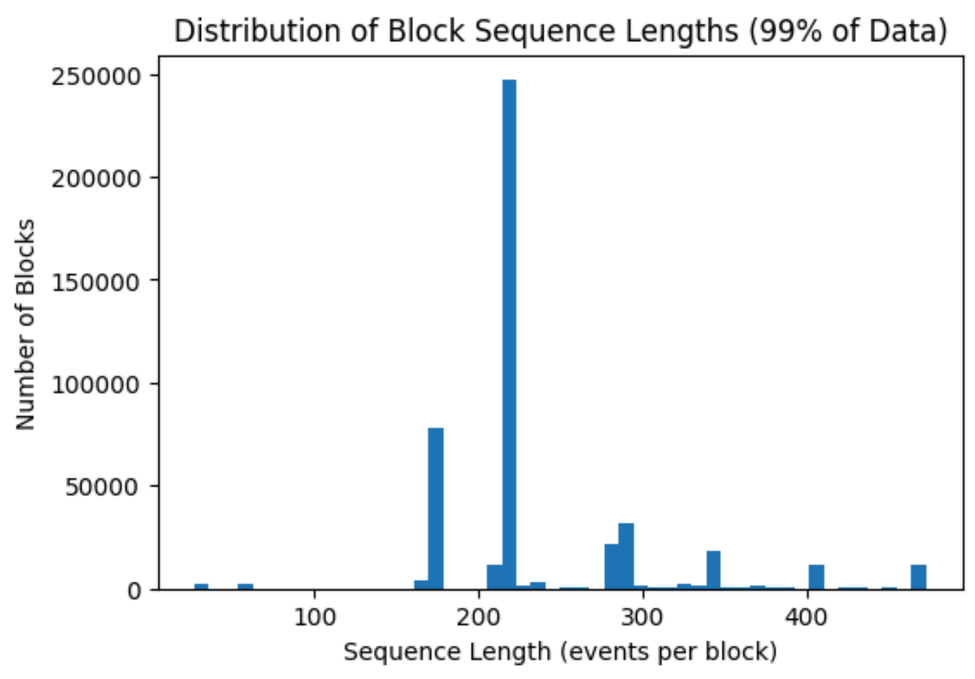
- X → input event windows
- y → future failure likelihood labels

Dataset Preprocessing

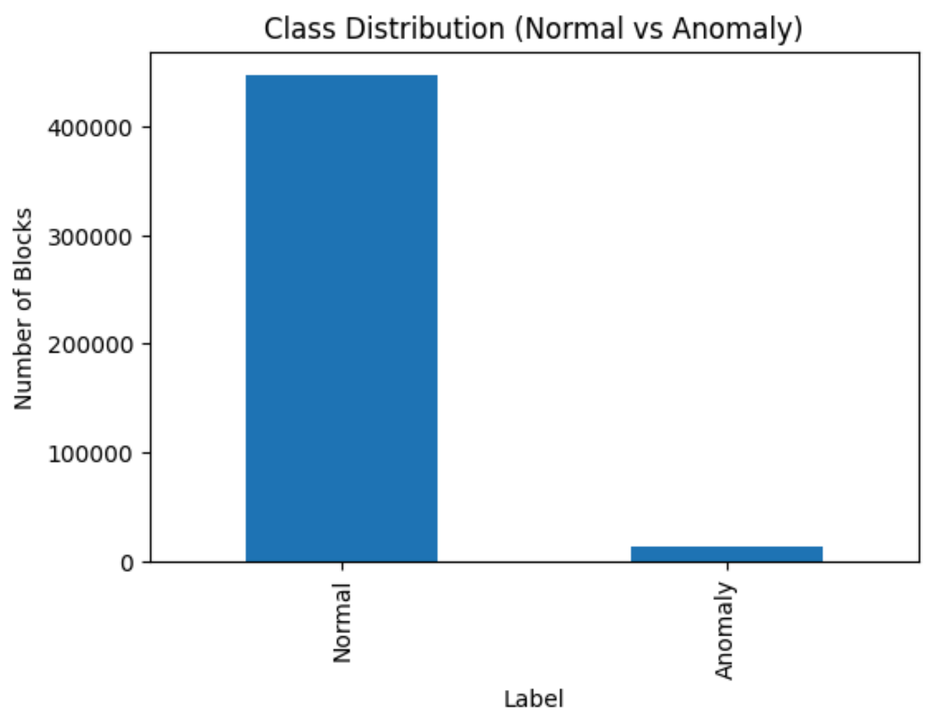
The Drain Parser converts raw unstructured system logs into structured event templates by replacing variable values with placeholders and assigning each template a unique event ID. Additional preprocessing steps include event sequence extraction, sequence length computation, event frequency and error count extraction, forecasting window generation, and sequence padding. These transformations convert raw logs into structured sequential inputs suitable for **BiLSTM-based early failure forecasting.**

Sequence Length Distribution

Shows how many log events occur in each block



Class Distribution



Class Imbalance

Weighted Cross-Entropy Mechanism

We used a weighted learning mechanism to assign higher importance to minority-class failure samples. Misclassifying failures was penalized more heavily, helping reduce majority-class bias and improve anomaly detection.

Focal Loss

We implemented Focal Loss to focus learning on hard-to-classify and rare failure samples:

$FL(pt) = \alpha(1 - pt)^{\gamma} CE(pt)$

This reduces the influence of easy samples and improves failure sensitivity and recall.

Under-sampling Strategy

We applied under-sampling to balance the dataset, maintaining a 3:1 ratio (normal:failure). This reduced class imbalance and helped the model learn minority-class patterns more effectively.

Threshold Optimization

Instead of a fixed threshold (0.5), we dynamically selected a threshold that maximized validation F1-score. This improved:

- precision-recall balance
- anomaly sensitivity
- minority-class prediction performance

Methodology

1. LOG EVENT SEQUENCE INPUT

Input to the system consists of tokenized log event sequences obtained from structured HDFS logs.

Example Sequence:
[E12, E4, E7, E7, E3]

Each event represents a **log template extracted using log parsing techniques such as Drain.**

Why event sequences:

System failures occur due to temporal patterns of events rather than isolated log messages. Sequential event modelling helps capture evolving system behaviour.

2. SLIDING WINDOW FORECAST GENERATION

Example*:
Input Window:
[E12, E4, E7]
Future Window:
[E7, E3]
→ Failure likely within next k events

The event sequence is divided into fixed-length sliding windows to create forecasting samples.

Why sliding window:

This allows the model to learn patterns that precede failures before the full sequence completes, enabling proactive failure prediction.

Methodology

3. FEATURE ENGINEERING LAYER

Sequential logs are transformed into structured numerical representations.

Why feature engineering:

These engineered features help the model identify statistical and temporal anomaly patterns associated with system failures.

4. SEQUENCE MODELING-BiLSTM NETWORK

The embedded event sequence is processed using a Bidirectional LSTM (BiLSTM) network.

BiLSTM captures:

- temporal dependencies
- forward and backward context
- recurring failure patterns
- escalation behavior before failures

Why BiLSTM:

BiLSTM processes sequences in both forward and backward directions, enabling richer contextual understanding of system log behavior compared to standard LSTM.

5. FAILURE PREDICTION LAYER

The output of the BiLSTM is passed through a fully connected prediction layer.

The model predicts:

x% chance of system failure likely in the next **k** events

Example output:

Failure probability = 0.87
Predicted horizon = next 5 events

Base Model - CNN

```
FINAL REPORT

              precision    recall  f1-score   support

0                0.82      0.83      0.82     15776
1                0.68      0.72      0.70      5009

accuracy                             0.80     20785

macro avg       0.75      0.78      0.76     20785
weighted avg    0.79      0.80      0.79     20785
```

- **High recall (0.91)** → CNN effectively detects most failure cases (strong early warning capability)
- **Moderate precision (0.76)** → higher false positives due to lack of temporal understanding
- **F1 score (0.83**) → good balance, making CNN a strong baseline model
- **Limitation** → captures local patterns well but misses sequence dependencies, which later models improve

XGBoost

```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.99      0.95     23045
           1       0.98      0.79      0.88     11522

    accuracy                           0.93     34567
   macro avg       0.95      0.89      0.91     34567
weighted avg       0.93      0.93      0.93     34567
```

- **High precision (0.98)** → makes very confident predictions with fewer false alarms
- **Good F1 score (0.88)** → achieves strong overall performance
- **Lower recall (0.79)** → misses a significant number of failure cases
- **Limitation** → lacks sequence understanding, making it less effective for capturing evolving patterns before failure, which is critical for early detection

```
================================================================================
 --- REAL-TIME FAILURE PREDICTION SIMULATOR (SMOOTHED) ---
================================================================================
Monitoring incoming HDFS logs. Window Size: 20 events.
Goal: Predict the failure before the sequence ends (Length: 33).

[020/33 logs] ✅ OK: System stable. Failure probability -> 0.37
[021/33 logs] ✅ OK: System stable. Failure probability -> 0.37
[022/33 logs] ✅ OK: System stable. Failure probability -> 0.37
[023/33 logs] ✅ OK: System stable. Failure probability -> 0.35
[024/33 logs] ✅ OK: System stable. Failure probability -> 0.32
[025/33 logs] ✅ OK: System stable. Failure probability -> 0.31
[026/33 logs] ✅ OK: System stable. Failure probability -> 0.29
[027/33 logs] ✅ OK: System stable. Failure probability -> 0.29
[028/33 logs] ✅ OK: System stable. Failure probability -> 0.29
[029/33 logs] ✅ OK: System stable. Failure probability -> 0.29
[030/33 logs] ✅ OK: System stable. Failure probability -> 0.29
[031/33 logs] ✅ OK: System stable. Failure probability -> 0.54
[032/33 logs] 🚨 WARNING: Probability of failure rising -> 0.70
[033/33 logs] 🚨 WARNING: Probability of failure rising -> 0.80

*** SYSTEM FAILURE OCCURRED AT THIS EXACT MOMENT! ***
================================================================================
```

Final Model - BiLSTM

FINAL REPORT

	precision	recall	f1-score
overall	0.8098	0.8732	0.8403
accuracy			0.8511
macro avg	0.8098	0.8732	0.8403
weighted avg	0.8098	0.8732	0.8403

```
FINAL TRAINING RESULTS
==========================
Precision : 0.8098
Recall    : 0.8732
F1 Score  : 0.8403
```

- **High recall (0.87)** → effectively detects most failure cases, making it reliable for early warning
- **Strong F1 score (0.84)** → maintains a good balance between precision and recall
- **Captures temporal dependencies** → learns sequence patterns leading up to failure, unlike static models
- **Selected model** → preferred due to higher recall, ensuring fewer missed failure cases

Early Failure Forecasting using BiLSTM

Training Stage

1. HDFS Log Dataset (LogHub)

Raw HDFS System Logs

2. Log Parsing & Preprocessing

Drain Parser

- Template Extraction
- Event ID Assignment
- Sequence Construction

3. Feature Engineering

- Event Sequence Extraction
- Sequence Length
- Event Frequency Counts
- Error Count (E7,E8,E9)
- Has Error Flag

4. Sliding Window Forecast Generation

For each block sequence

Input Window (length = L) | Future Window (length = k)

Label = 1 if future window contains failure events (E7,E8,E9), else 0

5. Sequence Padding

Pad all input windows to fixed length (L)

Event ID | Padding (0)

6. Embedding Layer

(Learned Embeddings)

7. BiLSTM Encoder

(bidirectional=True)

LSTM

8. Pooling Layer

Max Pooling

9. Fully Connected Layer

(Dense)

10. Output Layer

(Sigmoid)

Failure Probability within next k Events

11. Loss & Optimization

- Weighted Focal Loss (class imbalance handling)
- Adam Optimizer
- Learning Rate Scheduler
- Early Stopping

12. Threshold Optimization

Find optimal threshold that maximizes F1-score on validation set

13. Trained Model

Best BiLSTM Forecasting Model

Prediction (Inference) Stage

A. New Log Sequence

B. Same Preprocessing & Feature Engineering

C. Input Window

(Last L Events)

D. Trained BiLSTM Model

E. Output

Failure Probability within next k Events

>= Threshold | < Threshold

Failure Likely | No Failure Likely

Legend

- Input / Data
- Preprocessing
- Feature Engineering
- Windowing
- Model Components
- Training / Optimization
- Output / Decision

Notes

- L = input window length
- k = future window length (forecast horizon)
- Failure events: E7, E8, E9
- Padding value = 0

Implementation & Uses

- Cloud providers like AWS, Google Cloud, and Azure run HDFS-style distributed systems at massive scale. This system would monitor thousands of nodes simultaneously and alert engineers before a node or service goes down

- During high-traffic events like sales or product launches, distributed storage failures can cause checkout failures. This system’s predictive monitoring prevents revenue loss at critical moments

- Core banking platforms process millions of transactions across distributed servers. This system can predict failures before they cause downtime preventing financial loss and service disruption

Future Scope

Explainability of Predictions

- The model flags a sequence as anomalous but does not tell the engineer which specific events triggered the alert
- Integrating attention visualization or SHAP values would make the system actionable rather than a black box, which is critical for real deployment

Time-Based Prediction Horizon

- The current model predicts failure within the next N log events, but operators need time-aware alerts such as "failure likely in the next 5 minutes"
- Future work involves extracting timestamps from raw HDFS logs, computing inter-event time intervals, and reframing the prediction task from event-count horizons to time-based horizons

Thank You