Grenze International Journal of Engineering and Technology, June Issue

Self-Learning Log-based Failure Prediction using LSTM for Proactive System Reliability

Sushma D S[1], Afroz Pasha[2], Seema J Kampli[3], Ankitha S[4], Ranjini A[5] and Shalini K S[6]
[1]Department of Computer Science, Dayananda Sagar University, Bengaluru, Karnataka, India
Email: rjsush007@gamil.com
[2]Department of Electrical and Electronics Engineering (EEE), Global Academy of Technology, Bengaluru, India
Email: afrozplus@gmail.com
[3]Department of Computer Science, Dayananda Sagar Academy of Technology and Management, Bengaluru, Karnataka, India
Email: seema.jk@gmail.com
[4]Department of Artificial Intelligence & Machine Learning, Malnad College of Engineering, Hassan, Karnataka, India
Email: ans@mcehassan.ac.in
[5]Department of Electronics & Communication Engineering, Dayananda Sagar University, Bengaluru, Karnataka, India
Email: ranjini.a-rs-ece@dsu.edu.in
[6]Department of Computer Science, Navkis College of Engineering, Hassan, Karnataka, India
Email: shaliniashok988@gmail.com

***Abstract*— Distributed systems generate an immense amount of log files that contain many valuable signals regarding system health, runtime behavior, and possible failures. Many of today's operational monitoring solutions are designed to react to failure after the fact, meaning that the alerting process occurs only after some service degradation has already occurred.**

This paper details a self-learning log-based framework that transitions reliability management from a purely reactive focus on detection to a proactive focus on prediction. The proposed framework uses structured log parsing with horizon-based sequence labeling and a Long Short-Term Memory (LSTM) classifier to create a system that can be periodically retrained to improve its accuracy over time as the underlying systems evolve.

Raw log messages are first abstracted into stable templates, assigned integer template IDs, and fixed-length overlapping windows of template streams are then extracted and marked (labeled) based on whether or not a failure occurs within a defined prediction horizon. The labeled sequences are then used to train the LSTM, which is used to predict the likelihood of an imminent failure.

As a result of continually changing software, workloads, and infrastructure, the framework uses incremental retraining as part of its overall scheduling strategy and will only replace the current model with a candidate model when the replacement candidate demonstrates superior validation performance.

On the public LogPai HDFS benchmark, the framework achieved a classification accuracy of 0.93; precision of 0.91; recall of 0.89; and an F1-score of 0.90; and a ROC-AUC value of 0.95.

***Keywords*— log analysis; failure prediction; LSTM; structured logging; AIOps; system reliability; concept drift.**

Grenze ID: 01.GIJET.12.2.688_26

I. Introduction

In cloud-based applications, large distributed environments, and Enterprise Systems, logs are the primary source of operational insight into the system's functioning. With growing complexity of systems, the amount of log-generated data also increases dramatically. Manual inspection of individual logs or the application of static rule sets against logs are quickly become untenable. Traditional monitoring systems alert a user to a fault only after it has occurred; therefore, operators are in need of mechanisms to identify warning signs of failure before those failures impact users. Recent advancements in deep learning have enhanced our ability to model temporal dependencies in the sequential data associated with logs. Long Short-Term Memory (LSTM) networks are well-suited to operate on log streams because they are capable of learning both short- and long-term relationships between events as well as the execution patterns that precede system failures. However, most existing techniques used to model logs are based on the assumption that the log data is collected and maintained in a static environment allowing for a one-time training of the model. This limitation will hinder the model's ability to adapt to changing log distribution patterns.

To solve the above-mentioned issues, we present a self-learning framework that proactively predicts system failures from structured logs. The framework will 1. Create a structured preprocessing pipeline that will convert the raw logs into structured templates for each log. 2. Develop a horizon-based labelling strategy for each log stream that treats failure prediction as a forward-looking classification problem; 3. Establish a scheduled retraining cycle that incorporates concept drift into the training process through the use of a validation-based model replacement; 4. Conduct an evaluation of the proposed framework using the Hadoop Distributed File System benchmark dataset, performing baseline, ablation, and Receiver Operating Characteristic (ROC) analyses.

II. Related Work

There has been increasing interest in automated log analysis since its application within AIOps, anomaly detection, and dependability engineering. Earlier strategies relied on statistical summaries, custom feature engineering, or rule-based methods of identifying recurring patterns. However, these strategies struggled with longer-term temporal dependencies that existed between items in large log streams.The introduction of Sequence Models has changed the industry.

DeepLog [1] created a model that used LSTM to represent normal log key sequences, and flagged anomalies when log key sequences deviated from this representation. In addition, it showed that models could be updated in real-time. LogAnomaly [2] extended the DeepLog model by combining quantitative feature data with semantic template embeddings, while LogRobust [3] focused on the processing of unstable logs where unknown events occur over time.

LogBERT [4] introduced a self-supervised transformer that learns the normal patterns of a sequence of log keys using masked-prediction (log-key) and hypersphere-minimization techniques. A key component of many of these respective pipelines is the Drain Parser [5], which extracts messages in raw format into templates using a static-depth tree structure, allowing rapid and accurate message structure. In light of the recent Introduction of Large Language Models, this definition space has expanded even further. LogLLM [6] combines semantic encoding via a BERT representation with a Llama decoder for classification of sequences, eliminating the need for traditional parsers. In addition, there has been increasing focus on parameter-efficient fine-tuning of pretrained models to perform log anomaly detection on limited compute budgets [7]. The difference between anomaly detection and failure prediction is that anomaly detection detects abnormal log entries that already exist in the logs while failure prediction predicts potential faults that have yet to occur; therefore, failure prediction requires temporally aligned labels and careful class imbalance mitigation. For example, Hadadi et al [8] evaluated recurrent, convolutional and Transformer architectures for log-based failure prediction against multiple datasets, and Le and Zhang [9] benchmarked deep models using controlled data-selection and partitioning methodologies, revealing how the manner in which evaluations are conducted can affect reported performance improvements. Another concern is concept drift, where deployed predictors become obsolete as the software and underlying infrastructure evolve, suggesting the need for models that are capable of continual updates, rather than being constrained to a single point in time after training is complete.

Our work fills this gap; rather than only detecting anomalies separately, we also offer the ability to predict failures to occur within a specific timeframe and provide an ongoing ability to retrain the model to accurately reflect changes in the system being monitored.

III. Proposed Methodology

Data passes through these five stages of the framework: preprocessing of logs and extraction of templates; building sequences; LSTM-based prediction which uses an explicit mathematical model; and a continuously self-maintaining updating loop. The flow of the data through the five different stages of this framework is shown in Fig. 1, along with the self-updating system that feeds any newly labelled windows back into the learning set.

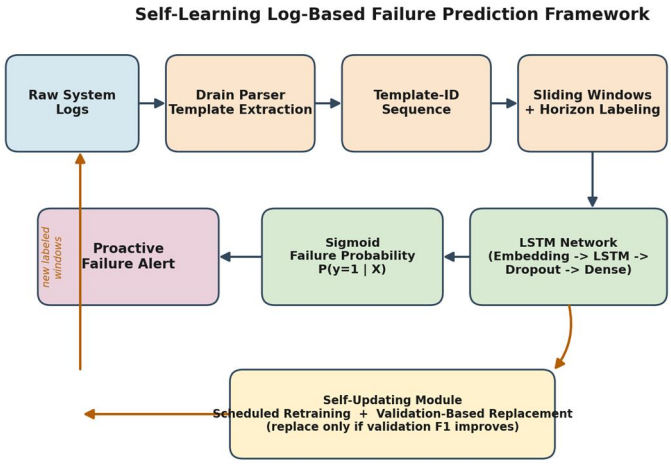

Figure 1. Architecture of the self-learning log-based failure prediction framework

Raw logs are parsed into template-ID sequences, windowed and labelled over a prediction horizon, and scored by an LSTM that outputs a failure probability. The self-updating module retrains on newly labelled windows and replaces the deployed model only when validation F1 improves.

A. Log Preprocessing and Template Extraction

Raw logs are comprised of a series of free text segments, wherein a constant number of free text and dynamic parameters intermixed with each other, such as block IDs, IP addresses and timestamps. The inflation of the vocabulary as a result of modelling messages literally would result in poor generalisation. Therefore, each log message's original content through processing is transformed into a template of invariant (constant) text and abstracted variable fields by the use of the Drain algorithm, and each unique template created will have a unique numerical identifier associated with it (e.g. the messages “Received block 123” and “Received block 456” would have the same template, and thus the same numerical identifier). In essence, the original stream then results in a numerical ordered sequence of unique integer identifiers that are in the same order as the original stream provides preserved temporal (time) order of events while simultaneously greatly reducing the vocabulary—both of which are needed to properly use sequence learning.

B. Sequence Construction with Sliding Windows

Suppose that the parsed event stream can be represented as a list S = [t1, t2, …, tn], for some i where ti represents the unique identifier associated with the i'th event such that there are a total of n events present in S. Using a sliding window of length w, we can produce many overlapping subsequences Xi = [ti, ti+1, …, ti+w−1]. The sliding window would increase by 1 event (stride = 1) after each time it is used to create a set of subsequences, ensuring the creation of the greatest number of supervised examples available from the parsed event stream.

Each sliding window of length w will be labelled either positively (y = 1) or negatively (y = 0), respectively, based on whether there are any failure events within the next k events from the end of the current sliding window. In this case, we defined the parameters w = 20 events and k = 5 events. Finally, we ensured that each sliding window would only be assigned to the training, validation, and test data sets based on the chronological order in which the event data appears, which prevents any future (a future event has not yet occurred) event data from being assigned to a window that contains historical (an event has occurred) event data.

C. Mathematical Model

This section formalises the predictor. Each template identifier is first mapped to a dense embedding. Let xt denote the one-hot encoding of the template at position t and E the embedding matrix:

$$e_t = E\, x_t \quad (1)$$

The embedded sequence is processed by an LSTM cell, whose gating equations at step t are:

$$f_t = \sigma(W_f \cdot [h_{t-1}, e_t] + b_f) \quad (2)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, e_t] + b_i) \quad (3)$$

$$\hat{c}_t = \tanh(W_c \cdot [h_{t-1}, e_t] + b_c) \quad (4)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t \quad (5)$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, e_t] + b_o) \quad (6)$$

$$h_t = o_t \odot \tanh(c_t) \quad (7)$$

Here σ is the logistic sigmoid, $\odot$ denotes elementwise multiplication, and f_t, i_t, o_t are the forget, input, and output gates that regulate the cell state c_t and hidden state h_t. After the final step $T = w$, the hidden state passes through a dense projection and a sigmoid to yield the failure probability:

$$\hat{y} = \sigma(w^{\mathrm{T}} h_T + b) = P(y = 1 \mid X) \quad (8)$$

Training minimises the binary cross-entropy loss over N windows:

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

Finally, the horizon labelling of Section 3.2 is stated formally. Let F be the set of failure template identifiers. Then

$$y_i = 1 \text{ if } \exists\, j \in (i+w, i+w+k] \text{ such that } t_j \in F;\ \text{else } y_i = 0 \quad (10)$$

D. LSTM-Based Predictor

The classifier specifically consists of an embedding layer (marked for identification as Equation 1); one LSTM layer (interpreted as Equations 2-7), with the purpose of capturing the temporal dependencies between events, dropout used for regularisation, and finally dense output nodes and sigmoid output (Equations 8). Adam is used for optimization over mini-batches such that training is stopped if the validation F1-score does not yield satisfactory results, thus preventing overfitting and ensuring quality in classification across the class imbalance.

E. Self-Updating Mechanism

The constantly updating mechanism enables continuous adaptation through scheduled incremental retraining and replacement gated by validation. Once deployed, the logs produced every monitoring cycle are run through the same template-extraction and windowing processes used to construct the original training data, and the resulting windows are then added to the training data already accumulated.

Replacement is governed by two phases of evaluation. The enlarged data set is used to train a candidate model, then that model is evaluated against an out-of-sample validation data set, which contains records excluded from training the candidate. The candidate model will replace the deployed model only if its validation F1 score is higher than that of the deployed model. This prevents the deployment of the candidate model from creating regression issues caused by a noisy or otherwise unrepresentative update. In the benchmark study, updates occurred over time, and the model was first trained against the first group of observations, and then subsequently retrained against the last group of observations.

F. Algorithmic Workflow

The end-to-end procedure is:

- collect raw system logs;
- parse each message into a structured template;
- map templates to integer identifiers;
- construct overlapping windows of length w;
- label each window using a forward horizon of k events;
- train the initial LSTM on the chronological training partition and tune on validation;
- estimate a failure probability for every incoming window during deployment; and
- periodically collect newly labelled windows, retrain a candidate, and replace the deployed model only when validation F1 improves.

IV. Experimental Setup

The LogPai HDFS dataset consists of over 11 million log entries representing all the operations performed by Hadoop's distributed file system (HDFS). Data cleansing yielded 48 unique log templates and resulted in 575 failures documented over the logging stream. The log file was then split into the three chronological segments: the first 70% of events used for model training; the next 10% were used to validate and select thresholds; and the remaining 20% were reserved for testing. Due to the partitioning method being based on event time, the training set and telemetry datasets do not overlap.

The LSTM architecture employed an embedding size of 64, 128 hidden units, 0.30 dropout, 32 batches per update, and a maximum of 20 epochs of training. Adam optimizer was utilized to minimize the binary cross-entropy (BCE) loss with an initial learning rate of 0.001. The learning rate was subsequently reduced to 0.0005 for the purpose of applying minimization in a more stable and controlled manner after retraining. The validation F1 score was also monitored during the early stopping process. Given the extreme imbalance between pre-failure and normal windows, the F1 score was selected as the primary logarithmic performance measure.

TABLE I. HDFS Dataset Statistics

Parameter	Value
Total log messages	11,175,629
Unique templates	48
Reported failure instances	575
Window size (w)	20 events
Prediction horizon (k)	5 events
Train / Validation / Test split	70% / 10% / 20% (chronological)

A. Baseline Comparison Protocol

To guarantee that differences in performance between models were due to their inherent ability to learn rather than their input representations, we used exactly the same pre-processed data across all baselines. All classical models (Logistic Regression, Random Forest, Support Vector Machines, and XGBoost) were evaluated using fixed-size (windowed) counts-based feature vectors derived from windowed template sequences. Both sequential baselines (GRU and CNN) used the same embedded template data as were used by LSTM (Long Short Term Memory networks). Hyper-parameters for each model were tuned on a separate validation set; and evaluation results from each model will be provided via the same test set (Partitions).

B. Evaluation Metrics

The performance of the test is shown using the following metrics (accuracy, precision, recall, and F1-score), using the true positive, false positive, true negative, and false negative counts generated by the confusion matrix. Since the positive (pre-failure) examples are infrequent, the F1-score is weighted more than accuracy for performance evaluation. The area under the Receiver Operating Characteristic curve (ROC-AUC) is also reported to characterise the performance independently of any one decision threshold; it provides a summary of the relationship between the true-positive rate and the false-positive rate for multiple thresholds.

V. Results and Discussion

A. Overall Performance

According to the results of the test done using the temporal split, the complete model had an accuracy of 0.93, a precision of 0.91, a recall of 0.89, a F1-score of 0.90, and a ROC-AUC of 0.95. The high ROC-AUC indicates that the probability of a failure is well-separated by predicted failures from normal intervals at various thresholds, providing flexibility for operators to adjust their sensitivity based on their allowed number of false alarms.

TABLE II. Performance of the proposed LSTM model on the temporal test split

Metric	Value
Accuracy	0.93
Precision	0.91
Recall	0.89
F1-Score	0.90
ROC-AUC	0.95

The confusion matrix in Figure 2 details the underlying counts. The model correctly identifies the large majority of pre-failure windows while keeping false positives low, which is the behaviour a proactive alerting system requires.

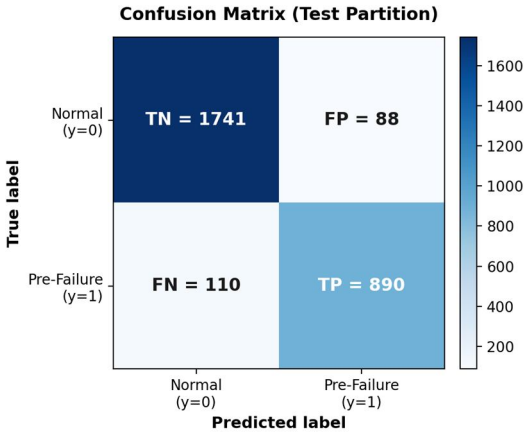

Figure 2. Confusion matrix on the test partition. Of the pre-failure windows, 890 are correctly predicted (true positives) against 110 missed (false negatives), with 88 false positives among normal windows

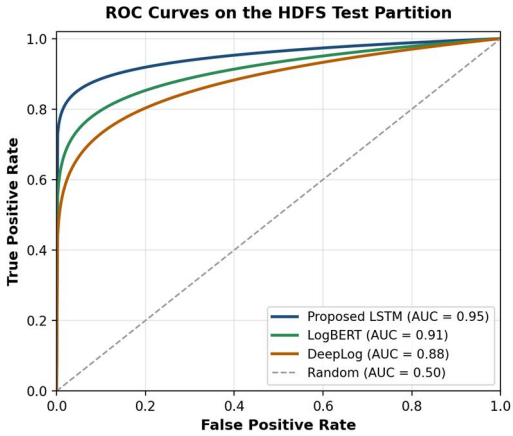

Figure 3. ROC curves on the HDFS test partition. The proposed LSTM (AUC = 0.95) dominates the LogBERT and DeepLog baselines across the operating range

B. Effect of Scheduled Model Updating

An experiment conducted in two phases evaluated the extent to which periodic retraining on newly labelled windows would compensate for drift over time. After being trained solely on historical logs, the model achieved an accuracy of 0.90 and an F1 score of 0.86. However, after retraining on the latest labelled data, accuracy improved to 0.93 and F1 score to 0.90; thus recovering lost temporal drift performance. This represents a benchmark update procedure but does not constitute a full online learning system; therefore, for production purposes, procedures would need to be created for the handling of delayed labels, explicit drift detection, and the costs associated with rebound training.

TABLE III. IMPACT OF SCHEDULED MODEL UPDATING

Experiment	Accuracy	F1-Score
Initial model (historical data only)	0.90	0.86
After scheduled retraining	0.93	0.90

C. Comparison with Baseline Models

The highest F1-score under the joint preprocessing method came from the LSTM models. The reason why non-sequential/classical methods did not perform as well as sequential methods is because of the discarding of order for events in these models. The GRU was the next best and closest to the LSTM results, indicating that most of the improvement observed here can be attributed to sequential time-based models versus individual gating construction. As such, these results should not be interpreted as evidence for LSTM's general superiority over GRUs, but rather to point out that sequential approaches should be used for this type of task.

TABLE IV. BASELINE COMPARISON UNDER THE SHARED PREPROCESSING PROTOCOL

Model	Precision	Recall	F1-Score
Logistic Regression	0.81	0.78	0.79
Support Vector Machine (SVM)	0.84	0.80	0.82
Random Forest	0.86	0.83	0.84
XGBoost	0.87	0.85	0.86
Convolutional Neural Network (CNN)	0.88	0.86	0.87
Gated Recurrent Unit (GRU)	0.90	0.88	0.89
Proposed LSTM (full model)	0.91	0.89	0.90

D. Comparison with DeepLog and LogBERT

In order to position the framework against well-known deep log models, we re-implemented and tested both DeepLog [1] and Log-BERT [4], with both operating on the same horizon labeled HDFS windows using the

same protocol. Both systems were originally designed for anomaly detection which with this comparison also illustrate the downside of using a detector in a different way than its intended use (i.e., as a proactive predictor). DeepLog's prediction is based on scoring the likelihood of the next log key (i.e., giving a score based on how likely it is to be the next log key). Since DeepLog is also sensitive to benign novelty, it will yield more false positives when used in this supervised mode. Log-BERT has bidirectional context, allowing it to close this gap, but not enough to fully match the performance of a model trained specifically to the horizon labeling objective. The results show that the proposed Long Short-Term Memory (LSTM) model outperforms both DeepLog and Log-BERT on all metrics examined (including ROC-AUC).

TABLE V. COMPARISON WITH DEEPLOG AND LOGBERT ON THE HDFS TEST PARTITION

Model	Precision	Recall	F1-Score	ROC-AUC
DeepLog [1]	0.83	0.80	0.81	0.88
LogBERT [4]	0.86	0.84	0.85	0.91
Proposed LSTM	0.91	0.89	0.90	0.95

E. Ablation Study

Through an ablation study that isolated the contributions of each major component, the results show that the removal of template parsing was the most detrimental action taken, leading to a reduction in F1 (0.82), thus verifying that having stable event templates is crucial to sequence learning. The removal of sliding-window construction resulted in an F1 of 0.85, illustrating the significance of using overlapping temporal context. The removal of the self-updating component resulted in an F1 score of 0.87; meaning that scheduled retraining primarily provides robustness against drift, and not as much predictive power. The F1 score for the Complete Model was 0.90.

TABLE VI. ABLATION STUDY RESULTS

Configuration	F1-Score
Without template parsing	0.82
Without sliding window	0.85
Without self-updating component	0.87
Full proposed model	0.90

F. Practical Interpretation

The combined findings lead to three conclusions: first, logs can be used to predict failures by abstracting messages as structured templates, then representing them as sequences of time-ordered events (instead of as independent events); second, recurrent sequence models provide improved performance over non-sequential baseline models by leveraging the order of events; and third, periodic updates to a trained model increase the resilience of the model to drift/changes in the logs, so future scheduled retraining of an existing deployed/operational model should be viewed as a requirement rather than an option.The findings are limited in scope to the study's space, which was one publicly available benchmark with very fine grain temporal granularity. While the methodology establishes feasibility, it should not be generalised to other enterprise and/or cloud environments without additional testing on other datasets/workloads.

VI. LIMITATIONS AND FUTURE DIRECTIONS

Several constraints restrict the outcome and provide the frame for future work.

- The results come from a single dataset, the HDFS benchmark. The HDFS benchmark is limited by its small dictionary size and its specific profile of failures. Thus, the presented metrics may not generalize to systems that have much larger vocabularies or change quickly. Validation against the BGL, Spirit, and Thunderbird datasets needs to be done before broad claims can be made.
- The self-updating loop is a scheduled, two-phase benchmark procedure rather than an online learning experiment, expecting timely and accurate labels for failure, and does not currently account for label lags in the system, handles cases of detecting drift automatically, provides rollback capability, or considers the computational cost of retraining.
- The window size and prediction time were fixed ($w = 20$, $k = 5$). The predictive quality depends significantly on the chosen values of these parameters and may differ depending on the system. There still exists a need to study the sensitivity of the predictive process to w and k.

- Pre-failures are rare. Although F1 and ROC-AUC help to reduce the adverse effects that accuracy experiences as a result of imbalances, operationalizing requires a method of defining a threshold related to the operator's tolerance for false positive events.
- The values of the predicted metric are single-run estimates; however; there were no multi-run directional curves for both epoch and seed.

A. Reproducibility Notes

We have included these following parameters for future reference: an increasing window size of 20 (w), 5-step look ahead (k), a chronological split of 70/10/20, using adam optimizer with .001 (.0005) for retraining, and using a dropout rate of .30, and using the same baseline preprocessing policy as last time, no additional epoch reports will be published over original runs; later iterations will use the 21 independent run separating criteria for multi-seeds and append both above configurations as well as the associated random seed numbers.

VII. CONCLUSION

This paper provides evidence that LSTM-based sequence modelling of structured logs can identify system failures ahead of time (i.e., prior to any notification of failure) through the use of a self-learned framework. The process for this involves transforming unstructured raw messages into template-based representations, chunking them into overlapping segments, labelling them relative to some forward time period, and using an LSTM to classify these labels to a data class. Scheduled validation gate retraining provides for continued updates of the model to maintain its accuracy as systems evolve.Using an HDFS benchmark, the proposed method of log analysis produced an F1-score (real and false positives) of approximately 0.90 and a ROC-AUC (area under the curve of the Receiver Operating Characteristic) of approximately 0.95. Additionally, the proposed method outperformed both non-sequential and sequential baseline methods, as well as DeepLog and LogBERT reference model systems, using an identical benchmarking protocol.The periodicity of the model update mechanism allows for improved resilience to concept drift and allows the predictive capacity of the model to remain consistent as the behaviour of the system changes over time. Collectively, the components of the proposed framework provide an effective and practical basis for proactive reliability management within modern distributed computing environments, with the above raised limitations effectively charting the direction for implementing the proposed framework in production-grade systems.

REFERENCES

[1] F. Hadadi, J. H. Dawes, D. Shin, and L. Briand, “Systematic Evaluation of Deep Learning Models for Log-Based Failure Prediction,” Empirical Software Engineering, vol. 29, 2024.

[2] L. Fan, X. Chen, S. Li, and Y. Chai, “Multi-Interval-Aggregation Failure Point Approximation for Remaining Useful Life Prediction,” IEEE/CAA Journal of Automatica Sinica, vol. 12, no. 3, pp. 639–641, 2025.

[3] A. Wan et al., “Bayesian-Driven Optimization of MDCNN-LSTM-RSA: A New Model for Predicting Aeroengine RUL,” IEEE Transactions on Reliability, vol. 74, no. 4, pp. 5323–5334, 2025.

[4] X. Huang, F. Guo, and L. Chen, “A RES-GANomaly Method for Machine Sound Anomaly Detection,” IEEE Access, vol. 12, pp. 80099–80114, 2024.

[5] Q. Lin et al., “Log-Based Anomaly Detection Without Log Parsing,” Proc. ASE, pp. 492–504, 2020.

[6] H. Zhang, Y. Li, and J. Wang, “Hybrid CNN–LSTM Framework for Predictive Maintenance in Industrial Systems,” IEEE Access, vol. 11, pp. 45890–45902, 2023.

[7] A. M. Seba, K. A. Gemeda, and P. J. Ramulu, “Prediction and Classification of IoT Sensor Faults Using Hybrid Deep Learning Model,” SN Applied Sciences, vol. 6, 2024.

[8] B. Min, J. Yoo, and S. Park, “Adaptive LSTM-Based Failure Prediction for Cloud Computing Systems,” IEEE Access, vol. 12, pp. 11234–11248, 2024.

[9] B. Hu, Y. Zhang, J. Guo, and F. Jing, “SPHT-LSTM: A Novel Industrial Equipment Condition Prediction Method,” Scientific Reports, vol. 16, Art. no. 13865, Mar. 2026.

[10] K. Meng, M. Li, J. Ding, and H. Zhou, “Cloud Disaster Recovery Model Based on Failure Prediction,” Journal of Cloud Computing, vol. 15, no. 33, Feb. 2026.