Journal of Computer Applications and Information Technology
Vol. 2, Issue. 3, Sep 2026, pp. 59~71
ISSN: 3107-6424, DOI: 10.32595/jcait/v2i3.2026.42

Explainable Machine Learning Framework for Predictive Resource Failure Detection in Distributed Computing Environments

Dr.Aida Ventkat Rao Dora[1], Dr.Abhishek Kumar Gupta[2]

[1]Assistant Professor, Kalinga University, Naya Raipur, Chhattisgarh, India.
[2]Kalinga University, Naya Raipur, Chhattisgarh, India.

Article Info

Article History:

Received Jul 18, 2026
Revised Aug 23, 2026
Accepted Sep 07, 2026

Keywords:

Explainable Artificial Intelligence (XAI),
Machine Learning,
Predictive Failure Detection,
Distributed Computing,
Resource Monitoring,
XGBoost,
SHAP,
Feature Importance,
Predictive Analytics

ABSTRACT

Large-scale distributed computing systems produce millions of telemetry events per day, yet existing failure prediction depends on static thresholds, ignores feature interdependencies and offers minimal interpretability. This research introduces the Explainable Machine Learning Framework for Predictive Resource Failure Detection (XML-PRFD), which integrates telemetry preprocessing, tuned supervised classification, decision-threshold calibration, and interpretability through SHAP values and permutation importance. The framework is evaluated on a synthetic dataset of 12,000 observations and ten features derived from seven interaction-dependent failure modes, split 80:20 into stratified training and test subsets, with Logistic Regression, Random Forest and XGBoost given equal tuning budgets. XGBoost achieves the best test-set performance, with 84.00% accuracy, 71.16% precision, 54.06% recall, an F1-score of 61.45% and a ROC-AUC of 0.810. SHAP attributes 43.3% of total feature importance to memory utilization and 20.4% to CPU utilization, and permutation importance identifies the same leading predictors (Spearman $\rho = 0.78$). The results demonstrate that the integration of tree ensemble methodologies with Shapley-value-based explanatory frameworks yields failure predictions characterized by both high accuracy and comprehensive auditability.

Corresponding Author:

Dr.Aida Ventkat Rao Dora,
Assistant Professor,
Kalinga University, Naya Raipur, Chhattisgarh, India.

1. INTRODUCTION

Resource failures constitute a standard phenomenon in expansive distributed computing infrastructures: empirical investigations of high-performance computing installations report tens of thousands of node interruptions annually [1], and the Google cluster trace demonstrates persistent task and machine failures amid rapidly evolving workload patterns [2]. These infrastructures additionally produce extensive monitoring telemetry, encompassing processor and memory utilization through I/O latency, network propagation delays, and garbage-collection suspension intervals, which failure prediction endeavors to convert into advance notifications [3]. Nonetheless,

the majority of monitoring frameworks continue to employ invariant thresholds applied to individual metrics independently, despite the fact that numerous failed storage devices exhibit no indication across any discrete SMART parameter [4] and cloud job failures emerge from the combined influence of resource allocation, task scheduling, and job specifications [5]; consequently, a single-metric threshold approach fails to identify failures resulting from synergistic combinations of modest indicators.

Machine learning models learn non-linear decision boundaries from telemetry and have been deployed for node-failure prediction in production clouds [6], but the best-performing models, such as tree ensembles, lack transparency, and operators need to know which resource triggered an alert to act on it, a need addressed by explainable artificial intelligence (XAI) [7], [8].

This paper introduces an Explainable Machine Learning Framework for Predictive Resource Failure Detection (XML-PRFD) that integrates gradient-boosted trees [9] with Shapley-value-based explanations [10] and permutation importance [11], [12], providing a transparent failure-prediction module for machine learning-driven administration of cloud-native systems [13]. The principal contributions include:

1. A modular architecture integrating telemetry data preparation, failure categorization, threshold determination, and a dual-method explanation component.
2. A synthetic evaluation dataset comprising 12,000 samples representing seven distinct failure patterns, enabling validation of explanations against established reference outcomes.
3. An equitable evaluation of Logistic Regression, Random Forest, and XGBoost utilizing consistent hyperparameter optimization budgets and a single evaluation of the reserved test dataset.
4. A mutually-validated SHAP and permutation-importance examination, incorporating robustness assessments across two alternative failure conditions.

The rest of the paper is organized as follows. Section 2 reviews related work, Section 3 presents the proposed framework and Section 4 its working. Section 5 reports the results, Section 6 discusses them and Section 7 concludes the paper.

2. LITERATURE SURVEY

This area has moved from simple threshold-based models to machine learning based approaches, and recently interpretable models. Many early works focused on anomaly detection [14]. However, in practice it was found that failure rates varied drastically between 22 HPC clusters over 9 years [1] and many failed disks in a large population (>100k) of Google drives showed no indication of failure from any SMART attribute [4]. Online failure prediction techniques can be divided into symptomatic, error based and auditing [3], and have since been dominated by supervised learning. These range from tree based models predicting drive and sector errors [15] and disk errors for VM placement [16] to the MING system that creates rankings on cloud nodes based on temporal and spatial failure trends [6]. Analysis has been performed at job level on the Google cluster traces showing failure correlation with resource usage and resubmission [2], [5]. Classifiers have been used to reduce waste caused by resource failure [17]. Long Short-Term Memory (LSTM) networks have been used to predict application failure in the cloud [18] and node failure in HPC systems [19]. Another approach is log mining. Log based detection has been benchmarked [20] and has been extended to the stack LSTM model DeepLog [21]. Detection has been extended using machine learning to detect failure in wireless IoT networks [22]. In terms of tabular data,

Random Forests use an average of uncorrelated trees [11]. Gradient boosting uses trees sequentially to fit the gradient of the loss function [23]. XGBoost improves on this by introducing regularisation and second order optimisation [9]. LightGBM provides faster training times using histograms [24]. Studies have shown that ensemble models provide better results [25]. Interpreting models is seen as essential to ensure trust in black-box models [7], [8]. Local Interpretable Model-agnostic Explanations (LIME) uses local approximation with a linear surrogate [26]. SHAP values use game theory to ensure consistent feature attribution [27]. TreeSHAP calculates exact values for tree models [28]. Permutation Importance determines how much each feature contributes to the performance of the model [11], [12].

Three novel aspects have been considered in this work. Firstly, simple threshold-based [14] and statistical models [1], [4] are easily interpretable but do not consider any metric interactions. Machine learning based predictors [6], [15]–[19] do consider metric interactions however rarely provide an indication of what resource caused the alarm, although this has been partially addressed using keys in DeepLog [21]. Secondly, deep sequence models [21], [18], [19] are not interpretable. Tree-based methods [9], [11] have similar performance on tabular data and produce exact Shapley values [28]. Lastly, failure data does not contain any labelling for causes meaning that the interpretations cannot be verified. Baseline models are often not as carefully tuned as the proposed model. The XML-PRFD approach uses a well tuned XGBoost model along with SHAP and permutation values using cross-validation. A fair tuning approach is used and a simulated dataset with known failure mechanisms. Table 1 summarizes representative studies on failure prediction and detection and contrasts them with this work.

Table 1. Representative studies on failure prediction and detection

Ref.	Data / domain	Method	Explanations	Main limitation
[4]	100,000+ disks, Google	Statistical analysis of SMART attributes	Descriptive only	Many failures show no single-attribute warning
[15]	Disk and sector errors, data centre	Tree-based and other classifiers	Not reported	Storage only
[16]	Disk errors, cloud platform	Learning-to-rank predictor	Not reported	Storage only; no operator view
[6]	Nodes, commercial cloud	Temporal + spatial models with ranking (MING)	Not reported	Opaque to operators
[17]	Google cluster trace	Supervised classifiers	Not reported	Job level; no explanation
[18]	Cloud applications	LSTM	Not reported	Opaque deep model
[19]	HPC system logs	Deep recurrent model (Desh)	Not reported	Log-only input
[21]	System logs	LSTM anomaly detection (DeepLog)	Partial (log keys)	Detects anomalies, not resource failures
This work	Simulated multi-resource telemetry	Tuned XGBoost vs RF vs LR	SHAP + permutation importance, cross-checked	Simulated data; production validation pending

3. PROPOSED FRAMEWORK

XML-PRFD is a 5-stage pipeline for generating failure alerts with evidence from raw telemetry data.

3.1. Telemetry Ingestion and Feature Space

The ingestion stage processes raw telemetry data to generate 1 snapshot per node and monitoring period containing the 10 features shown in Table 2. These features provide an overview of the health status of a compute, memory, storage, networking, application and environment. We aim to predict a binary label indicating if the node has failed during the prediction period.

Figure 1 shows the general framework. Raw telemetry data are taken in at the top, they then go through the pipeline of preprocessing, learning, decision and explanation stages to produce an alert and explanation to be passed to an operator.

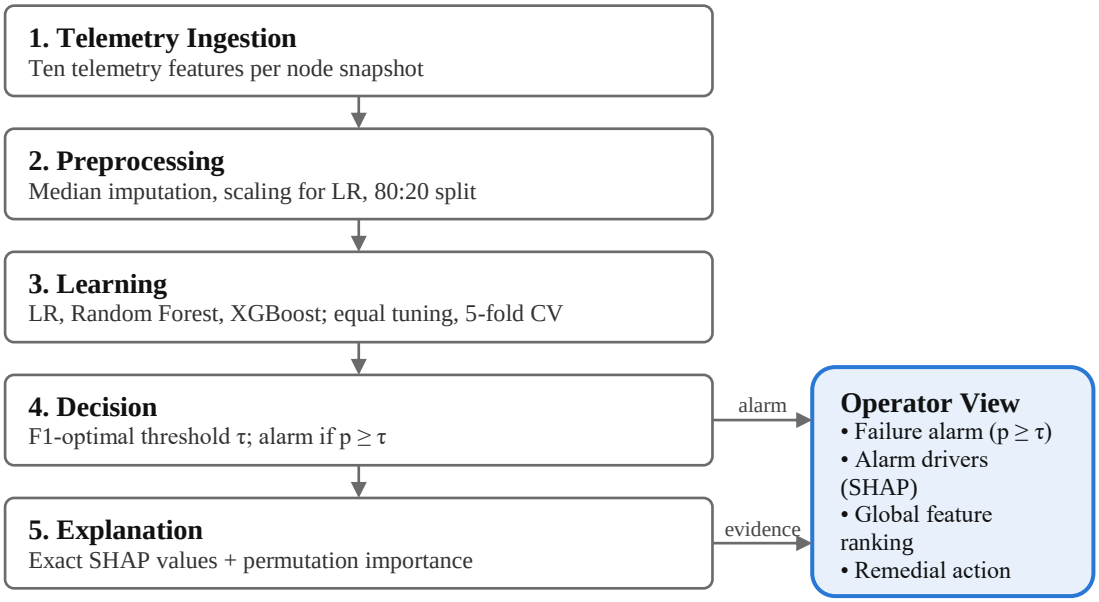


Figure 1. Architecture of the proposed XML-PRFD framework

The decision layer converts a failure probability into an alarm, while the explanation layer attributes it to individual telemetry features; each layer is described below.

Table 2. Telemetry features used by the framework

No.	Feature	Unit	Resource group	Typical failure relevance
F1	CPU utilization	%	Compute	Saturation, scheduling delays
F2	Memory utilization	%	Memory	Out-of-memory kills, swapping
F3	Disk I/O wait	ms	Storage	I/O contention, slow disks
F4	Network latency	ms	Network	Congestion, partitions
F5	Error rate	% of requests	Application	Software faults, error bursts
F6	Request rate	requests/s	Workload	Load spikes
F7	Node temperature	°C	Environment	Thermal throttling
F8	Active threads	count	Application	Thread exhaustion
F9	GC pause time	ms	Runtime	Memory pressure, stalls
F10	Uptime	hours	Lifecycle	Resource leaks, ageing

3.2. Preprocessing Layer

The telemetry has missing values due to reboots and dropped packets. The pre-processing stage imputes these using the median of each feature. This is calculated from the training data to avoid any information leaking into the model during testing. The features are scaled to have 0 mean and unit variance for the Logistic Regression algorithm. Tree based models are used with no

scaling as they are invariant to monotonic transformations. The data was split into a stratified training set (80%) and test set (20%) to maintain the failure rate.

3.3. Learning Layer

The three classifiers below are trained on the same inputs. Logistic Regression is used as an interpretable baseline model. Random Forest is a bagging ensemble of decision trees [11]. XGBoost is the main model in our framework that trains an additive ensemble of K regression trees by minimizing the regularized objective function in Equation (1) [9]:

$$\mathcal{L} = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k), \qquad \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2 \tag{1}$$

where ℓ is the logistic loss function, T is the number of leaves in a tree, w is the vector of leaf weights, and γ and λ are penalties for model complexity. By fitting new trees to the first and second order gradients of the loss function we can learn thresholds and interactions (e.g., high memory and long GC pauses).

3.4. Decision Layer

The classifier provides failure probabilities p. An alarm is issued if $p \geq \tau$. By default, a threshold of $\tau = 0.5$ is used which does not usually produce the best results for a minority class problem. In order to optimize the F1 score of the out-of-fold training predictions the threshold is chosen by the decision layer. This means that the threshold will never be applied to the test set. The threshold can then be adjusted in order to find the right balance between false negatives and false positives.

3.5. Explanation Layer

This layer provides an explanation to understand which features drive the model and why a particular node has been highlighted. SHAP defines a value φ_i for each feature i in the model that represents the contribution of that feature to a particular prediction. This is equivalent to the Shapley value [10], [27]. As shown in Equation (2):

$$\varphi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(M-|S|-1)!}{M!} [v(S \cup \{i\}) - v(S)] \tag{2}$$

where F is the set of all M features, S is a subset of features that excludes feature i, and $v(S)$ is the expected value of the model output (log-odds) given knowledge of the features in S. Summing the contributions gives the exact difference between the prediction and mean prediction. This produces additive explanations that are interpretable. To get a global feature importance ranking the absolute value of φ_i can be averaged over many instances.

Permutation importance is calculated by measuring the decrease in the test ROC-AUC s due to randomly permuting the values of a feature j and averaging over R instances [11], [12]. This can be expressed as:

$$PI_j = s - \frac{1}{R}\sum_{r=1}^{R} s_{r,j} \tag{3}$$

where s is the test ROC-AUC of the fitted model and $s_{r,j}$ is the ROC-AUC obtained after the r-th random permutation of feature j.

As permutation importance measures the dependence of the model on a feature to provide accurate predictions, and SHAP values break down the model predictions, the ranking correlation between them is reported as Spearman.

4. WORKING OF THE FRAMEWORK

4.1. Simulated Telemetry Dataset

The data consist of 12,000 samples of snapshots where each snapshot has the 10 features in Table 2. Realistic correlations between the resources were achieved using a common latent factor $L \sim N(0,1)$. CPU utilization was sampled using the formula $(55 + 16(0.6L + 0.8\varepsilon))$ where ε is standard normal noise. Similarly, memory utilization was sampled using $(60 + 15(0.5L + 0.87\varepsilon))$. All samples were truncated to ensure they were within the correct ranges. For the remaining resources, disk I/O wait times were Gamma distributed, network latency and error rate were Log-Normal distributed, Node Temperature increased as CPU load increased, number of active threads followed a Poisson distribution which increased as request rate increased, and GC Pause Time increased as memory utilization exceeded 60%.

Failures were simulated using seven failure mechanisms. These are shown in Table 3 and represent the types of failure seen in production clusters. Each mechanism contributes to the log odds of a failure occurring. The logistic function $\sigma(\cdot)$ is used. P_k is the k-th percentile of the feature. Memory, CPU and Uptime have small linear components to represent slow degradation. Gaussian noise ($\sigma = 0.35$) is added to account for hidden factors, and the label is sampled from a Bernoulli distribution with probability equal to the calculated probability. An intercept of -2.0 was used to achieve approximately 24% failure rate.

Table 3. Failure mechanisms embedded in the simulated telemetry

No.	Mechanism	Condition (contribution to log-odds)	Features
M1	Memory exhaustion (OOM)	$2.8 \cdot \sigma((\text{mem} - 85)/2)$	F2
M2	Memory pressure with GC thrashing	$1.8 \cdot \sigma((\text{mem} - 72)/3) \cdot \sigma((z_{gc} - 0.8)/0.3)$	F2, F9
M3	CPU saturation with thermal throttling	$2.2 \cdot \sigma((\text{cpu} - 82)/2.5) \cdot \sigma((\text{temp} - 55)/2)$	F1, F7
M4	Joint resource pressure	1.5 if cpu > 70% and mem > 70%	F1, F2
M5	I/O contention under load	1.3 if disk > P90 and requests > P75	F3, F6
M6	Error burst	1.1 if error rate > P95	F5
M7	Network-partition signature	1.0 if latency > P90 and requests < P25	F4, F6

In order to represent sensor dropouts, 2% of the data was randomly deleted for disk I/O wait, network latency, node temperature and GC pause time. This left 947 missing values in total. The resulting data has 2,831 failures (23.6%). With a stratified sample of 80:20, 9,600 observations were put into the training set and 2,400 within the test set with 566 failures.

Two further scenarios were generated for sensitivity analysis: a smooth scenario, in which risk rises gradually with memory and CPU utilization, and a threshold scenario, in which risk jumps once memory passes about 82% or CPU about 85%.

4.2. Experimental Protocol

The protocol treats all three models identically and uses the test set exactly once.

1. Generate the dataset and split it into stratified training (80%) and test (20%) sets with seed 42.
2. Build a pipeline per model: median imputation, standard scaling (Logistic Regression only) and the classifier.
3. Tune each pipeline by random search with 15 sampled configurations and five-fold stratified cross-validation on the training set, maximising ROC-AUC [29].

4. Produce five-fold out-of-fold probability predictions on the training set with the tuned configuration, and select the decision threshold τ that maximises F1 over a grid from 0.10 to 0.90.
5. Refit the tuned pipeline on the full training set, predict the test set, and apply τ.
6. Compute accuracy, precision, recall, F1-score, ROC-AUC and the confusion matrix on the test set.
7. Explain the XGBoost model with SHAP on 300 test instances and with permutation importance (20 repeats) on the full test set, and compute the Spearman correlation between the two rankings.

4.3. Hyperparameter Search

Table 4 lists the search space of each model and the configuration selected by cross-validation. Each model received the same budget of 15 configurations × 5 folds = 75 fits.

Table 4. Hyperparameter search spaces and selected configurations

Model	Search space	Selected configuration
Logistic Regression	$C \in [10^{-3}, 10^{2}]$ (log-uniform); penalty ∈ {L1, L2}	C = 0.144, L1 penalty
Random Forest	Trees ∈ {200, 400, 600}; max depth ∈ {8, 12, 16, 24, none}; min samples per leaf ∈ {1, 2, 4, 8}; max features ∈ {√d, 0.5, 0.7}	200 trees, depth 8, 8 samples per leaf, max features 0.7
XGBoost	Boosting rounds ∈ {200, 400, 600}; learning rate ∈ {0.02, 0.03, 0.05, 0.08, 0.1}; max depth ∈ {3, 4, 5, 6, none}; max leaves ∈ {15, 31, 63}; min samples per leaf ∈ {10, 20, 40}; L2 regularisation $\lambda \in \{0, 0.5, 1, 2, 5\}$	200 rounds, learning rate 0.02, 15 leaves, 40 samples per leaf, $\lambda = 2$

4.4. Explanation Computation

The interventional Shapley values were calculated with high precision in log-odds space by systematically evaluating all $2^{10} = 1{,}024$ possible feature coalitions against 40 background training examples. This approach circumvents the approximation error associated with KernelSHAP. The sum of the calculated interventional Shapley values exhibited accuracy within 4.0×10^{-15} for each respective example.

4.5. Evaluation Metrics

The evaluation metrics are formally defined in Equations (4) through (6), wherein TP represents true positives, TN represents true negatives, FP represents false positives, and FN represents false negatives:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \tag{4}$$

$$\text{Precision} = \frac{TP}{TP+FP}, \qquad \text{Recall} = \frac{TP}{TP+FN} \tag{5}$$

$$F_1 = \frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}} \tag{6}$$

The ROC-AUC metric provides a measure of quality in ranking that is independent of τ. Recall is the most important metric where missing failures incurs more cost than a false alarm.

4.6. Implementation Details

The framework was constructed utilizing Python in conjunction with the scikit-learn package [30]. Furthermore, NumPy, pandas, and Matplotlib were employed for supplementary functionality. The Gradient Boosting model was trained through the histogram-based gradient

boosting classifier available within scikit-learn, which implements an equivalent second-order histogram split L2 regularization methodology comparable to XGBoost [9], [24]. The script contains automatic import functionality for the XGboost and shap packages when such packages are accessible. All experimental procedures were executed with seed value 42.

5. RESULTS

5.1. Predictive Performance

Table 5 shows the performance of each model on the test set with their optimal threshold applied based upon F1 score. The XGBoost model performs best in terms of F1 score, recall and ROC-AUC. Random Forest performs just better in terms of accuracy and precision. However, Logistic Regression performs worst overall.

Table 5. Test-set performance (n = 2,400; 566 failures) with tuned thresholds

Model	Threshold τ	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	ROC-AUC
Logistic Regression	0.39	82.67	66.74	52.83	58.97	0.804
Random Forest	0.35	84.08	72.22	52.83	61.02	0.809
XGBoost (proposed)	0.34	84.00	71.16	54.06	61.45	0.810

Cross-validation on the training set (Table 6) confirms the ranking.

Table 6. Five-fold cross-validation on the training set (n = 9,600)

Model	CV ROC-AUC (mean ± SD)	OOF Accuracy (%)	OOF Precision (%)	OOF Recall (%)	OOF F1-score (%)
Logistic Regression	0.788 ± 0.012	81.34	61.98	54.13	57.79
Random Forest	0.795 ± 0.009	82.90	67.85	52.27	59.05
XGBoost (proposed)	0.796 ± 0.009	82.70	66.67	53.33	59.26

Figure 2 plots the ROC curves of the three tuned models on the test set, and Figure 3 presents the confusion matrix of XGBoost at its selected threshold τ = 0.34.

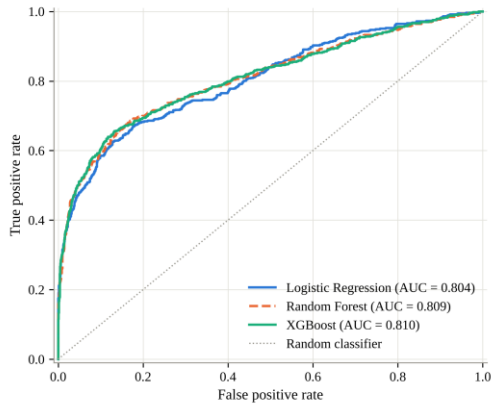


Figure 2. ROC curves of the three models on the test set (n = 2,400)

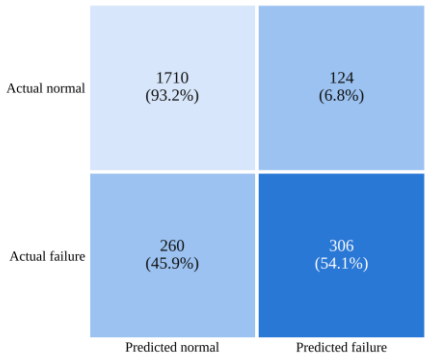


Figure 3. Confusion matrix of the XGBoost model at τ = 0.34

In Figure 2, XGBoost and Random Forest lie above Logistic Regression between false-positive rates of 0.05 and 0.45, the operating range of an alarm system, because they capture interaction-driven failures that linear boundaries cannot separate. Figure 3 shows that XGBoost

detects 306 of the 566 failures with 124 false alarms among 1,834 normal snapshots (a false-positive rate of 6.8%).

5.2. Effect of Threshold Selection

At the default threshold $\tau = 0.5$, XGBoost reached 84.50% accuracy and 81.29% precision but detected only 44.52% of failures (F1-score 57.53%). The F1-optimal threshold raised recall by 9.5 percentage points and F1-score by 3.9 points at a cost of 10.1 points of precision, a larger gain than any change of model.

5.3. Explanation Analysis

Both methods rank memory utilization first and CPU utilization second by a wide margin (Table 7), with a Spearman rank correlation of 0.78 between the two rankings.

Table 7. Global feature importance of the XGBoost model

Feature	Mean |SHAP| (log-odds)	SHAP share (%)	SHAP rank	Permutation ROC-AUC drop	Permutation rank
Memory utilization	0.700	43.3	1	0.1437	1
CPU utilization	0.329	20.4	2	0.0498	2
GC pause time	0.166	10.2	3	0.0060	4
Uptime	0.114	7.1	4	0.0012	8
Request rate	0.086	5.3	5	0.0041	6
Disk I/O wait	0.061	3.8	6	0.0045	5
Error rate	0.051	3.2	7	0.0133	3
Node temperature	0.051	3.2	8	0.0031	7
Network latency	0.045	2.8	9	0.0001	9
Active threads	0.013	0.8	10	−0.0001	10

These rankings match the simulation design: memory enters three mechanisms (M1, M2, M4) and CPU two (M3, M4), GC pause time ranks third under SHAP through M2, and active threads, which enter no mechanism, rank last under both methods. The algorithms differ in how they handle the error rate. Permuting this feature causes an increase in accuracy as it has a rare but drastic effect (M6). The uptime has a small constant effect which is spread across all instances using SHAP.

Figure 4 shows the SHAP summary plot for 300 test instances; each point is one instance, placed by its contribution to the log-odds of failure and coloured by feature value.

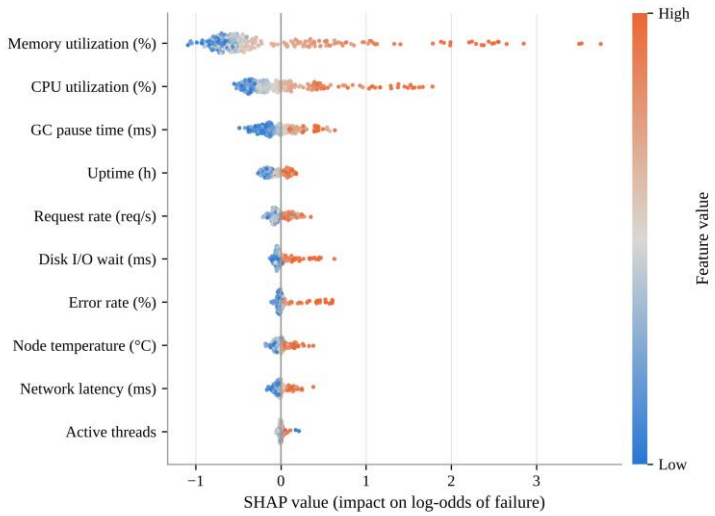


Figure 4. SHAP summary (beeswarm) plot for 300 test instances

High memory and CPU values push predictions strongly towards failure, by up to about +3.8 log-odds for memory, while low values reduce risk only modestly.

Figure 5 plots the SHAP value for memory against memory utilization, coloured by CPU utilization, and Figure 6 shows the permutation importance of each feature over 20 repeats.

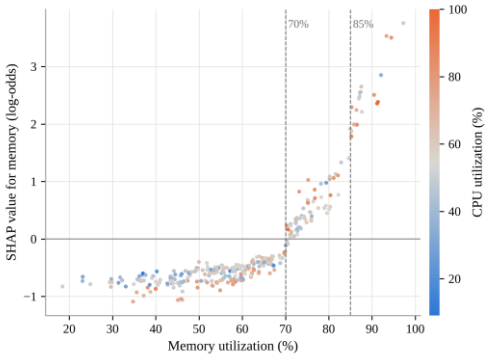


Figure 5. SHAP dependence plot for memory utilization, coloured by CPU utilization

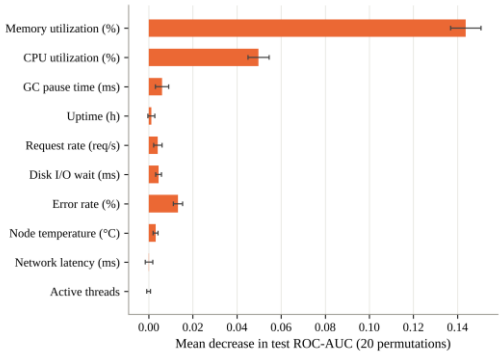


Figure 6. Permutation importance of the XGBoost model (mean ± 1 SD, 20 repeats)

The memory contribution in Figure 5 is flat below about 65%, crosses zero near 70% and rises steeply beyond 80%, matching the thresholds in mechanisms M4 and M1. In Figure 6, the intervals for memory and CPU lie far from those of the other features, so the top-two ranking is robust.

5.4. Sensitivity to the Failure-Generation Scenario

Table 8 applies the protocol to two alternative scenarios. XGBoost achieves the highest cross-validated ROC-AUC in all three, but when risk rises gradually Logistic Regression is within 0.001 of the best test ROC-AUC, so non-linear models gain an advantage only when failures are driven by thresholds and interactions.

Table 8. Results across failure-generation scenarios (tuned models, test set)

Scenario	Model	CV ROC-AUC	Test ROC-AUC	Test F1-score (%)	Test accuracy (%)
Interaction (main)	Logistic Regression	0.788	0.804	58.97	82.67

Scenario	Model	CV ROC-AUC	Test ROC-AUC	Test F1-score (%)	Test accuracy (%)
Interaction (main)	Random Forest	0.795	0.809	61.02	84.08
Interaction (main)	XGBoost	0.796	0.810	61.45	84.00
Smooth	Logistic Regression	0.923	0.932	77.33	87.42
Smooth	Random Forest	0.922	0.933	77.90	87.38
Smooth	XGBoost	0.926	0.932	78.51	87.96
Threshold	Logistic Regression	0.847	0.827	62.04	81.54
Threshold	Random Forest	0.854	0.834	65.18	83.08
Threshold	XGBoost	0.858	0.835	64.70	83.54

6. DISCUSSION

The findings show that gradient-boosted trees are effective for interaction-driven failures, that the explanation layer identifies the genuine failure drivers, and that the decision threshold matters more to operators than the choice among good models. The mechanisms in Table 3 are conjunctions and steps, which boosted trees express through successive splits [9], [23], whereas a linear model only adds feature effects; Logistic Regression therefore matched the tree ensembles only when risk rose gradually (Table 8), while the gap between XGBoost and Random Forest stayed within one cross-validation standard deviation. Because the true mechanisms are known, the explanations could be validated: both methods ranked memory and CPU highest and thread count last, Figure 5 reproduced the 70% and 85% thresholds, and their small disagreements on rare and widespread effects show the value of reporting both [10], [12]. For operators, these attributions turn an alarm into an action, and moving τ from 0.5 to its F1-optimal value raised recall from 44.5% to 54.1%. The main limitations are the simulated data, which call for validation on production traces such as the Google cluster data [2], the absence of temporal features [21], [19], and the use of scikit-learn's histogram gradient boosting instead of the xgboost library, which may alter exact figures slightly.

7. CONCLUSION AND FUTURE WORK

This research introduced XML-PRFD, an explainable machine learning framework for predictive resource failure detection that integrates telemetry preprocessing, tuned classifiers, F1-optimal thresholds and a dual-method explanation layer. On a simulated benchmark of 12,000 snapshots, XGBoost achieved the highest test ROC-AUC (0.810), F1-score (61.45%) and recall (54.06%) under equal tuning budgets, and SHAP and permutation importance consistently identified memory and CPU utilization as the principal failure drivers. Future work will validate the framework on production traces such as the Google cluster data, add temporal features and cost-sensitive thresholds, and assess how explanations influence operator decisions.

REFERENCES

[1] Schroeder, B., & Gibson, G. A. (2010). A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing*, *7*(4), 337-350. https://doi.org/10.1109/TDSC.2009.4

[2] Reiss, C., Tumanov, A., Ganger, G. R., Katz, R. H., & Kozuch, M. A. (2012). Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM*

Symposium on Cloud Computing (pp. 1-13). Association for Computing Machinery. https://doi.org/10.1145/2391229.2391236

[3] Salfner, F., Lenk, M., & Malek, M. (2010). A survey of online failure prediction methods. *ACM Computing Surveys*, *42*(3), Article 10. https://doi.org/10.1145/1670679.1670680

[4] Pinheiro, E., Weber, W.-D., & Barroso, L. A. (2007). Failure trends in a large disk drive population. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST '07)* (pp. 17-28). USENIX Association.

[5] Chen, X., Lu, C.-D., & Pattabiraman, K. (2014). Failure analysis of jobs in compute clouds: A Google cluster case study. In *2014 IEEE 25th International Symposium on Software Reliability Engineering (ISSRE)* (pp. 167-177). IEEE. https://doi.org/10.1109/ISSRE.2014.34

[6] Lin, Q., Hsieh, K., Dang, Y., Zhang, H., Sui, K., Xu, Y., Lou, J.-G., Li, C., Wu, Y., Yao, R., Chintalapati, M., & Zhang, D. (2018). Predicting node failure in cloud service systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 480-490). Association for Computing Machinery. https://doi.org/10.1145/3236024.3236060

[7] Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. (2020). Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, *58*, 82-115. https://doi.org/10.1016/j.inffus.2019.12.012

[8] Gunning, D., & Aha, D. W. (2019). DARPA's explainable artificial intelligence program. *AI Magazine*, *40*(2), 44-58. https://doi.org/10.1609/aimag.v40i2.2850

[9] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785-794). Association for Computing Machinery. https://doi.org/10.1145/2939672.2939785

[10] Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30* (pp. 4765-4774).

[11] Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5-32. https://doi.org/10.1023/A:1010933404324

[12] Fisher, A., Rudin, C., & Dominici, F. (2019). All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *Journal of Machine Learning Research*, *20*(177), 1-81.

[13] Nagarajan, B., & Mahalingam, R. (2026). Deep learning-based intelligent framework for cloud-native 5G core network management and optimization. *International Innovative Research Journal of Engineering and Technology*, *11*(4). https://doi.org/10.32595/iirjet.org/v11i4.2026.250

[14] Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, *41*(3), Article 15. https://doi.org/10.1145/1541880.1541882

[15] Mahdisoltani, F., Stefanovici, I., & Schroeder, B. (2017). Proactive error prediction to improve storage system reliability. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)* (pp. 391-402). USENIX Association.

[16] Xu, Y., Sui, K., Yao, R., Zhang, H., Lin, Q., Dang, Y., Li, P., Jiang, K., Zhang, W., Lou, J.-G., Chintalapati, M., & Zhang, D. (2018). Improving service availability of cloud systems by predicting disk error. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)* (pp. 481-494). USENIX Association.

[17] Rosà, A., Chen, L. Y., & Binder, W. (2015). Predicting and mitigating jobs failures in big data clusters. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (pp. 221-230). IEEE. https://doi.org/10.1109/CCGrid.2015.139

[18] Islam, T., & Manivannan, D. (2017). Predicting application failure in cloud: A machine learning approach. In *2017 IEEE International Conference on Cognitive Computing (ICCC)* (pp. 24-31). IEEE. https://doi.org/10.1109/IEEE.ICCC.2017.11

[19] Das, A., Mueller, F., Siegel, C., & Vishnu, A. (2018). Desh: Deep learning for system health prediction of lead times to failure in HPC. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing* (pp. 40-51). Association for Computing Machinery. https://doi.org/10.1145/3208040.3208051

[20] He, S., Zhu, J., He, P., & Lyu, M. R. (2016). Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)* (pp. 207-218). IEEE. https://doi.org/10.1109/ISSRE.2016.21

[21] Du, M., Li, F., Zheng, G., & 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* (pp. 1285-1298). Association for Computing Machinery. https://doi.org/10.1145/3133956.3134015

[22] Tan, W. M., & Teo, Z. Y. (2025). WiSIDS: A lightweight machine learning-based intrusion detection system for securing wireless IoT networks. *Journal of Wireless Networks and Communication Systems, 1*(3), 1-12. https://doi.org/10.32595/jwncs/v1i3.2025.12

[23] Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, *29*(5), 1189-1232. https://doi.org/10.1214/aos/1013203451

[24] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems 30* (pp. 3146-3154).

[25] Shrestha, U., & Goudar, S. I. (2025). Enhancing predictive accuracy using ensemble machine learning models: A data-centric approach. *Journal of Computer Applications and Information Technology*, *1*(3), 60-72. https://doi.org/10.32595/jcait/v1i3.2025.19

[26] Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1135-1144). Association for Computing Machinery. https://doi.org/10.1145/2939672.2939778

[27] Shapley, L. S. (1953). A value for n-person games. In H. W. Kuhn & A. W. Tucker (Eds.), *Contributions to the Theory of Games* (Vol. II, pp. 307-317). Princeton University Press.

[28] Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., & Lee, S.-I. (2020). From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, *2*(1), 56-67. https://doi.org/10.1038/s42256-019-0138-9

[29] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*(10), 281-305.

[30] 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., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*(85), 2825-2830.