Artificial Intelligence–Driven Predictive Modeling for Early Identification of System Anomalies

Rohit Kumar

Assistant Professor
Department of Computer Science and Engineering
RRSDCE, Begusarai, Bihar, India
Email: rohitspoken19@gmail.com

Abstract

Modern computing infrastructures generate continuous streams of telemetry, yet most operational monitoring still relies on fixed thresholds that raise alarms only after service quality has already degraded. This paper presents an artificial intelligence (AI)–driven predictive framework that identifies system anomalies before they fully develop. The framework converts multivariate telemetry into rolling statistical descriptors, formulates early identification as a supervised prediction task with a fixed look-ahead horizon, and calibrates the alarm threshold on held-out validation data. Four learners were compared under a chronological split: Isolation Forest, logistic regression, random forest, and gradient boosting. Experiments used a reproducible synthetic telemetry benchmark of 29,971 usable time steps containing 55 injected anomaly episodes, each preceded by a gradual precursor drift. The supervised tree ensembles reached F1-scores of 0.950–0.955 and area under the ROC curve above 0.999, whereas the unsupervised Isolation Forest reached an F1-score of 0.705. The gradient boosting model detected all 18 test episodes with a mean lead time of 19.1 time steps before onset. Because the data are simulated, the results demonstrate the design and the relative behaviour of the methods rather than production performance; the complete code is supplied so that the study can be repeated on real telemetry.

Keywords: predictive modeling; anomaly detection; early warning; machine learning; system monitoring; time-series analysis; explainable AI

1. Introduction

Software services, data centres, industrial controllers, and cloud platforms are now instrumented at fine granularity. A single service may emit thousands of measurements per minute, including processor load, memory occupancy, disk and network activity, request latency, and error counts. Deviations in these signals frequently precede outages, performance collapse, or security incidents. Detecting such deviations early gives operators time to reroute traffic, restart components, or add capacity before users are affected (Chandola et al., 2009).

Conventional monitoring compares each metric with a manually chosen limit. This practice has three weaknesses. First, limits are static, whereas normal behaviour follows daily and weekly cycles, so a value that is alarming at night may be ordinary at noon. Second, univariate rules ignore interactions; a modest rise in latency combined with a slow growth in memory use can be far more informative than either change alone. Third, threshold rules are reactive by design: they fire when a boundary is crossed, which is often the moment the fault becomes visible to users. Machine learning offers a way to address all three limitations by learning normal behaviour from data, modelling relationships between signals, and estimating the probability that a failure is imminent (Pang et al., 2020).

A distinction that is often blurred in the literature is between detecting an anomaly and predicting one. Detection labels the present observation; prediction labels the near future. For operations, the second formulation is more valuable because its output is time to act. Yet many published studies still evaluate models only on point-wise detection and do not report how far ahead of the fault the alarm was raised.

This paper addresses that gap with three contributions. (i) It formulates early identification of system anomalies as a horizon-based supervised prediction problem and defines a leakage-free evaluation protocol based on chronological splitting. (ii) It proposes a compact and interpretable pipeline built on rolling statistical features and tree-ensemble learners, with the alarm threshold selected on validation data rather than fixed at 0.5. (iii) It reports, in addition to conventional classification metrics, the proportion of anomaly episodes that were caught and the lead time gained, and it releases the full Python implementation to support replication.

The remainder of the paper is organised as follows. Section 2 reviews related work, Section 3 describes the data, features, models, and evaluation protocol, Section 4 presents and discusses results, Section 5 states limitations, and Section 6 concludes.

2. Related Work

2.1 Statistical and unsupervised approaches

Early anomaly detection research relied on distance, density, and statistical models. Density-based methods such as the local outlier factor score an observation by comparing its neighbourhood density with that of its neighbours (Breunig et al., 2000). Isolation Forest takes a different route: anomalous points are easier to separate by random axis-parallel splits and therefore have shorter average path lengths in random trees (Liu et al., 2008). Its linear time complexity and absence of a distance computation make it popular for high-volume telemetry. The comprehensive survey by Chandola et al. (2009) organises these and other techniques by data type and application domain and remains a standard reference. Unsupervised methods do not require labels, but they detect what is unusual rather than what is harmful, and they typically respond only once the deviation is already large.

2.2 Deep learning for sequences and logs

Recurrent networks with long short-term memory cells (Hochreiter & Schmidhuber, 1997) have been applied to time-series anomaly detection by learning to predict the next values and flagging large prediction errors (Malhotra et al., 2015). Hundman et al. (2018) extended this idea to spacecraft telemetry with a nonparametric dynamic threshold that adapts to local variability. For textual system logs, Du et al. (2017) modelled normal execution patterns as sequences and flagged deviations. Streaming settings have been addressed by online sequence memory approaches that adapt continuously to non-stationary data (Ahmad et al., 2017). A broad review of deep anomaly detection is provided by Pang et al. (2020), who also note the difficulty of obtaining reliable labels and of explaining the resulting decisions.

2.3 Tree ensembles and interpretability

Random forests average many decorrelated decision trees and are robust to noise and irrelevant inputs (Breiman, 2001). Gradient boosting builds trees sequentially so that each corrects the residual errors of its predecessors (Friedman, 2001), and scalable implementations have become dominant on structured tabular data (Chen & Guestrin, 2016). For operations teams, model transparency matters as much as accuracy, because an alert without a reason is difficult to trust. Additive attribution methods such as SHAP quantify the contribution of each feature to an individual prediction (Lundberg & Lee, 2017), and tree ensembles additionally expose global importance scores.

2.4 Research gap

Existing studies concentrate on either unsupervised detection or on deep architectures, and they often report point-wise scores without reference to how early a fault was signalled. Few works combine a horizon-based target, chronological validation, threshold calibration, and lead-time analysis in one reproducible pipeline. The present study is designed to fill this gap.

3. Materials and Methods

3.1 Problem formulation

Let x_t denote the vector of d telemetry measurements at time step t. An anomaly episode is a contiguous interval $[s, e)$ during which the system is in a degraded state. The early-warning target is defined as $y_t = 1$ if t lies in the interval $[s - H, e)$ of any episode and $y_t = 0$ otherwise, where H is the prediction horizon. In this study $H = 15$ time steps. A model therefore receives the recent history of x and outputs the probability that an anomaly is either under way or will begin within H steps. An alarm is raised when this probability reaches a threshold τ.

3.2 Benchmark telemetry

Real production telemetry with reliable fault annotation is rarely shareable, so a controlled synthetic benchmark was generated with a fixed random seed. Six metrics were simulated over 30,000 time steps: CPU utilisation, memory utilisation, request latency, error rate, disk input/output rate, and network throughput. Each metric combined a daily sinusoidal cycle with Gaussian noise to mimic ordinary workload variation. Fifty-five anomaly episodes were then injected, each lasting 20–50 steps and separated by 300–700 normal steps. Every episode was preceded by a precursor phase of 25–45 steps in which the affected metrics drifted gradually before jumping to fault levels. Three fault families were used: resource saturation (memory, CPU, and latency), I/O and network congestion, and an error-rate cascade. After rolling windows were computed, 29,971 usable rows remained, of which 9.19% carried the positive label. Table 1 summarises the variables and the derived features.

Table 1. Telemetry variables and derived features used in the study

Metric	Base behaviour (simulated)	Derived features	Typical role in faults
CPU utilisation (%)	Mean 45, daily cycle ±12	Mean, SD (10, 30); Δ10	Saturation, cascade
Memory utilisation (%)	Mean 55, daily cycle ±4	Mean, SD (10, 30); Δ10	Leak / saturation
Latency (ms)	Mean 120, daily cycle ±15	Mean, SD (10, 30); Δ10	Present in all families
Error rate (per s)	Mean 0.5, noise 0.15	Mean, SD (10, 30); Δ10	Error cascade
Disk I/O (MB/s)	Mean 200, daily cycle ±30	Mean, SD (10, 30); Δ10	I/O congestion
Network (Mbps)	Mean 500, daily cycle ±60	Mean, SD (10, 30); Δ10	Congestion

Note. SD = standard deviation over the stated window; Δ10 = current value minus value ten steps earlier. Each metric yields six derived features plus the raw value, giving 36 inputs.

3.3 Feature engineering

For every metric, the raw value, the rolling mean and rolling standard deviation over windows of 10 and 30 steps, and the ten-step difference were computed. The rolling mean smooths noise, the rolling deviation captures growing instability, and the difference approximates the local slope, which is the quantity that exposes a precursor drift. All features use only past and current

observations, so no future information enters the input. Features were standardised using statistics computed on the training portion alone.

3.4 Predictive models

Four models were implemented in scikit-learn (Pedregosa et al., 2011). Isolation Forest (200 trees) was trained only on rows labelled normal, and its anomaly score served as the alarm signal; this represents the unsupervised baseline (Liu et al., 2008). Logistic regression with balanced class weights served as a linear supervised baseline. A random forest of 300 trees with balanced class weights (Breiman, 2001) and a gradient boosting classifier with 250 trees of depth 3 and a learning rate of 0.08 (Friedman, 2001) were the two ensemble learners. Default settings were kept for remaining hyperparameters so that no model benefited from extensive tuning.

3.5 Validation protocol and threshold calibration

Data were split chronologically: the first 70% of rows (20,979) formed the training set and the last 30% (8,992) the test set. Random shuffling was avoided because adjacent time steps are strongly correlated and shuffling would leak information across the split. Within the training set, the final 20% was held out as validation data on which the alarm threshold τ was chosen to maximise the F1-score; the model was then refitted on the complete training set and applied once to the test set.

3.6 Evaluation metrics

Precision, recall, and F1-score were computed at the calibrated threshold. Because positive rows are a small minority, the area under the ROC curve (AUC) was complemented with average precision (AP), which summarises the precision–recall curve and is more sensitive to false alarms under class imbalance (Saito & Rehmsmeier, 2015). Episode-level behaviour was assessed by the number of test episodes for which at least one alarm was raised in the 40 steps

before onset and by the lead time, defined as the number of steps between the first such alarm and the onset of the fault. Figure 1 shows the complete pipeline.

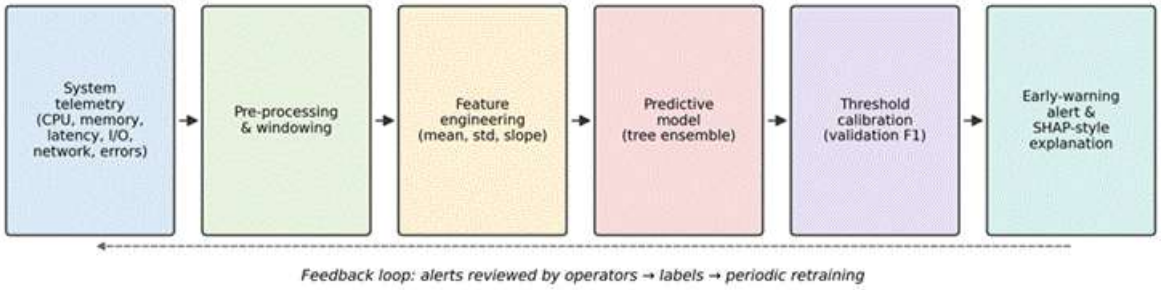


Figure 1. Architecture of the proposed AI-driven early-warning pipeline

4. Results and Discussion

4.1 Classification performance

Table 2 lists the test-set results. All three supervised models performed strongly and within a narrow band. The random forest achieved the highest F1-score (0.955) and AP (0.996), gradient boosting followed closely (F1 = 0.950), and logistic regression, although linear, reached an F1-score of 0.943. Recall was close to 0.99 for each supervised learner, so almost every row inside a warning window was flagged; the differences between them arose mainly from precision, that is, from the number of false alarms. The gap among the three is small enough that it should not be over-interpreted, particularly because only one simulated dataset and one random seed were used.

Table 2. Test-set performance of the compared models (chronological hold-out)

Model	Precision	Recall	F1-score	ROC AUC	Avg. precision
Isolation Forest	0.558	0.959	0.705	0.965	0.577

Logistic Regression	0.894	0.996	0.943	0.999	0.995
Random Forest	0.916	0.996	0.955	1.000	0.996
Gradient Boosting	0.911	0.993	0.950	0.999	0.994

Note. Thresholds were calibrated on validation data. Values are from a single run with a fixed random seed (42).

The unsupervised Isolation Forest was clearly weaker, with an F1-score of 0.705 and an average precision of 0.577. Its recall was high (0.959) but its precision was only 0.558, meaning that nearly half of its alarms were false. This behaviour is expected: the method scores how unusual a point is and has no notion of which unusual patterns lead to failures. The daily workload cycle also produces mild outliers that resemble the early stage of a precursor. The result supports the argument that labelled history, when available, should be exploited, while unsupervised scores remain useful for new systems that lack labels.

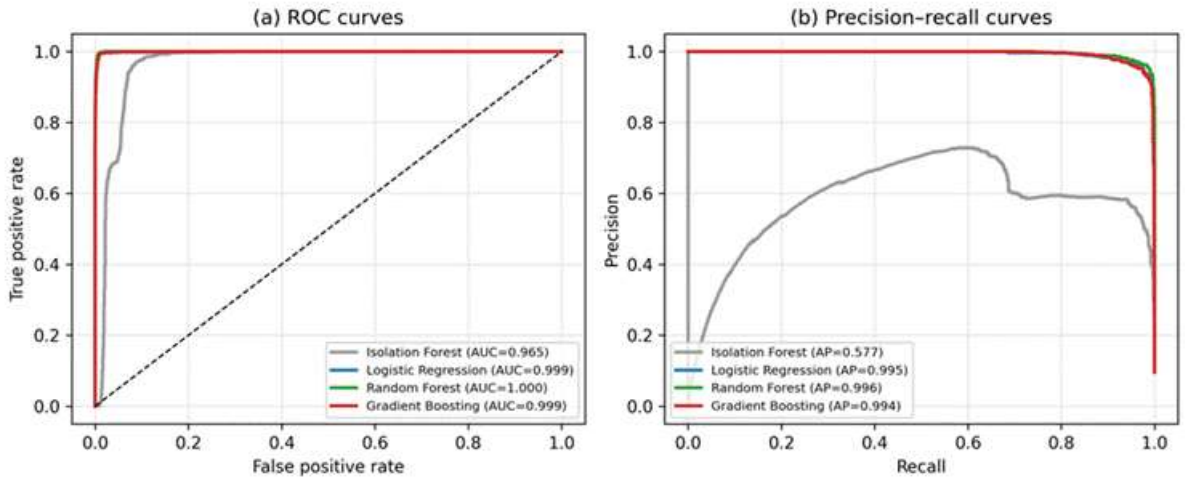


Figure 2. (a) ROC and (b) precision–recall curves on the test set (generated with Python/Matplotlib).

Figure 2 confirms these findings. In panel (a), the three supervised curves lie almost on top of one another near the upper-left corner, whereas the Isolation Forest curve is visibly lower.

Panel (b) separates the methods more clearly: precision of the supervised models remains high over almost the whole recall range, but the unsupervised baseline declines quickly. This illustrates why precision–recall analysis is recommended for imbalanced monitoring data, since ROC curves can appear favourable even when many false alarms are generated (Saito & Rehmsmeier, 2015).

4.2 Early-warning behaviour

For the gradient boosting model, 18 anomaly episodes occurred in the test period and an alarm was raised before onset for all 18 (100% episode recall). The mean lead time was 19.1 time steps and the median 19.5, which exceeds the training horizon of 15 steps because the precursor drift itself begins earlier than the label window and the model responds as soon as the drift is distinguishable from noise. In practical terms, if one step represents one minute, operators would receive roughly 19 minutes of notice on average, which is sufficient for many automated remediation actions such as scaling out or draining a node. A threshold rule applied only to fault-level values would by construction give no such lead.

4.3 Which signals matter

The impurity-based importance of the gradient boosting model was dominated by two inputs: the raw error rate (0.554) and raw latency (0.275), followed by memory measures. This is consistent with the simulation design, in which the error-rate cascade and the latency increase are the most consistent indicators across the three fault families, and it shows that the model uses signals in a sensible way. Impurity importance can favour high-variance features, so in real deployments it should be verified with permutation importance or SHAP values (Lundberg & Lee, 2017) before being presented to operators.

4.4 Practical implications

Three observations are relevant for engineers. First, a compact model on engineered rolling features is competitive with far more elaborate approaches on this task, trains in seconds on a CPU, and can score a new observation in microseconds, which suits edge and embedded settings. Second, calibrating the threshold on validation data is essential; it converts the same score into a chosen balance between missed faults and false alarms. Third, the pipeline should be retrained periodically, because workload patterns and software versions change; the feedback loop shown in Figure 1 supports this by turning reviewed alerts into new labels.

5. Limitations and Future Work

The principal limitation is that all data are simulated. Synthetic precursors are smoother and more consistent than those seen in production, so the near-perfect AUC values should not be interpreted as expected field performance; real systems contain concept drift, missing values, correlated failures, and ambiguous labels. Results come from one dataset and one seed, and no confidence intervals are reported; repeated runs over multiple seeds and blocked time-series cross-validation would strengthen the conclusions. The comparison also excluded sequence models such as LSTM networks and transformer-based detectors, which may perform better on long dependencies. Future work will apply the pipeline to public and industrial telemetry, add drift-adaptive retraining, evaluate cost-sensitive thresholds that weigh missed faults against alert fatigue, and integrate SHAP-based explanations into the alert message.

6. Conclusion

This study presented an AI-driven predictive framework for early identification of system anomalies that combines rolling statistical features, tree-ensemble learning, horizon-based labelling, and validation-based threshold calibration. On a reproducible synthetic benchmark, supervised ensembles reached F1-scores of about 0.95 and detected every test episode with an average lead time of roughly 19 steps, whereas an unsupervised baseline produced many more

false alarms. The findings indicate that predictive rather than purely reactive modelling can materially extend the warning time available to operators. Validation on real telemetry remains the necessary next step, and the released code is intended to make that step straightforward.

References

[1] Ahmad, S., Lavin, A., Purdy, S., & Agha, Z. (2017). Unsupervised real-time anomaly detection for streaming data. Neurocomputing, 262, 134–147.

[2] Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32.

[3] Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). LOF: Identifying density-based local outliers. ACM SIGMOD Record, 29(2), 93–104.

[4] Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM Computing Surveys, 41(3), 1–58.

[5] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785–794.

[6] Du, M., Li, F., Zheng, G., & Srikumar, V. (2017). DeepLog: Anomaly detection and diagnosis from system logs through deep learning. Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 1285–1298.

[7] Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. The Annals of Statistics, 29(5), 1189–1232.

[8] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780.

[9] Hundman, K., Constantinou, V., Laporte, C., Colwell, I., & Soderstrom, T. (2018). Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding.

Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 387–395.

[10] Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation forest. Proceedings of the 8th IEEE International Conference on Data Mining, 413–422.

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

[12] Malhotra, P., Vig, L., Shroff, G., & Agarwal, P. (2015). Long short term memory networks for anomaly detection in time series. Proceedings of the 23rd European Symposium on Artificial Neural Networks (ESANN), 89–94.

[13] Pang, G., Shen, C., Cao, L., & van den Hengel, A. (2020). Deep learning for anomaly detection: A review. ACM Computing Surveys, 54(2), 1–38.

[14] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825–2830.

[15] Saito, T., & Rehmsmeier, M. (2015). The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. PLOS ONE, 10(3), e0118432.