LIGHTWEIGHT ML-BASED SYSTEM CLASSIFICATION FOR RESOURCE-CONSTRAINED MONITORING ENVIRONMENTS

Jeremy D. Howard

Independent Scholar, Huntsville, Alabama, USA

ABSTRACT

Modern system monitoring in distributed and resource-constrained environments is limited by reduced observability, high-overhead instrumentation, and reliance on rule-based or computationally intensive machine learning models. This study presents a lightweight machine learning pipeline for system-state classification using structured synthetic telemetry data, enabling non-intrusive deployment without reliance on sensitive operational inputs. Logistic Regression achieved an accuracy of 0.97, with precision of 0.973, recall of 0.947, and F1-score of 0.959, outperforming a Random Forest model (F1 = 0.909). Five-fold cross-validation yielded a mean F1-score of 0.980 (±0.012), indicating stable performance across data partitions. A rule-based baseline failed to detect degraded states (F1 = 0.0), highlighting the limitations of static threshold-based monitoring. Average inference time was approximately 0.006 ms per sample under local conditions. These findings demonstrate that low-complexity, interpretable models can provide reliable and efficient system-state classification in constrained environments.

KEYWORDS

Lightweight Machine Learning, System Classification, & Resource-Constrained Environment

1. INTRODUCTION

The technical systems of modernity present a monitoring challenge that was not seen in older single-server environments. Cloud platforms, distributed applications, microservices, IoT devices, and edge nodes all create large amounts of performance data while also increasing operational complexity. Several recent papers have noted that telemetry has expanded faster than many organizations' ability to turn it into useful decisions [1][2][3]. Operators commonly monitor CPU load, memory use, latency, packet loss, counters, and service errors. These metrics are useful, but many organizations still rely on static rules such as "alert if CPU exceeds X" or "alert if latency exceeds Y." That logic is easy to implement, but it can miss broader patterns or trigger nuisance alerts. Research in modern distributed environments also notes that maintaining situational awareness becomes more difficult as systems scale and interact [4]. Bearing this in mind, it becomes apparent that an operational framework designed with time and computation in mind could be beneficial. Is Machine Learning (ML) the answer to the issue?

ML does offer a practical alternative. Instead of checking one metric at a time, a model can learn relationships across several variables and potentially telemetry data simultaneously. Thus, providing an effective approach to capturing system performance. Prior studies in anomaly detection and IoT security show that ML can improve detection performance when compared with traditional approaches [5][6][7]. In opposition, many published solutions depend on larger digital form factors that have heavier architectures, are computationally resource-intensive, require extensive cloud resources, deep learning pipelines, or specialized datasets. This creates a

DOI: 10.5121/ijnlc.2026.15203

friction point for environments that need simple, explainable, low-overhead tools. The viability of these concerns is further supported by studies in both TinyML and Edge AI that repeatedly show that real deployment constraints often matter as much as algorithmic sophistication [8][9]. Another issue is access to real telemetry data. Security restrictions, privacy concerns, proprietary systems, or heavily restricted enclave environments can limit dataset availability. Synthetic data research suggests that simulated datasets can support model development when used carefully and with transparency [10][11][12][13].

This paper evaluates a lightweight ML pipeline designed for low-computation monitoring environments. The study evaluated whether the model could classify system state reliably using structured telemetry inputs. Logistic Regression, Random Forest, and a simple threshold baseline were evaluated using standard classification metrics, cross-validation, and runtime efficiency measurement.

2. RELATED WORK

The literature relevant to this study spans five interrelated domains: telemetry-based monitoring and observability, ML and anomaly detection, resource-constrained ML, model selection and baseline performance, and synthetic data for ML applications.

2.1. Telemetry-Based Monitoring and Observability

Telemetry has become a core mechanism for understanding performance in modern systems. Prior work emphasized the value of CPU, memory, network, and service-level metrics for improving resource allocation, detecting bottlenecks, and supporting operational decision-making [1][2][3]. Additionally, telemetry-driven AI was described as an emerging method for converting raw operational signals into predictive and automated workflows [4]. Despite these benefits, telemetry volume alone does not guarantee actionable insight. Large-scale and distributed systems may generate more signals than human interface operators can efficiently interpret; as well as the level and depth of competency and understanding of performance metrics presents a concern for large scale deployment of this tactic in performance monitoring. When taken together, these concepts suggest the need for an automated classification and prioritization solution.

2.2. Machine Learning for Anomaly Detection

A significant body of literature has demonstrated the application of ML in anomaly detection, intrusion detection, and irregular behavior identification. Considering the large-scale proliferation of ML and AI, this level of implementation is of no surprise. Multi-layer IoT-edge-cloud architectures have used ML to improve detection speed and resilience, while security-focused reviews document strong performance across varied attacks and anomaly scenarios [5][6]. Similarly, edge-based anomaly detection research demonstrated the practicality of training multiple candidate classifiers and selecting the strongest performer for deployment [7]. While anomalies can lead toward degradation of system performance anomaly detection is not identical to system-state classification. Many anomaly studies focus on rare-event discovery rather than direct nominal-versus-degraded classification. The possession of near-real time system status awareness is a vital operational need for many professional settings and environments.

2.3. Resource Constrained Machine Learning and Edge AI

TinyML and Edge AI literature has provided critical insight into deployment constraints. Specifically, the need for models that balance accuracy with computational efficiency; as well as consumption of both available memory and energy. Memory limits, energy budgets, local inference, and low latency frequently constrain deployment choices for embedded systems [8][9]. Further, trade-offs between model complexity, latency, and deployment feasibility are a concern for edge devices [9]. These studies collectively argue that evaluation of ML models must extend beyond predictive performance to include runtime efficiency and resource utilization. This perspective directly informs the design of lightweight monitoring systems where computational overhead must remain minimal.

2.4. Simpler Model Selection and Baseline Performance

Baseline model evaluation has been highlighted as being vital [14]. Several authors have cautioned against drawing the assumption that a more complex model will outperform simpler alternatives. Baseline-focused work argues that rigorous comparison against simple methods is essential before claiming meaningful improvement [14]. Comparative healthcare prediction studies similarly show that Logistic Regression can remain highly competitive relative to more complex ML approaches [15]. These findings challenge the assumption that model complexity inherently improves performance and support the inclusion of lightweight models as viable solutions in constrained environments.

2.5. Synthetic Data for ML Model Development

Synthetic data has emerged as a practical solution for addressing limitations in data accessibility, specifically as a proposed response to privacy, scarcity, and sharing constraints. Synthetic datasets can enable model development while preserving privacy and security constraints. Reviews have noted that synthetic datasets can preserve utility while reducing exposure of sensitive records, though representativeness and fairness must be evaluated carefully [10][11][13]. However, these works also stress the importance of evaluating synthetic data for utility, realism, and bias. In classification tasks, reliance on synthetic data necessitates robust evaluation practices, including multiple performance metrics and validation strategies, to ensure that results are meaningful and not artifacts of the data generation process.

2.6. Identified Gap

Across these domains, a consistent gap emerges. While telemetry provides necessary system insight; demonstration of the implementation of a lightweight ML approach attractiveness and fit for constrained environments due to favorable compute and memory requirements has not been thoroughly tested. However, authors repeatedly identify concerns involving external validity, dataset realism, class imbalance, explainability, and real-world scalability. Existing research often prioritizes anomaly detection, complex architectures, or high-resource environments, leaving a gap for practical, low-overhead solutions that can be deployed in edge-based or restricted systems. Thus, assessment of a reproducible pipeline for supervised system-state classification that operates effectively under constrained conditions such as an isolated secure enclave could bring viability to modern operational systems. This study addresses that gap by integrating telemetry-based feature construction, synthetic data generation, and baseline ML models into a unified framework designed for efficiency, interpretability, and operational viability.

3. PROBLEM FORMULATION

System monitoring can be framed through the lens of a supervised binary classification problem in which telemetry observations are used to infer operational state of a system. In this study, the objective was to classify system behaviors into one of two categories:

- **0 = Normal**, representing nominal operating conditions.
- **1= Degraded**, representing impaired or unstable performance conditions

Each observation was represented by a structured feature vector composed of five telemetry variables:

- CPU_USAGE
- MEMORY_USAGE
- LATENCY_MS
- ERROR_RATE
- PACKET_LOSS

These variables were selected due to their practical relevance in system health monitoring and their repeated appearance in telemetry-based performance literature [1][2].

Formally, the classification task can be expressed as learning a mapping:

Equation 1. The Mapping Equation

$$f : X \rightarrow Y$$

The Variables for the Mapping Equation represent:

- $X \in R^5$ represents the telemetry feature vector composed of cpu_usage, memory_usage, latency_ms, error_rate, and packet_loss.
- $Y \in \{0,1\}$ represents the binary system-state label, where 0 indicates normal operation and 1 indicates a degraded system state.

This formulation defines the supervised learning objective of mapping system telemetry inputs to operational state outputs. Traditional indicators of degradation conditions include CPU utilization or excessive latency as threshold logic and metrics. However, prior literature suggests that operational degradation is often multi-factorial and may emerge through interactions among variables rather than isolated threshold violations [5][6].

To evaluate whether lightweight ML can improve state awareness, three comparative approaches were assessed:

- Logistic Regression
- Random Forest
- Rule-Based Threshold Baseline

The central hypothesis of this study was that a lightweight interpretable model could outperform static threshold logic while maintaining lower complexity than a higher-order ensemble method.

4. DATA GENERATION PROCESS

4.1. Dataset Construction

Due to limited access to real-world constrained enclave operational telemetry datasets, a synthetic dataset was generated to simulate system behavior under normal and degraded conditions.

Synthetic data has been repeatedly proposed as a practical alternative when privacy, security, proprietary controls, or restricted network environments limit data availability; thus, making this dataset viable for the purposes of the study [10][11][12].

The dataset consisted of 200 total observations, with class labels distributed as follows:

- Normal (0): 125 samples (62.5%)
- Degraded (1): 75 samples (37.5%)

This distribution reflects a moderate class imbalance, which is representative of real-world monitoring environments where degraded states occur less frequently than nominal operation. The data was structured in tabular observations containing the variables listed in Table 1.

Table 1. Observational Data Variables

Feature	Meaning	Description
CPU_USAGE	CPU Load	Simulated processor utilization
MEMORY_USAGE	Memory Load	Simulated memory utilization
LATENCY_MS	Delay	Simulated network latency
ERROR_RATE	Faults	Simulated operational error frequency
PACKET_LOSS	Network Loss	Simulated network packet loss

While synthetic data enables controlled experimentation, it may not fully capture real-world variability such as temporal drift, correlated failures, or adversarial conditions. The generated dataset was designed to approximate typical system degradation patterns by increasing multiple stress indicators simultaneously. However, the representativeness of these patterns remains a limitation, and results should be interpreted as proof-of-concept rather than direct operational validation.

4.2. Data Preparation

The dataset was loaded into a Python virtual environment using pandas and separated into:

- Feature matrix X
- Target vector y

The target column label was removed from the feature matrix prior to training.

4.3. Train-Test Split

To evaluate generalization performance, the dataset was partitioned using an 80/20 train-test split:

- 80% training data
- 20% testing data

Stratified sampling was used to preserve class proportions across both partitions, reducing sampling imbalance risk. A fixed random seed (random_state = 42) was used to support reproducibility.

4.4. Validation Strategy

To assess model stability beyond a single train-test split, 5-fold stratified cross-validation was conducted. This method rotates training and validation partitions while preserving class balance in each fold, producing a mean and standard deviation estimate of model performance.

4.5. Baseline Comparator

A threshold-based baseline was implemented to simulate conventional monitoring logic:

$$Degraded = (cpu\_usage > 80) \wedge (latency\_ms > 100)$$

This baseline was intentionally simple and served as a practical comparator against learned classification models.

5. METHODOLOGY

This study employed a comparative supervised ML framework to evaluate lightweight system-state classification under constrained conditions. Three classification approaches were assessed:

- Logistic Regression
- Random Forest
- Rule-Based Threshold Baseline

Logistic Regression and Random Forest were selected to represent contrasting model classes: a lightweight linear classifier and a higher-complexity ensemble method. This selection enables evaluation of whether increased model complexity provides meaningful performance gains under constrained conditions. While the Rule-based Threshold Baseline was selected to represent traditional approaches in static system monitoring rules.

5.1. Logistic Regression

Logistic Regression was selected as the primary lightweight model due to its computational efficiency, interpretability, and established effectiveness in binary classification tasks. Prior literature has repeatedly shown that simpler models can remain highly competitive when feature relationships are sufficiently informative [14][15]. This formulation follows standard logistic regression modeling approaches as described in [15]. The Logistic Regression model estimates the probability of a system being in a degraded state using a linear combination of input features followed by a nonlinear transformation.

First, a linear predictor is computed:

Equation 2. Linear Predictor

$$z = w^T x + b$$

Variables within the Linear Predictor Equation represent:

- $x \in R^5$ is the input feature vector representing telemetry measurements.
- $w \in R^5$ is the vector of learned model coefficients (feature weights).
- w^T x denotes the dot product between the weight vector and feature vector.
- $b \in R$ is the bias term (intercept).
- $z \in R$ is the linear combination of weighted inputs.

This linear predictor is then passed through the sigmoid (logistic) function:

Equation 3. Sigmoid Equation

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Variables within the Sigmoid Equation represent:

- σ(z) maps any real-valued input z to a value between 0 and 1.
- e is the base of the natural logarithm.

The final model output is the probability of the degraded class:

Equation 4. Probability of Degradation Classification Equation

$$P(y = 1|X) = \frac{1}{1 + e^{-(w^T x + b)}}$$

Variables within the Probability of Degraded Classification Equation represent:

- P(y = 1 | X) represents the probability that the system is in a degraded state given input features X.

Predictions are obtained by applying a classification threshold (typically 0.5), where values above the threshold are classified as degraded and values below as normal. Predictions above the classification threshold were assigned to the degraded class. For model convergence, the maximum iteration parameter was set to: max_iter = 1000. This formulation follows standard logistic regression modeling approaches as described in [15][16][17]. The use of Logistic Regression in this study aligns with prior work demonstrating that interpretable linear models can provide competitive performance relative to more complex machine learning approaches in structured data environments [15].

5.2. Random Forest

Random Forest was selected as a more complex comparator model. Random Forest combines multiple decision trees through ensemble voting and is often used to capture nonlinear relationships and feature interactions. This model was included to test whether increased complexity would outperform Logistic Regression on structured telemetry data. The Random Forest classifier was initialized using default scikit-learn settings with a fixed random seed: random_state = 42.

5.3. Rule-Based Threshold Baseline

To represent traditional monitoring logic, a static threshold rule was implemented:

$$Degraded = (cpu\_usage > 80) \wedge (latency\_ms > 100)$$

This baseline simulates common alerting approaches in which isolated metrics trigger warnings once predefined limits are exceeded. Its inclusion provides a practical benchmark for determining whether ML meaningfully improves upon conventional threshold monitoring.

5.4. Performance Metrics

All models were evaluated using standard binary classification metrics:

- Accuracy
- Precision
- Recall
- F1-score

These measures were selected because accuracy alone may obscure imbalance or error tradeoffs. Precision and recall are particularly relevant in monitoring applications where false alarms and missed degraded states both carry operational cost. Confusion matrices were also generated for model-level error inspection.

5.5. Cross-Validation

To assess stability beyond a single train-test split, 5-fold stratified cross-validation was performed. Each fold preserved class balance while rotating validation subsets. The resulting mean F1-score and standard deviation were used to estimate model consistency across partitions.

5.6. Model Persistence

The highest-performing trained model was serialized using the *joblib* library and saved for reuse in downstream inference workflows. This supports reproducibility and future deployment integration.

6. EXPERIMENTAL SETUP

6.1. Software Environment

All experiments were conducted in a local Python environment using the following software stack, that included Python Version 3.14.3, scikit-learn Version 1.8.0, pandas Version 3.0.2, and joblib Version 1.5.3. The implementation was executed within a virtual environment to isolate dependencies, remove potential test noise, and support reproducibility.

6.2. Hardware Environment

Model training and evaluation was conducted using a standard consumer workstation environment. No dedicated GPU acceleration or distributed compute resources were required to support interoperability of the model with Edge and low-computation power devices. Further, this was necessary to demonstrate consistency with the study objective of evaluating lightweight ML approaches that can function without specialized hardware.

6.3. Operating System and Execution Content

Experiments were executed in a current Windows-based development environment using Visual Studio Code Version 1.115.0, and PowerShell terminal execution. This workflow reflects a practical practitioner-level implementation rather than a specialized research cluster environment. The use of a consumer product in a virtual environment was a first phase test design for the model's capabilities and limitations on a single low-performance environment. The intent was to represent actions and computing power requirements an Edge Device, such as a Raspberry Pi, could manage while functioning alongside a high-performance server. This test environment was meant as the first phase and proof-of-concept that would lead to more technical and diverse testing phases.

6.4. Data Pipeline Execution

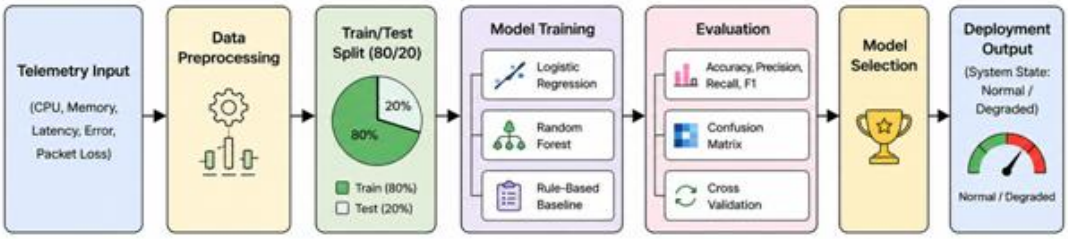

Figure 1. Lightweight system-state classification pipeline from telemetry ingestion to model output.

A stepwise approach was taken in this experiment, the pipeline steps were as follows:

Step 1: Load synthetic telemetry dataset (n=200).
Step 2: Separate features and labels.
Step 3: Perform stratified 80/20 train-test split.
Step 4: Train candidate models.
Step 5: Evaluate performance metrics.
Step 6: Conduct 5-fold cross-validation.
Step 7: Compare against threshold baseline.
Step 8: Save best-performing model.

The pipeline was implemented through modular Python scripts to support repeatability. The overall system pipeline is illustrated in Figure 1.

6.5. Runtime Efficiency Measurement

Inference efficiency was measured by timing prediction execution across the held-out test set and dividing total elapsed time by number of samples. Average inference time was reported in milliseconds per sample to assess suitability for low-overhead monitoring contexts.

6.6. Reproducibility Controls

To reduce randomness and improve repeatability, fixed seeds were used where supported: random_state = 42. This parameter was applied during:

- Train-test split
- Random Forest initialization
- Cross-validation shuffling

7. Results and Discussion

7.1. Comparative Model Performance

Three approaches were evaluated for binary system-state classification: Logistic Regression, Random Forest, and a rule-based threshold baseline. Results demonstrated clear separation in performance across the three methods. Logistic Regression produced the strongest overall results, achieving an accuracy of 0.97, precision of 0.973, recall of 0.947, and F1-score of 0.959. Random Forest achieved an accuracy of 0.935, precision of 0.956, recall of 0.867, and F1-score of 0.909. The rule-based baseline substantially underperformed both ML approaches, achieving an accuracy of 0.625, recall of 0.000, and F1-score of 0.000. These findings support prior work suggesting that well-structured simpler models can remain highly competitive and may

outperform more complex alternatives when feature relationships are sufficiently informative [14][15]. The comparative model performance metrics verified the intent of the experiment to evaluate low-computation environments.

7.2. Logistic Regression Error Analysis

The confusion matrix for Logistic Regression showed:

- **True Negatives** = 123
- **True Positives** = 71
- **False Positives** = 2
- **False Negatives** = 4

This indicates that the model correctly classified the large majority of both normal and degraded system states while maintaining low false alarm and missed-detection rates. From an operational perspective, false negatives often represent the more serious error type because degraded conditions may go unrecognized. The low false negative count observed in this study suggests reliable detection performance under simulated conditions. The logistic regression error analysis metrics verified the intent of the experiment to evaluate the light-weight ML model's ability to perform the proposed task.

7.3. Cross-Validation Stability

Five-fold stratified cross-validation was conducted to assess model consistency beyond a single train-test split.

Logistic Regression achieved:

- Mean F1-score = **0.980**
- Standard Deviation = **0.012**

Random Forest achieved:

- Mean F1-score = **0.882**
- Standard Deviation = **0.022**

The Logistic Regression model not only outperformed Random Forest on the held-out test set but also demonstrated stronger stability across partitions. The low variance suggests that performance was not dependent on a favorable single split. This finding is particularly relevant for practical deployment, where robustness across changing data conditions is often more valuable than isolated peak performance. The cross-validation stability metric aided in the experiment's intent of evaluating repeatability and effectiveness of the model.

7.4. Baseline Comparison

The threshold-based baseline classified a system as degraded only when:

$$(cpu\_usage > 80) \wedge (latency\_ms > 100)$$

This rule produced:

- **True Positives** = 0
- **False Negatives** = 75
- **Recall** = 0.000
- **F1-score** = 0.000

The baseline failed to detect any degraded states in the held-out dataset. This result illustrates a common limitation of static monitoring rules: system degradation may emerge through interacting moderate indicators rather than a single severe threshold breach. These findings align with prior literature arguing that learned decision boundaries can outperform rigid rules when system behavior is multi-dimensional [5][6].

7.5. Runtime Efficiency

Average inference time for the Logistic Regression model was approximately 0.006 milliseconds per sample under local evaluation conditions. This result suggests potential suitability for low-overhead environments. However, these results were obtained on a consumer workstation and should not be interpreted as direct evidence of performance on embedded or edge devices without further hardware-specific validation. For constrained systems, prediction speed and implementation simplicity may offer practical advantages beyond marginal gains in benchmark accuracy.

7.6. Discussion Summary

Three central findings emerged from this study:

- Lightweight Logistic Regression outperformed a more complex Random Forest model.
- Logistic Regression performance remained stable across cross-validation folds.
- Traditional threshold monitoring failed to identify degraded system states.

Taken together, these findings suggest that practical system-state classification does not always require high-complexity models. In some constrained environments, simpler interpretable models may represent the more operationally effective choice.

7.7. Comparison with Contemporary Approaches

Recent studies in anomaly detection and IoT monitoring frequently employ deep learning architectures, ensemble methods, or hybrid edge-cloud frameworks to achieve high detection performance [5][6]. These approaches often report strong classification accuracy but require significantly greater computational resources, model complexity, and infrastructure support.
In contrast, the present study demonstrates that a lightweight Logistic Regression model achieved an F1-score of 0.959 with minimal computational overhead and an average inference time of approximately 0.006 ms per sample. While direct comparison across studies is limited due to differences in datasets and problem framing, the results suggest that comparable classification performance can be achieved without reliance on high-complexity models. Furthermore, prior work emphasizes anomaly detection rather than explicit system-state classification, which limits direct applicability to operational monitoring contexts. The present approach addresses this gap by focusing on binary system-state classification with interpretable and deployable models, supporting practical use in constrained environments.

8. DEPLOYMENT CONSIDERATIONS

The results of this study suggest practical applicability in environments where low overhead, explainability, and local execution are prioritized. Potential deployment contexts include:

- Edge monitoring nodes
- Raspberry Pi or small-form-factor devices
- Air-gapped operational enclaves
- Local telemetry collectors
- Resource-limited industrial or field systems

Because Logistic Regression requires minimal memory and processing overhead relative to larger architectures, it may be integrated into lightweight monitoring pipelines without dedicated accelerators or cloud connectivity, making it a viable candidate for secure enclaves. A practical implementation model would involve passive telemetry collection, local feature generation, and periodic state inference with alert forwarding to operators or supervisory systems. Such

architectures may be especially useful where bandwidth, power, or security restrictions limit heavier solutions, as can be seen in geographically distributed systems. Prior TinyML and Edge AI literature similarly emphasizes the value of compact models for constrained deployment scenarios [8][9]. Future experiments involving operational networks and systems would add additional validity to this study's findings.

9. LIMITATIONS

Several limitations should be considered when interpreting these findings. First, the dataset used in this study was synthetically generated rather than derived from live production telemetry. Although synthetic data enables controlled experimentation, it may not fully capture real-world noise, drift, irregular workloads, or adversarial conditions. Second, the task was limited to binary classification of normal versus degraded states. Real operational systems may require multi-class state assessment, severity ranking, or root-cause attribution. Third, the feature set was intentionally compact and limited to five telemetry variables. Additional features such as disk I/O, throughput, service response time, or temporal trends may improve future model performance. Fourth, inference timing was measured under local development conditions rather than on dedicated embedded hardware. Runtime performance on target deployment devices may vary. Finally, only two ML models and one baseline rule were evaluated. Additional classifiers, calibrated thresholds, or temporal sequence models may provide alternative tradeoffs. These limitations provide a clear path for future refinement while not invalidating the present findings.

10.CONCLUSION

This study presented a lightweight ML pipeline for binary system-state classification using structured synthetic telemetry data. Across comparative evaluation, Logistic Regression achieved the strongest overall performance, outperforming Random Forest and substantially exceeding a rule-based threshold baseline. The model achieved high classification performance, stable cross-validation results, and rapid inference timing, indicating that reliable system-state awareness can be obtained without complex architectures or heavy computational requirements.

These findings challenge the assumption that increased model complexity is always necessary for operational monitoring tasks. In constrained environments, interpretable and efficient models may provide the most practical balance of performance, speed, and deployability. Future work should evaluate the proposed pipeline using real telemetry streams, expanded feature sets, temporal modeling approaches, and embedded deployment targets. Overall, the results support lightweight ML as a viable path for monitoring modern systems where simplicity and operational efficiency matter.

REFERENCES

[1] Chandrachood, A, (2023) "Optimizing resource allocation through telemetry-based performance monitoring", North American Journal of Engineering and Research, Vol. 4, No. 4, pp1-7.

[2] Yaseen, N, (2025) "From counters to telemetry: A survey of programmable network-wide monitoring", Network, Vol. 5, No. 38, pp1-29.

[3] Ghosh, S, & Srinivas, D, (2025) "A comprehensive review of telemetry data-driven AI cybersecurity solutions in IoT-based insurance ecosystems", International Journal of Emerging Research in Engineering and Technology, Vol. 6, No. 3, pp103-112.

[4] Esposito, M, Bakhtin, A, Ahmad, N, Robredo, M, Su, R, Lenarduzzi, V, & Taibi, D, (2025), "Autonomic microservice management via agentic AI and MAPE-K integration", European Conference on Software Architecture, pp105-118.

[5] Baimukhanov, S, Ali, H, Yazici, A, (2025) "Enhancing ML-based anomaly detection in data management for security through integration of IoT, cloud, and edge computing", Expert Systems with Applications, 128700, pp1-25.
[6] Rafique, S,A, Amira, A, Musa, N,S, & Murugan, T, (2024) "Machine learning and deep learning techniques for internet of things network anomaly detection-Current research trends", Sensors, Volume 24, 1968, pp1-32.
[7] Baltaji, G,E, (2022) "Anomaly detection at the edge implementing machine learning techniques [Masters Thesis]", Politecnico Di Torino, Department of Mathmatical Sciences, pp 1-40.
[8] Immomen, R, Hamalainen, T, (2022) "Tiny machine learning for resource-constrained microcontrollers", Journal of Sensors, Vol. 2022, Art ID. 7437023, 1-11.
[9] Kumar, R, Sharma, A, (2024) "Edge AI: A review of machine learning models for resource-constrained devices", Artificial Intelligence and Machine Learning Review, Vol. 5, No. 3, pp1-11.
[10] Long, Y, Kroeger, S, Zaeh, M,F, & Brintrup, A, (2025) "Leveraging synthetic data to tackle machine learning challenges in supply chains: Challenges, methods, applications, and research opportunities", International Journal of Production Research, pp1-22.
[11] Kiran, A, Rubini, P, & Saravana Kumar, S, (2025) "Comprehensive review of privacy, utility, and fairness offered by synthetic data, IEEE Access, Vol. 13, pp15795-15811.
[12] Chaithanya, D,J, Nagaraja, B,G, & Ravikiran, H,K, (2025) "Advanced predictive analytics with synthetic data: A comprehensive machine learning approach for predicting chronic and lifestyle diseases", Journal of Integrated Science & Technology, Vol. 13, No. 6, pp1139-1141.
[13] Jacobson, B,J, (2025) "Machine learning, synthetic data, and the politics of difference", Theory, Culture, & Society, Vol. 42, No. 3, pp41-58.
[14] van de Bijl, E,P, (2025) "From baselines to breakthroughs: Fundamentals and applications of machine learning in cybersecurity".
[15] Wu, H, Liao, B, Ji, T, Ma, K, Luo, Y, & Zhang, S, (2025) "Comparison between traditional logistic regression and machine learning for predicting mortality in adult sepsis patients", Frontiers in Medicine, Vol. 11, pp1496869-1496873.
[16] Bishop, C,M, (2006) "Pattern Recognition and Machine Learning", Springer.
[17] Hastie, T, Tibshirani, R, & Friedman, J, (2009) "The Elements of Statistical Learning", Springer.

AUTHOR

Dr. Jeremy D. Howard is an independent scholar working as an Electrical Design Engineer and has over 24 years of experience working in Command and Control Systems, Tactical Data Links, and both Sensors and Effectors as part of the United States Army. In his current setting he specializes in the flow of Tactical Data, Cybersecurity, Network Compliance, and the operational integration of AI/ML.