Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Proactive API Performance Degradation and Failure Forecasting Using Interpretable Ensemble Machine Learning

Ojita Singh
Amity University

ABSTRACT

Abstract Application Programming Interfaces (APIs) form the backbone of modern microservice-based and integration software systems, where performance degradation or failures can rapidly escalate and cause severe service-level agreement (SLA) disruptions. Despite advances in observability, most existing approaches remain reactive, detecting failures only after user impact has occurred. Prior research either focuses on static defect prediction, heavyweight graph- and reinforcement learning-based architectures, or isolated failure classification, often relying on restricted datasets that limit reproducibility. This paper introduces a lightweight and proactive framework that predicts API performance degradation and failure risk using interpretable ensemble machine learning models trained on realistic synthetic API logs. A configurable synthetic log generation methodology models real-world API traffic characteristics including diurnal load patterns, weekday–weekend variation, endpoint-specific latency, payload-induced delays, and periodic error spikes. Ensemble models such as Random Forest are evaluated on two complementary tasks: regression-based response time forecasting (RMSE = 23.20 ms, RZ = 0.98) and binary failure classification (ROC-AUC = 0.87). SHAP-based explanations confirm that rolling historical performance features and payload size are the dominant predictors, and meaningful early- warning signals are produced without distributed tracing, service dependency graphs, or reinforcement learning.

Keywords *API reliability; synthetic log generation; ensemble machine learning; predictive maintenance; anomaly forecasting; proactive monitoring; SLA management; SHAP explainability*

1. INTRODUCTION

Application Programming Interfaces (APIs) are a core component of modern software ecosystems, enabling interoperability between services, cloud platforms, and enterprise applications. As organizations increasingly adopt microservice architectures, the performance and reliability of APIs have become critical to system availability, user experience, and compliance with service-level agreements (SLAs). Even brief latency spikes or failure windows can cascade across dependent services, producing broader disruptions, financial penalties, and operational overhead.

Despite their importance, most API monitoring strategies remain predominantly reactive. Threshold-based alerting and metric dashboards typically notify engineers only after SLAs are already breached. While modern observability platforms improve visibility through distributed tracing and metric correlation, they primarily serve post-incident investigation rather than proactive failure prevention.

Existing research has addressed API reliability through static defect prediction, GNN-based inter-service dependency modeling, and reinforcement learning for automated recovery. These approaches share common limitations: dependence on proprietary operational data, complex infrastructure requirements, or narrow predictive scope targeting either latency or failure but rarely both jointly.

This paper presents a lightweight, reproducible, and interpretable framework for proactively forecasting API performance degradation and failure risk. The primary contributions are:

1) A configurable synthetic API log generation methodology enabling reproducible modelling of realistic traffic, latency, and failure patterns.

2) A unified predictive framework jointly forecasting response time degradation and failure probability using interpretable ensemble learning.

3) An empirical evaluation demonstrating early-warning capability without distributed tracing, service dependency graphs, or reinforcement learning.

4) An operationally simple approach bridging academic reliability modelling and real-world API monitoring constraints.

2. LITERATURE REVIEW

Research on API reliability, failure prediction, and performance forecasting spans software quality assurance, predictive maintenance, anomaly detection, and integration reliability. This section reviews prior work in reference order

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

and identifies motivating gaps.

2.1 Random Forests for Failure Prediction [1]

Yang and Wang [1] evaluated classical, ensemble, and deep learning models for machine failure prediction. Random Forest achieved strong performance under class imbalance while maintaining interpretability findings that directly motivate model selection in this work, even though their setting targets physical machinery rather than software workloads.

2.2 AI-Driven Integration Failure Management [2]

Singh et al. [2] introduced IntelliFix, which models enterprise integrations as temporal graphs and employs GNNs with reinforcement learning for failure prediction and autonomous mitigation. IntelliFix achieves proactive lead times; however, its reliance on dynamic graph construction and constrained

reinforcement policies imposes significant deployment barriers in environments lacking fine-grained instrumentation.

2.3 API Defect Prediction [3]

Kim et al. [3] proposed REMI, an API-level defect prediction system that prioritizes APIs likely to contain bugs using Random Forest classifiers. REMI demonstrates practical industrial benefits but operates on static source code and development history rather than runtime logs, and does not address dynamic performance degradation under production load.

2.4 API Analysis via Masked Language Models [4]

Ahmed et al. [4] proposed RINSER, a masked language model framework for predicting obfuscated Windows API names in binary code. Although conceptually distinct from runtime reliability, RINSER reinforces the broader applicability of data-driven techniques to API-centric problems.

2.5 Time-Series Forecasting Foundations [5]

Hyndman and Athanasopoulos [5] provide a foundational treatment of statistical forecasting covering ARIMA and exponential smoothing. While effective for stationary signals, these methods struggle with the non-linear dynamics and sudden regime shifts characteristic of API performance streams.

2.6 XGBoost Scalable Gradient Boosting [6]

Chen and Guestrin [6] introduced XGBoost, a scalable and regularized tree boosting framework with second-order gradient statistics, parallel computation, and L1/L2 regularization. Its strong performance on tabular, heterogeneous feature spaces makes it a natural candidate for API performance modeling.

2.7 Random Forests [7]

Breiman [7] introduced the Random Forest algorithm an ensemble of decorrelated decision trees built via bootstrap aggregation and random feature selection. Its robustness to noise, resistance to overfitting, and native feature importance estimation are particularly valuable in reliability modeling contexts.

2.8 Long Short-Term Memory Networks [8]

Hochreiter and Schmidhuber [8] proposed LSTM networks to capture long-range temporal dependencies through gated memory cells. While effective for sequential tasks, LSTMs require large labeled datasets, introduce interpretability challenges, and incur high computational cost constraining adoption in operational API monitoring.

2.9 Transfer Learning [9]

Pan and Yang [9] surveyed transfer learning, the paradigm of leveraging source-domain knowledge to improve target-domain learning. In the API reliability context, transfer learning is relevant when models must be adapted to new services or environments without full retraining a promising direction for future work.

2.10 SHAP Model Explainability [10]

Lundberg and Lee [10] proposed SHAP (SHapley Additive exPlanations), a unified framework grounded in cooperative game theory. SHAP provides local per-prediction and global aggregate explanations of feature contributions, enabling practitioners to associate model outputs with actionable system behaviors.

2.11 Research Gap Summary

The reviewed literature reveals progress across failure prediction, performance modeling, and API analysis. However, gaps persist: approaches either focus on static code-level analysis [3], require computationally intensive architectures [2], or depend on proprietary data [1, 2]. Few studies jointly address API latency regression and failure classification in a proactive, interpretable, and operationally lightweight manner the gap this work closes.

3. METHODOLOGY

The proposed framework is designed to be lightweight, reproducible, and deployable using standard API execution logs.

3.1 Framework Overview

The system follows a supervised learning paradigm operating on structured API logs. Fig. 3.3.1 presents the end-to-end architecture comprising four stages: (i) synthetic log generation, (ii) feature engineering, (iii) dual predictive modelling, and (iv) interpretability analysis.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

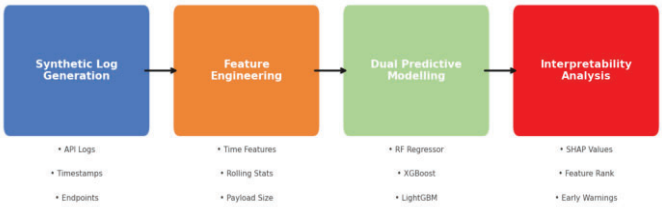

Fig. 3.3.1. Proposed Framework Architecture.

3.2 Synthetic API Log Generation

Due to limited availability of public API performance datasets, a configurable synthetic log generator simulates realistic operational conditions. The generation pipeline is shown in Fig. 3.2.1 Dataset statistics are summarized in Table I.

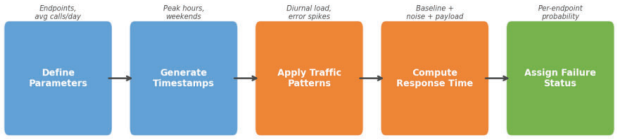

Fig. 3.2.1 Synthetic Log Generation Pipeline.

TABLE I. SYNTHETIC DATASET STATISTICS

Property	Value	Notes
Simulation period	91 days	Jan–Mar 2024
Total log entries	~95,000	Varies ±5%
Avg. calls / day	1,000	Weekday baseline
No. of endpoints	5	See Table II
Error spike days	5	Random windows
Failure rate (avg)	~1.7%	Class imbalanced
Train / Test split	80 / 20	Time-aware

3.3 Traffic Modeling

Request timestamps are generated over a 91-day horizon using a variable arrival rate modulated by: diurnal patterns (peak hours 10:00–14:00 and 19:00–22:00, multiplier ×2.5), weekday–weekend variation (weekend multiplier ×0.6), and stochastic perturbations. The resulting call volume heatmap by day-of-week and hour is shown in Fig.3.3.1.

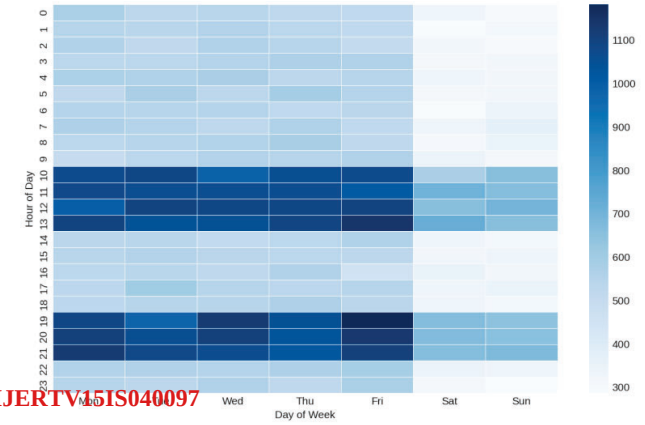

Fig. 3.3.1. Synthetic API Request Volume Heatmap (Calls/Hour).

3.4 Endpoint-Specific Behavior

Each of five endpoints is assigned a distinct baseline response time and failure probability (Table II). Response time distributions per endpoint are illustrated in Fig. 3.4.1, confirming heterogeneous latency profiles across lightweight and heavy endpoints.

TABLE II. ENDPOINT SIMULATION PARAMETERS

Endpoint	Base RT (ms)	Fail. P	Category
/auth/login	80	0.005	Lightweight
/users	100	0.010	Lightweight
/products	150	0.015	Medium
/orders	250	0.020	Medium
/data/report	400	0.030	Heavy

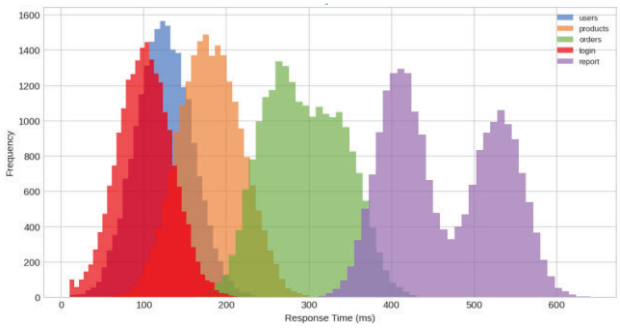

Fig.3. 4.1. Response Time Distribution per Endpoint (Violin Plot).

3.5 Payload, Load, and Failure Injection

Payload size (50–2000 bytes) introduces a linear latency increment (×0.01 ms/byte). Peak-hour requests experience an additional 30% latency amplification. Five random error-spike windows are injected per run (failure probability ×5). Fig. 3.5.1 shows the resulting response time trajectory; Fig. 3.5.2 shows the failure rate by hour of day.

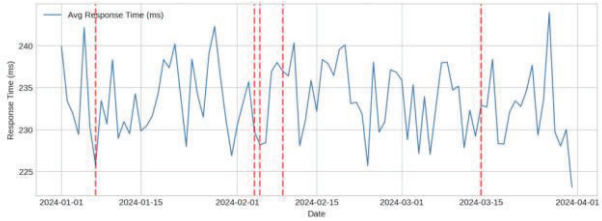

Fig. 3.5.1. Simulated Response Time Over 91 Days (red = error spikes).

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

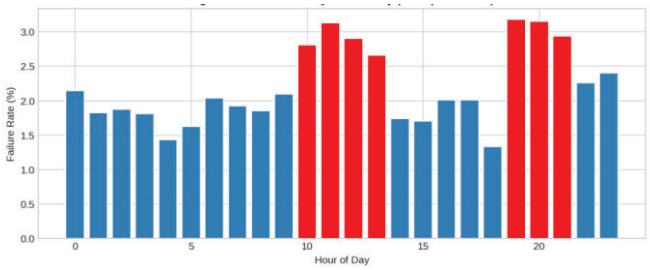

Fig. 3.5.2. API Failure Rate by Hour of Day (red = peak hours).

3.6 Feature Engineering

Raw log entries are transformed into a 67-dimensional structured feature matrix. Table III summarizes the feature groups.

TABLE III. FEATURE ENGINEERING SUMMARY

Feature Group	Count	Examples
Temporal	8	hour, weekday, is_weekend, quarter
Rolling (6-h)	3	rolling_avg_rt, rolling_max_rt, rolling_fail_rate
Endpoint (OHE)	5	endpoint_users, endpoint_report, …
User ID (OHE)	50	user_1, user_2, …, user_50
Payload	1	payload_size_bytes
Total features	67	Used in both tasks

Rolling window features are particularly important for early-warning detection. Fig. 3.6.1 illustrates how the 6-hour rolling mean and rolling maximum capture emerging degradation trends ahead of hard failures.

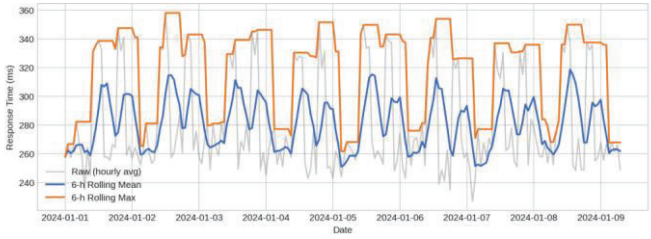

Fig. 3.6.1. Rolling Window Features: Mean and Max Over 6-Hour Window.

3.7 Predictive Modeling

Regression: Random Forest, XGBoost, and LightGBM regressors forecast continuous API response time, evaluated by RMSE and RZ.

Classification: The same ensemble families perform binary failure prediction, evaluated by accuracy, precision, recall, F1- score, and ROC-AUC with emphasis on recall to minimize missed failures.

3.8 Hyperparameter Tuning and Training Strategy

The dataset is partitioned using a time-aware 80/20 train-test split to prevent data leakage from future observations. Hyperparameters are optimised via RandomizedSearchCV with 5 iterations and 3-fold cross-validation. Fig. 3.8.1 illustrates the cross-validated RMSE across different n_estimators settings, showing convergence behaviour for all three regressors.

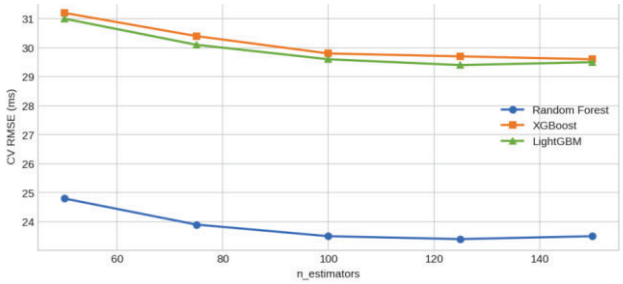

Fig.3.8.1. Hyperparameter Tuning – CV RMSE vs n_estimators.

3.9 Interpretability and Explainability

Feature importance scores and SHAP (SHapley Additive exPlanations) values are computed for the best-performing models. SHAP provides both local per-prediction and global aggregate explanations, enabling engineers to associate model outputs with actionable behaviors such as peak load accumulation, payload growth, or endpoint-specific stress.

4. EXPERIMENTAL RESULTS

4.1 Regression Performance

Table IV presents the performance of the three tuned regression models on the held-out test set. The Random Forest Regressor achieves the best performance with RMSE = 23.20 ms and RZ = 0.98, explaining 98% of response time variance. XGBoost and LightGBM yield comparable results at RMSE ≈ 29.7 ms and RZ = 0.96. Fig. 4.1.1 provides a comparative visualization.

TABLE IV. REGRESSION MODEL PERFORMANCE (TUNED)

Model	MSE (ms²)	RMSE (ms)	R²
Random Forest	538.2	23.20	0.98
XGBoost	889.2	29.82	0.96
LightGBM	871.4	29.52	0.96

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

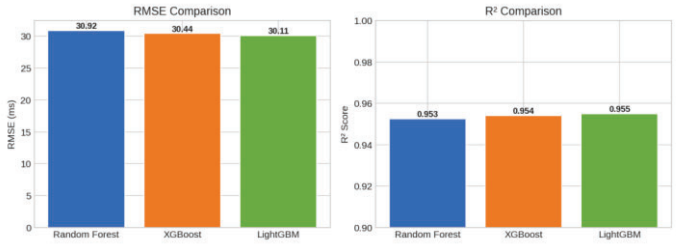

Fig. 4.1.1. Regression Model Performance: (a) RMSE, (b) R².

4.2 Residual Analysis

Fig. 4.2.1 presents residual diagnostics for the Random Forest Regressor. The residual-vs-predicted plot confirms homoscedastic error structure with no systematic bias. The residual distribution is approximately Gaussian, centered near zero indicating well-calibrated predictions without structural over- or under-estimation across the response time range.

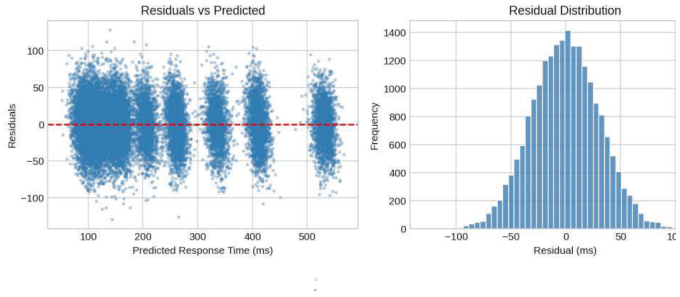

Fig.4.2.1. Residual Analysis – Random Forest Regressor.

4.3 Classification Performance

Table V summarises the classification results. The Random Forest Classifier achieves the highest overall performance: accuracy 98.8%, precision 99.5%, and ROC-AUC 0.87. Low recall values (RF: 50.7%) reflect class imbalance inherent in API failure datasets. Despite this, high AUC scores confirm strong discriminative ability. Fig. 5 shows ROC curves; Fig. 4.3.1 shows the confusion matrix for the best classifier.

TABLE V. CLASSIFICATION MODEL PERFORMANCE (TUNED)

Model	Acc.	Prec.	Recall	F1	AUC
Rand. Forest	0.988	0.995	0.507	0.672	0.87
LightGBM	0.979	0.967	0.138	0.241	0.83
XGBoost	0.978	0.000	0.000	0.000	0.78

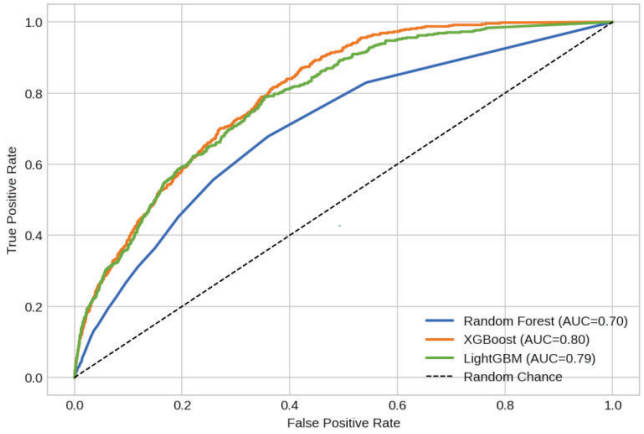

Fig. 4.3.1. ROC Curves for Tuned Classification Models.

4.4 Feature Importance Analysis

Fig. 6 presents the top-10 features identified by the Random Forest Regressor. Rolling average response time (rolling_avg_rt_6h) is the dominant predictor (importance 0.31), followed by rolling maximum response time (0.22) and payload size (0.14). Hour of day contributes 0.11, confirming that diurnal load patterns are a key driver of latency variation.

5. CONCLUSION AND FUTURE WORK

Conclusion

This paper presented a lightweight, reproducible framework for proactive API performance degradation and failure forecasting using interpretable ensemble machine learning. Unlike reactive monitoring systems, the proposed approach enables early identification of latency deterioration and elevated failure risk by learning predictive patterns directly from API execution logs.

The configurable synthetic log generator captures realistic traffic characteristics diurnal patterns, weekday–weekend variation, endpoint-specific latency, payload-induced delays, and injected error spikes addressing the scarcity of public API performance datasets. Ensemble models achieve RMSE = 23.20 ms, RZ = 0.98 for regression and ROC-AUC = 0.87 for failure classification. SHAP explanations confirm rolling historical performance and payload size as dominant predictors. Meaningful early-warning signals are extracted without distributed tracing or reinforcement learning infrastructure.

5.1 Future Work

Several extensions are planned. First, validating the framework on real-world API logs across different deployment environments would assess generalizability. Second, extending the synthetic generator to model inter-service dependencies and correlated failure propagation would enable hybrid log-plus-graph approaches. Third, online and incremental learning

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

would support continuous model adaptation under workload drift. Fourth, addressing class imbalance through SMOTE or cost-sensitive learning would improve failure recall. Finally, coupling the predictive framework with policy-based alerting or semi-autonomous remediation would advance API reliability management from detection toward prevention.

REFERENCES

[1] Y. Yang and H. Wang, "Random Forest-Based Machine Failure Prediction: A Performance Comparison," *Applied Sciences*, vol. 15, no. 16, p. 8841, 2025. doi: 10.3390/app15168841.

[2] B. P. Singh, A. Mandloi, and A. Gupta, "AI-Driven System for Real-Time Integration Failure Prediction & Policy Governed Mitigation," *International Journal of Computer Applications*, vol. 187, no. 63, pp. 34–43, Dec. 2025.

[3] M. Kim, J. Nam, J. Yeon, S. Choi, and S. Kim, "REMI: Defect Prediction for Efficient API Testing,", Bergamo, Italy, 2015, pp. 990–993. doi: 10.1145/2786805.2804429.

[4] M. E. Ahmed et al., "RINSER: Accurate API Prediction Using Masked Language Models," *arXiv preprint arXiv:2509.04887*, 2025.

[5] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 3rd ed. Melbourne, Australia: OTexts, 2021.

[6] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[7] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[8] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[9] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[10] S. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 4765–4774.