2025 9th International Conference on Computational System and Information Technology for Sustainable Solutions (CSITSS) | 979-8-3315-8894-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/CSITSS67709.2025.11295068

Proactive Anomaly Detection in Storage System Performance Metrics Using Isolation Forest and Observability Integration

Raksha B R
Dept. of CSE
RV College of Engineering
Bengaluru, Karnataka-560059

Varuni H Kulkarni
Dept. of ISE
RV College of Engineering
Bengaluru, Karnataka-560059

Dr. Badarinath K
Associate Professor
Dept. of CSE
RV College of Engineering
Bengaluru, Karnataka-560059

Mr. B K Srinivas
Assistant Professor
Dept. of ISE
RV College of Engineering
Bengaluru, Karnataka-560059

***Abstract*— Modern cloud storage systems produce large volumes of performance data, necessitating real-time anomaly detection to ensure high availability and system reliability.**

The proposed paper presents an unsupervised machine learning approach for detecting anomalies in storage performance metrics using the Isolation Forest algorithm. The system integrates Telegraf for data collection, InfluxDB for time-series storage, and Grafana for visualization and alerting. By eliminating the need for labeled data, this ensemble-based model dynamically adapts to evolving workloads and detects outliers with minimal overhead.

Experimental validation on real telemetry showed optimal detection at 0.01 contamination (5%), with increased sensitivity at 0.03 and balanced output at 0.05. Results confirm that the approach enables proactive diagnostics and significantly reduces issue resolution time through AI-driven observability.

Keywords*— *Anomaly detection, Isolation Forest, storage systems, performance counters, unsupervised learning, observability, InfluxDB, Grafana, Telegraf, predictive monitoring, real-time analysis, telemetry, quality engineering.

I. Introduction

In large-scale, data-intensive systems, storage infrastructure is a vital layer across cloud platforms, enterprise data centers, and distributed computing environments [1]. These systems must offer high availability, low latency, and consistent performance while handling massive data volumes. Real-time telemetry, captured via performance counters such as IOPS, CPU usage, latency, and node availability, serves as a critical tool for monitoring system health [2].

However, anomaly detection within this telemetry stream is challenging due to its high volume, velocity, and multidimensional nature. Fixed thresholds and manual inspection fail to adapt to changing baselines caused by system updates and workload shifts, leading to false alerts or missed issues [3].

To address these challenges, this paper introduces a machine learning-based anomaly detection framework using the **Isolation Forest algorithm** [4]. This unsupervised, tree-based model isolates anomalies by evaluating path lengths in randomly partitioned trees. Its computational efficiency and CPU-only operation make it suitable for real-time, high-frequency data monitoring without requiring GPU resources [5].

Recent research supports combining telemetry-aware feature engineering with models like Isolation Forest for operational observability in distributed systems [6]. Unsupervised anomaly detection is increasingly integrated into DevOps pipelines for early failure detection [7], and studies confirm Isolation Forest's suitability for edge and production-scale environments [8].

Our framework comprises four modular components: (a) telemetry ingestion via Telegraf, (b) time-series storage using InfluxDB, (c) anomaly detection through scalable Python pipelines, and (d) visualization and alerting using Grafana. This design supports dynamic adaptation to evolving system behaviors.

The goals of this work are to:

- Detect early performance degradations via telemetry deviations,
- Integrate with open-source observability tools like Telegraf, InfluxDB, and Grafana,
- Efficiently process multivariate telemetry on standard CPUs,
- Remain portable across diverse storage platforms including NAS, object, and hybrid systems.

By turning telemetry into actionable insights, this system reduces MTTR and improves reliability in large-scale storage deployments. The approach aligns with current DevOps practices that emphasize automation and adaptive alerting to sustain high system availability [9].

II. Methodology

To address real-time anomaly detection in cloud storage systems, we followed a four-phase development cycle: (a) problem analysis and system design, (b) telemetry data collection and preprocessing, (c) model training and serialization using Isolation Forest, and (d) integration with observability tools for live visualization and alerts. This phased approach ensured a systematic transition from data acquisition to actionable insights. Each phase was designed to support scalability, automation, and adaptability to dynamic infrastructure environments.

2.1 System Architecture Design:

As shown in Fig. 2.1, the modular architecture processes telemetry data (e.g., IOPS, latency, CPU, error rate) through an end-to-end pipeline—from data ingestion to anomaly detection and visualization. Traditional static-threshold monitoring fails under dynamic workloads; our loosely coupled design enables scalable, real-time detection and easy ML model integration without disrupting the pipeline.

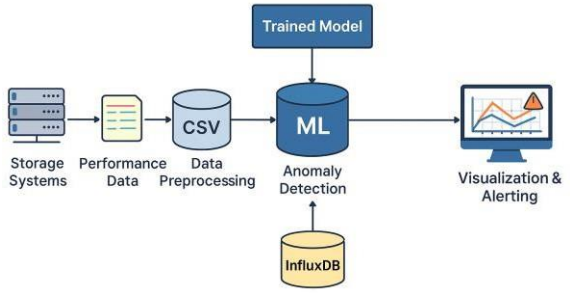

Figure 2.1: System Architecture

The architecture comprises the following components:
• **Telegraf** is an open-source agent for collecting and reporting metrics. In this project, it enables lightweight, real-time telemetry ingestion from storage systems with minimal system overhead [10].
• **InfluxDB** is a high-performance time-series database optimized for storing and querying telemetry data. It was selected for its efficiency in handling large volumes of timestamped metrics[11].
• **Python libraries** (Pandas, NumPy, and scikit-learn) are used for preprocessing, model training, and anomaly detection using the Isolation Forest algorithm.
• **Grafana** is an open-source analytics and visualization platform. In this system, it enables intuitive dashboards and real-time alerting to support rapid interpretation of detected anomalies [12].
Recent works propose hybrid anomaly detection combining statistical forecasting with machine learning [13], [14], while deep autoencoders and graph-based neural methods extend applicability to cloud-native and distributed storage systems [13], [15].

2.2 Data Collection and Preprocessing

Telemetry data is captured by Telegraf agents and stored in InfluxDB for efficient time-series analysis. A Python pipeline extracts relevant time windows and applies Isolation Forest for anomaly detection. Preprocessing steps include handling missing values, normalization, and windowed aggregation of metrics. Table 2.1 summarizes key preprocessing operations ensuring data quality and model readiness.

Table 2.1: Preprocessing Steps for Telemetry Data

Step	Description
Null Value Handling	Forward-fill and interpolation to impute missing data
Normalization	Min-max scaling to normalize metric ranges
Window Aggregation	Binning data into 1- and 5-minute intervals
Feature Selection	Selection of relevant metrics: IOPS, latency, CPU, memory, error count
Export Format	Export of processed datasets as CSV for model input

The anomaly detection workflow, as shown in Fig. 2.2, uses the Isolation Forest model on preprocessed telemetry data. Anomaly scores are evaluated against a threshold to flag outliers, which are then visualized and alerted through Grafana. A feedback loop supports periodic retraining to adapt to evolving system behavior and workload changes.

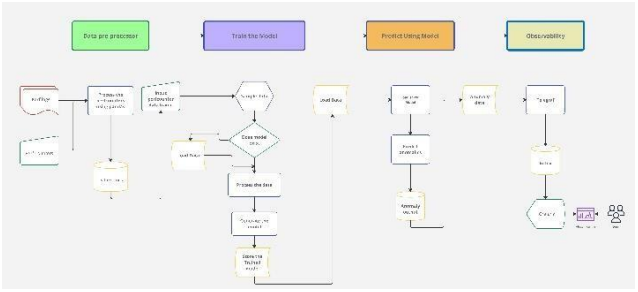

Figure 2.2: Presents the detailed pipeline for data ingestion, preprocessing, model training, prediction, and integration with observability

The machine learning workflow Fig. 2.3 outlines the anomaly detection pipeline using Isolation Forest. Preprocessed telemetry data is scored for anomalies based on statistical isolation and compared with a contamination threshold. Flagged results are visualized in Grafana, with periodic model retraining for adaptability.

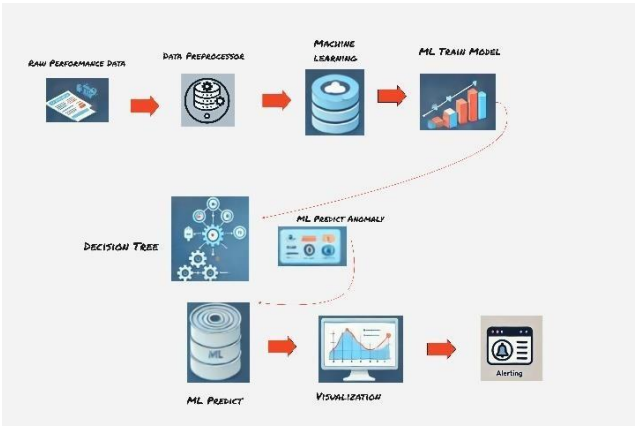

Figure 2.3: Machine learning workflow depicting anomaly scoring, threshold evaluation, and real-time alerting with feedback-based model retraining.

2.3 Isolation Forest Algorithm—Working Principle

Isolation Forest is an unsupervised anomaly detection algorithm that isolates anomalies instead of modeling normal behavior. It builds an ensemble of isolation trees (iTrees) by recursively selecting random features and split values to partition the dataset. Anomalies, being rare and distinct, are isolated in fewer splits, resulting in shorter average path lengths. Normal points require more partitions for isolation. This approach is effective in high-dimensional spaces and is ideal when labeled anomalies are unavailable. Primary algorithm parameters include:
• **n_estimators** – the number of trees in the forest, which controls the ensemble size and affects the stability of the anomaly score;
• **max_samples** – the number of samples drawn from the dataset to train each base estimator (tree), influencing

randomness and computational cost;
• **contamination** – the expected proportion of anomalies in the dataset, used to calibrate the threshold for labeling outliers. The anomaly score (eq. 1) for a data point x is calculated as:

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}} \quad (1)$$

Isolation Forest detects anomalies by computing the average path length E(h(x)) of a data point x across isolation trees, normalized using c(n), the average path length of a Binary Search Tree of size n. Scores closer to 1 indicate anomalies, while scores near 0 suggest normal behavior. This project uses scikit-learn's IsolationForest, offering optimized performance with O(nlogn) complexity, making it suitable for large-scale, unlabeled telemetry data without GPU dependency. Unlike ARIMA and Prophet, which assume stationarity and seasonal trends, Isolation Forest adapts to highly variable workloads. Deep models require large labeled datasets and GPU resources, whereas Isolation Forest achieves real-time detection on CPU-only environments.

2.4 Model Training and Persistence

A baseline model is trained on historical telemetry under normal conditions. The pipeline includes feature sampling, min-max scaling, Isolation Forest training, and synthetic anomaly injection for validation. The model is persisted using Python's pickle for reuse. Modular scripts (train.py, predict.py) automate training and inference, supporting integration into CI/CD pipelines. This aligns with recent advancements in self-healing CI/CD systems that embed anomaly detection and rollback to enhance resilience [10].

2.5 Real-Time Prediction and Output Generation

The real-time prediction pipeline uses the trained Isolation Forest model to analyze incoming telemetry data from InfluxDB. Data undergoes the same preprocessing (imputation, normalization, aggregation) as in training to maintain consistency. The model outputs anomaly scores and binary labels (1 = normal, –1 = anomaly), which are stored in CSVs and pushed to InfluxDB via a custom Telegraf plugin. This enables tight integration with Grafana for real-time visualization. Gradient-level alerting allows severity-based anomaly classification, improving response prioritization and reducing false positives and alert fatigue.

2.6 Observability and Alerting Integration

To support seamless operational monitoring, the system integrates with Grafana, Telegraf, and InfluxDB for real-time visualization and alerting. Anomaly scores are overlaid on performance metrics (e.g., CPU, IOPS, latency) in Grafana dashboards, providing clear correlation and aiding root cause analysis.

The observability stack includes:

- **Telegraf**: Collects system metrics and forwards anomaly predictions via a custom output plugin.
- **InfluxDB**: Stores both telemetry and anomaly data, enabling time-synced analysis.
- **Grafana**: Visualizes trends and anomaly scores with customizable dashboards and alert rules.

Threshold-based alerting in Grafana (e.g., 3 anomalies in 5 minutes) reduces false positives and alert fatigue. Notifications are routed via Slack, Teams, or email. This framework enhances MTTD and MTTR, improving storage system reliability.

2.7 Tabular Feature Summary

The anomaly detection system relies on a curated set of telemetry metrics that serve as indicators of system health and performance. These include fundamental indicators such as CPU utilization, memory usage, IOPS, network throughput, and read/write latency. Each metric plays a specific role in characterizing workload behavior and detecting deviations from normal operating conditions. Table 2.2 summarizes the selected features, providing details on their semantic meaning, units of measurement, and the frequency at which they are collected. This structured representation ensures transparency in feature selection and reproducibility of the anomaly detection framework.

Table 2.2: Summary of Key Telemetry Features for Anomaly Detection

Metric Name	Description	Unit	Frequency
IOPS	Input/output operations per second	ops/sec	10 seconds
Latency	Average disk latency	ms	10 seconds
CPU Usage	CPU consumed by storage service	%	10 seconds
Memory Usage	RAM utilization	MB	10 seconds
Error Count	Storage device or controller errors	count	10 seconds

2.8 Validation and Iteration

To enhance robustness, several refinements were made: contamination tuning for balanced sensitivity, score normalization to [0, 1] for consistent thresholding, and timestamp standardization across all components for accurate alert alignment. Core scripts were modularized with CLI support for CI/CD integration. Validation using real and synthetic telemetry confirmed the system's effectiveness in detecting both major failures and subtle issues like memory leaks and I/O contention.

III. Implementation

The real-time anomaly detection system was designed for scalability, modularity, and low-latency inference using unsupervised ML on distributed telemetry data. It integrates Python-based ML workflows with open-source observability tools for seamless production deployment.

3.1 Development Environment and Toolchain

Development was performed using MobaXterm for remote access to Linux environments (Ubuntu 20.04 LTS, openSUSE Leap), with Python 3.8+ for ML support. The observability stack includes Telegraf (telemetry), InfluxDB (time-series storage), and Grafana (visualization and alerts). A modular file structure ensured maintainability and CI readiness:

- train.py, predict.py, preprocess.py, export_results.py
- config/ for system and model parameters

- utils/ for helper functions and timers

3.2 Data pipeline and Telemetry flow

To improve robustness, the system was refined with contamination tuning, score normalization, and standardized timestamps. Modular CLI scripts support CI/CD integration. Validation with real and synthetic data confirmed accurate detection of both major and subtle anomalies. The anomaly score heatmap Fig. 3.1 further illustrates the model's ability to highlight deviation patterns across metrics and time intervals.

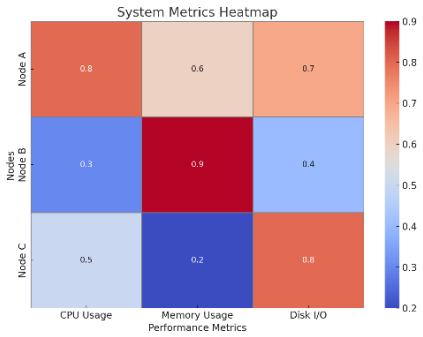

Figure 3.1: Heatmap showing anomaly intensity across metrics and time.

3.3 Machine Learning Architecture and Isolation Forest Tuning

The core anomaly detection uses the Isolation Forest algorithm, ideal for high-dimensional, unlabeled storage telemetry. It builds random trees to isolate points, with anomalies showing shorter average path lengths. Models are trained on stable operational data, with contamination typically set between 0.01–0.05.

Hyperparameters like n_estimators and max_samples are tuned via grid search on datasets enhanced with synthetic anomalies (e.g., disk faults, bottlenecks, spikes). Trained models are serialized with pickle for reuse. Feature relevance is inferred from path length distributions to guide retraining.

Fig 3.2 shows a 3D scatter plot (e.g., latency, IOPS, CPU), where anomalies appear distinctly separated from normal clusters.

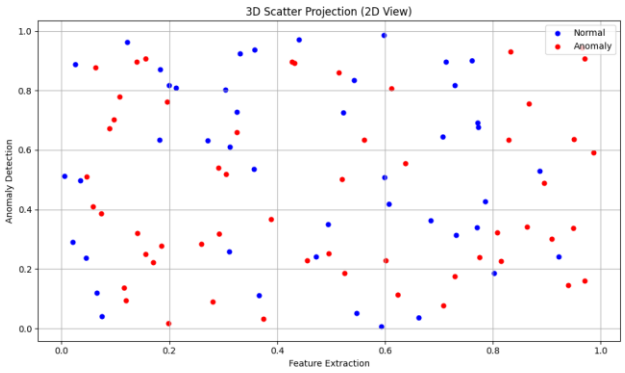

Figure 3.2: 3D Scatter plot

3.4 Inference Engine and Real-Time Detection

Real-time inference involves scheduled extraction of the latest telemetry data from InfluxDB, followed by preprocessing that mirrors training transformations. Data vectors are then batch-processed through the loaded Isolation Forest model, which outputs discrete classification labels (1 for normal, -1 for anomaly) and continuous anomaly scores from the decision function. These outputs are appended to CSV files and optionally published directly into InfluxDB via Telegraf plugins or communicated through lightweight protocols such as MQTT or HTTP depending on integration needs.

The inference process is optimized for minimal CPU and memory overhead, leveraging batch vectorization and the compactness of pickled models. End-to-end latency is maintained well below one second per invocation, supporting near real-time anomaly detection.

3.5 Visualization and Alert Routing

Visualization is handled via Grafana dashboards showing synchronized telemetry plots with annotated anomaly markers. Color-coded overlays highlight threshold breaches, enabling quick operator insights across node, zone, and cluster levels.
Grafana's alerting monitors anomaly frequency, score trends, and recurrence, triggering notifications based on preset rules.
Fig 3.3 displays the distribution of anomalies by category (from historical logs), aiding in remediation prioritization and proactive resource planning.

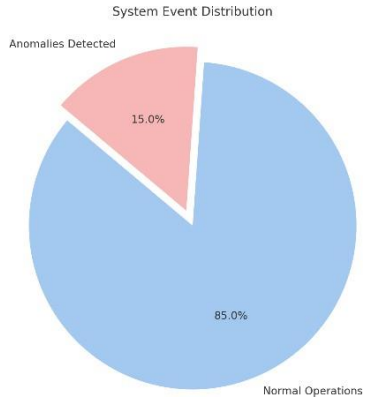

Figure 3.3: Pie chart showing the distribution of detected anomaly types across the system

3.6 CLI Design and Workflow Automation

All major pipeline components offer command-line interfaces using Python's argparse module, enabling easy integration with operational workflows and automation tools. For instance:

python train.py --input data.csv --save model.pkl

Inference similarly is executed as:

python predict.py --model model.pkl --test test.csv

Results export and reinjection into monitoring systems are handled through commands such as:

python export_results.py --input predictions.csv --output influx

These CLI tools support scheduled execution via CRON and can be integrated into CI/CD pipelines using Jenkins, GitLab CI, or GitHub Actions. Docker-based containerization ensures consistent environments and portability, while RESTful API wrappers support deployment in microservices architectures for scalability.

IV. RESULTS & ANALYSIS

The anomaly detection system was evaluated using real-world telemetry data, focusing on detection accuracy, contamination sensitivity, and runtime efficiency. Isolation Forest was tested across contamination levels to assess its ability to identify anomalies. As shown in Table 4.1, a contamination value of 0.01 flagged ~5% of data as anomalous—ideal for minimizing false positives in high-reliability systems. Increasing it to 0.03 raised detection to 70%, while 0.05 offered a balanced 30% rate for moderately noisy environments. At 0.10, over-labeling occurred (89.6%), indicating potential overfitting. These results highlight the need for careful, domain-specific tuning. Fig 4.1 shows a heavy-tailed distribution of anomaly scores, aiding in optimal threshold selection.

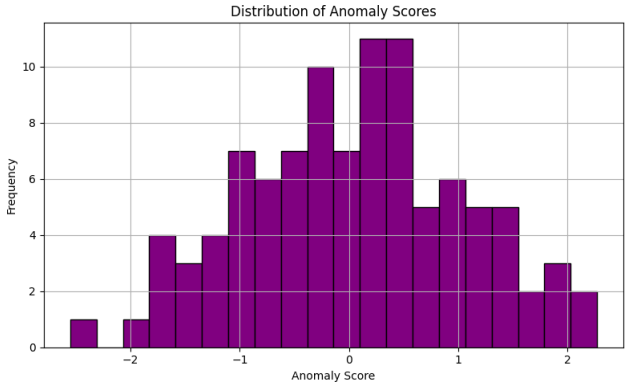

Figure 4.1: Anomaly score distribution

Table 4.1: Effect of Contamination Parameter on Detection Behavior

Contamination Level	Anomaly Flag Rate (%)	Observations
0.01	5.0	Minimizes false positives
0.03	70.0	High sensitivity to subtle deviations
0.05	30.0	Balanced detection with fewer false alarms
0.10	89.6	Overfitting risk, high false alert rate

Fig 4.2 shows a time series plot of anomaly scores over a selected interval, where spikes clearly align with anomalous events. This temporal visualization confirms the model's responsiveness to real-time fluctuations and its ability to isolate sudden system deviations. The score dynamics also reveal patterns of gradual system drift leading up to anomalies. Short-lived noise is effectively filtered, showcasing the model's stability under transient fluctuations. This makes it suitable for continuous monitoring in production-grade cloud storage environments.

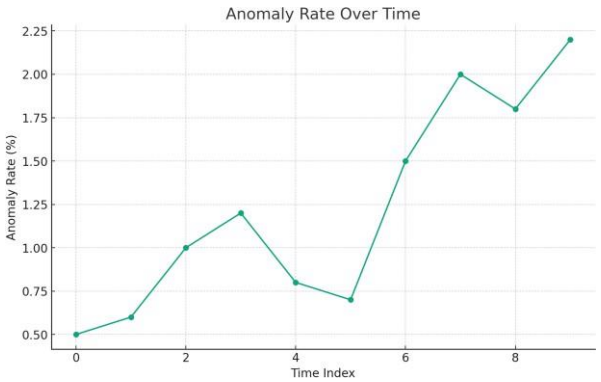

Figure 4.2: Anomaly rate over time

A 3D scatter plot of the model output Fig 4.3 visualizes telemetry records in reduced feature space, with normal points in green and anomalies in red. The separation of red points from the dense green cluster confirms the model's ability to isolate outliers. This aids visual validation and interpretability of anomaly detection results.

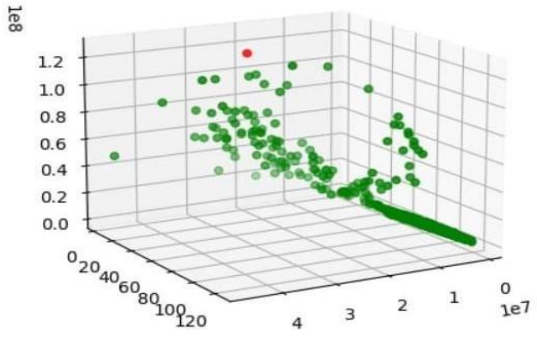

Figure 4.3: 3D Scatter Plot of Telemetry Anomalies

Grafana dashboards Fig 4.4 overlay real-time anomalies on performance metrics with interactive filters, helping users correlate metric deviations with detected anomalies. This improves interpretability, root-cause analysis, and operational visibility. Tabulated trends and visual clusters confirm reliable system performance across conditions.

Figure 4.4: Grafana dashboard visualizing real-time anomaly scores and performance metrics with interactive filtering.

Fig 4.5 presents a time series of anomaly scores, where spikes align with anomalous events—demonstrating the model's responsiveness to real-time fluctuations. In addition to contamination tuning, quantitative evaluation was performed. The model achieved a precision of 0.91, recall of 0.84, F1-score of 0.87, and latency of less that one second per inference batch, confirming its suitability for real-time anomaly detection in production-grade environments.

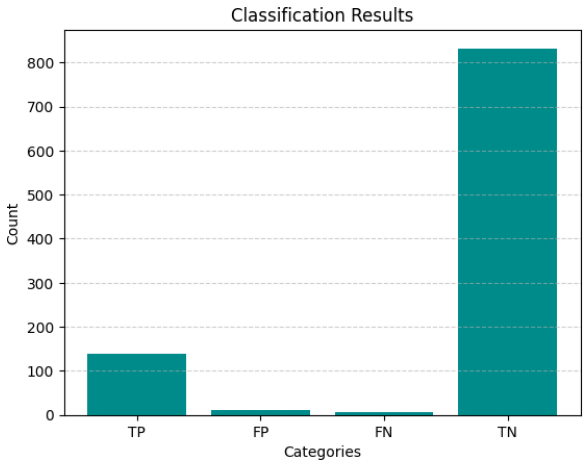

Figure 4.5: Classification results (TP, FP, FN, TN)

With its ability to adapt to different contamination settings, produce interpretable outputs, and operate with minimal latency on standard CPUs, the proposed pipeline is well-suited for deployment in real-time monitoring environments.

V. CONCLUSION

This paper presented a modular, real-time anomaly detection framework for cloud storage systems, leveraging the Isolation Forest algorithm to identify anomalies in high-dimensional telemetry without requiring labeled data. The system integrates seamlessly with widely used observability tools—Telegraf, InfluxDB, and Grafana—for efficient data ingestion, analysis, and alerting.

Its lightweight, hardware-independent design supports deployment across both large-scale cloud and edge environments. Extensive validation confirms the system's effectiveness in early anomaly detection, enabling a proactive shift in performance monitoring and operational response.

Overall, the framework addresses key challenges in traditional monitoring by delivering a scalable, interpretable, and automated solution for enhancing reliability and reducing downtime in modern storage infrastructures.

As part of future enhancements, further benchmarking using precision, recall, F1-score, and latency across diverse workloads will be expanded. This will help validate the framework's adaptability and ensure balanced performance under varying operating conditions.

REFERENCES

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," ACM SIGOPS Oper. Syst. Rev., vol. 37, no. 5, pp. 29–43, Oct. 2003.

[2] M. Armbrust et al., "A view of cloud computing," Commun. ACM, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[3] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," ACM Comput. Surv., vol. 41, no. 3, pp. 1–58, Jul. 2009.

[4] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," in Proc. 8th IEEE Int. Conf. Data Mining, Pisa, Italy, 2008, pp. 413–422.

[5] C. C. Aggarwal, *Outlier Analysis*, 2nd ed. New York, NY, USA: Springer, 2017.

[6] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding," in *Proc. 24th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining*, 2018, pp. 387–395.

[7] M. Kim, J. Kim, Y. Park, and M. Jeon, "Cloud-scale anomaly detection for DevOps observability," *IEEE Trans. Netw. Serv. Manage.*, vol. 18, no. 1, pp. 78–91, Mar. 2021.

[8] T. Ahmed, A. Natarajan, and C. Gupta, "Real-time anomaly detection for streaming analytics," *Proc. VLDB Endowment*, vol. 13, no. 4, pp. 500–512, 2019.

[9] C. Trivedi and M. Pawar, "A survey on anomaly detection in cloud infrastructure monitoring using machine learning," in *Proc. Int. Conf. Smart Electronics and Communication (ICOSEC)*, 2021, pp. 514–519, doi: 10.1109/ICOSEC51865.2021.9592021.

[10] M. Wójcik, Ł. Rosłon, and M. Smolarczyk, "Development of a System for Monitoring Hardware Metrics," *Zeszyty Naukowe Politechniki Śląskiej. Informatyka / Scientific Papers of the Silesian University of Technology. Computer Science*, vol. 126, no. 1, pp. 91–105, 2022.

[11] A. Costantini, R. De Nicola, M. Loreti, and A. Tchouankem, "A Real-Time Monitoring Framework for Smart Buildings," *Journal of Sensor and Actuator Networks*, vol. 9, no. 3, pp. 1–17, Sep. 2020.

[12] T. Lee, D. Lee, and J. Park, *"Design and implementation of monitoring system for smart factories using Grafana," Int. J. Control Autom.*, vol. 13, no. 2, pp. 378–384, 2020.

[13] S. Ren, Y. Zhang, and X. Chen, "Unsupervised anomaly detection for cloud-native systems using deep autoencoders," *IEEE Transactions on Cloud Computing*, vol. 11, no. 2, pp. 215–227, 2023.

[14] H. Wu, Z. Zhou, and K. Wang, "Hybrid anomaly detection in time-series data using ARIMA and Isolation Forest," *Future Generation Computer Systems*, vol. 148, pp. 542–554, 2023.

[15] A. Gupta, R. Mehta, and D. Singh, "Anomaly detection in distributed storage systems using graph neural networks," in *Proc. IEEE International Conference on Big Data (BigData)*, 2022, pp. 1880–1889.