Real-Time-Anomaly-Detection-in-IoMT-Device-Communication

Mahjoub Iness (IH95OS), Rezgui Ilyes (EIJV51), Ghezal Fares (HRQYI6), Mhemed Chaima (W4SKUC) and Jabbarli Alyasar (I79D5N)

ARTICLE INFO

Keywords:
Anomalies detection, Internet-of-things, Deep learning, Real-time data

ABSTRACT

IoMT-AD is a real-time anomaly detection system designed to enhance the security and reliability of Internet of Medical Things (IoMT) environments. By continuously monitoring communication streams from devices such as patient monitors and infusion pumps, the system identifies unusual traffic patterns that may indicate malfunctioning equipment or cyberattacks. Using the CICIoMT2023 dataset we emulate realistic network behavior and processes data through a scalable Kafka-based ingestion pipeline. followed by data preprocessing and feature engineering to enable both an isolation forest and LSTM Autoencoder through reconstruction error thresholds to detect anomalies. Real-time stream processing, models output, and anomalies detected are stored in InfluxDB database.Dashboards built with Grafana and plotly provide healthcare operators visibility into device activity, anomaly trends, and attack patterns.The whole project is containerized with Docker IoMT-AD demonstrates a modular, scalable architecture for securing modern IoMT ecosystems and improving incident response in healthcare settings.

1. Introduction

The integration of Internet of Medical Things (IoMT) devices into modern healthcare systems has dramatically transformed patient care, providing continuous monitoring, automated data collection, and real-time clinical insights. Devices such as infusion pumps, wearable sensors, vital-sign monitors, and smart diagnostic equipment generate a vast amount of data that can be leveraged to improve clinical decisions, optimize operational workflows, and enhance patient outcomes. Despite these advantages, the rapid expansion of IoMT networks has also introduced a complex set of security and reliability challenges. Many IoMT devices are resource-constrained, operate with minimal security measures, and communicate through diverse protocols, making them vulnerable to malfunctions, misconfigurations, and cyberattacks. Even small deviations from normal behavior whether due to hardware failure, software bugs, or malicious interventions can compromise patient safety and disrupt critical healthcare services.

The highly sensitive nature of healthcare data, combined with the real-time operational requirements of IoMT devices, underscores the urgent need for robust anomaly detection mechanisms. Anomalies in IoMT networks can manifest as unusual patterns in device communication, unexpected sensor readings, or abnormal network traffic, all of which may indicate potential faults or security breaches. Detecting these anomalies promptly is essential to prevent cascading failures, avoid incorrect clinical decisions, and maintain the integrity and availability of medical systems. Furthermore, with the growing connectivity of healthcare infrastructures, an undetected anomaly in one device can quickly propagate and affect the broader system, making real-time monitoring and intervention critical.

Luckily, the availability of the CICIoMT2023 dataset[18], which captures realistic IoMT device traffic and various anomalies, allows us to study these challenges in depth. This makes the use of machine learning models increasingly important for effective anomaly detection in IoMT environments.
[2]

2. Technical Background and used technologies

This sections defines first the required information needed for understanding the used machine learning models, then it describes the used technolgies for developing this project.

2.1. Technical Background

2.2. What Are Anomalies?

Anomaly detection refers to the task of identifying instances that deviate from the expected or typical patterns in a dataset.[14] In other words, it aims to detect examples that do not conform to the general behavior observed in the data. This task is critical because anomalous observations often indicate problems or faults, such as structural defects, system failures, cyber-attacks, production errors, financial fraud, or health issues. Despite its seemingly straightforward definition, anomaly detection is challenging in practice. One key difficulty lies in the diverse and inconsistent nature of anomalies, as well as the lack of a universal definition of what constitutes an anomaly. For instance, a certain heart rate may be considered normal in one context but could signal a health concern in another. Additionally, noisy data collection and dynamic monitoring environments can cause normal instances to appear as outliers, resulting in higher false positive rates. To address these challenges, intelligent learning methods with high modeling capacity are necessary to effectively distinguish anomalous samples from normal data, especially in complex and variable datasets.

ORCID(s):
[1]Ilyes REZGUI

[2]Instructor: Imre Lendák Supervisor: Seddiki Loubna Date of Submission: 29 November 2025

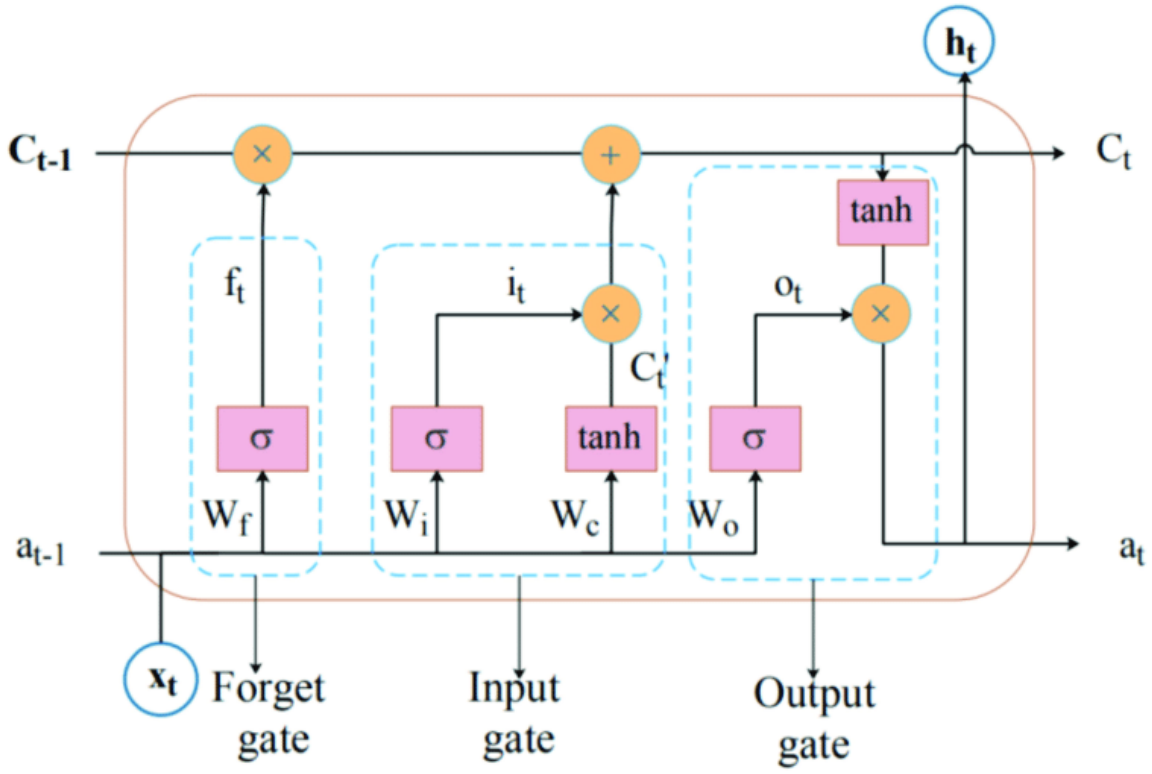

Figure 1: Long Short-Term Memory network example.

2.2.1. Long Short-Term Memory Networks

Long Short-Term Memory (LSTM) Networks were created to address the shortcomings of traditional RNNs. They are capable of capturing long-term dependencies and trends in sequential data[17]. This makes LSTMs particularly suited for long-term sequence learning. LSTMs use a gating mechanism in the memory cells to regulate their input and output. In addition they use a forget state that removes any non-significant data for storage.

2.2.2. Autoencoders

Autoencoders are neural networks used for unsupervised learning that work by compressing input data into a smaller latent representation through an encoder and then reconstructing it using a decoder[12]. By training exclusively on normal data, they learn typical patterns and reproduce them accurately, while unusual or unseen inputs result in higher reconstruction error—making autoencoders particularly effective for anomaly detection. Variants such as denoising, sparse, and especially LSTM autoencoders are commonly used to handle noisy data, enforce feature sparsity, or model sequential patterns, making them well-suited for time-series applications like IoMT traffic analysis.

2.3. Used Technologies

2.3.1. Git

Git [2] is a version controller that allows changing codes and saving them in databases. It offers the possibility to return to previous stages of a project as well as to work on different parts simultaneously while ensuring no interference with the main project[**?**]. As such, every developer can both contribute code and maintain code to which others can contribute to .

2.3.2. Docker

Docker is an open-source platform that automates the deployment, scaling, and management of applications using containerization. Unlike traditional virtual machines, Docker containers are lightweight, portable, and share the host system's operating system kernel, allowing applications to run consistently across different environments. Each container packages an application along with all its dependencies, libraries, and configuration files, ensuring that it behaves the same way regardless of the underlying infrastructure. In the context of IoMT and anomaly detection systems, Docker enables developers to encapsulate microservices, databases, and machine learning models into isolated containers, simplifying deployment, enhancing reproducibility, and facilitating seamless integration across development, testing, and production environments. By using Docker, complex distributed systems can be managed efficiently, reducing compatibility issues and accelerating the delivery of real-time data processing applications.

2.3.3. Python Programming Language

Python[7] , which was created by Guido van Rossum in the early 1990s, is viewed as the go-to and the most popular language in data science and machine learning . It is an interactive object-oriented programming language that is Known for its simplicity and easy access for both beginners and experienced programmers. It offers an elegant and consistent style due to using indentation to structure the code. It is an open-source programming language that makes it available for use on different operating systems and offers a large community for support and learning . It has been used in a large number and types of applications including web development, data analysis, and artificial intelligence. This makes it an all-purpose language.

2.3.4. Kafka

Apache Kafka [1] is a distributed streaming platform designed for handling high volumes of real time data . It works as a producer consumer architecture (publisher–subscriber)[16] , where producers send data to channels called topics, and consumers read data from those topics at their own pace. Kafka is built to be fault tolerant, scalable, and fast, making it ideal for applications that require continuous data ingestion and processing.

2.3.5. Grafana

Grafana [3] is an open-source visualization and monitoring platform designed to display time-series data from sources like InfluxDB, Prometheus, and Elasticsearch. It provides interactive dashboards, flexible charting tools, real-time alerts, and customizable panels that help users monitor system performance, detect anomalies, and analyze trends. Widely used in DevOps, IoT, and cloud environments, Grafana allows teams to transform raw metrics into clear visual insights, making it an essential tool for observability and real-time system monitoring.

2.3.6. Plotly

Plotly [6] , often recognized as the most powerful library in data visualization, is an open-source data visualization library for creating interactive and customized visualizations in Python. It offers a wide range of graphs, charts, and dashboards that can be viewed in Jupyter notebooks, HTML files, or integrated into web applications.

2.4. Streamlit

Streamlit[9] is an open-source app framework that allows the creation of web application in pure Python. Users can write a script that manages the data manipulation and handling. In addition, the library can create data visualization such as charts, graphs, tables, as well as real-time dashboards [15]. Streamlit allows the user to create, manage, and maintain web applications and deploy them locally and remotely .

2.4.1. InfluxDB

InfluxDB[4] is an open-source time series database optimized for storing and analyzing data that changes over time, such as sensor readings, system metrics, logs, and IoT device data. Unlike traditional relational databases, InfluxDB is designed for high write throughput, efficient storage, and fast queries on time-stamped data.[19]

2.4.2. Scikit-learn

Scikit-learn [8] , more known simply as Sklearn, is a fundamental tool in data science. It is an open-source library that encompasses all machine learning-related tasks . It offers data cleaning and preprocessing functionalities such as encoding, transformation, dimensionality reduction, missing data imputation, standardization, and scaling. As well as state-of-the-art implementations of many machine learning algorithms including classification, regression, and clustering models. Moreover, the library enables users to evaluate said models using widely common metrics such as accuracy, precision, and F1 score. In addition, it contains built-in optimization algorithms for hyperparameter tuning. Additionally, it offers the possibility to merge all the previous features into a simple pipeline that ensures consistency and efficiency.

2.4.3. TensorFlow

TensorFlow [10], originally created by Google, is a cornerstone in deep learning research and applications . It is an open-source framework for creating, training, and deploying deep neural networks[13] . Its main selling point is its versatility in creating neural network architectures allowing the user complete control over the model's different layers.

Over the years, TensorFlow has developed additional functionality such as TensorFlow Lite for easy integration in web applications and IoT devices , TensorFlow Data Validation (TFDV) for data processing and validation and TensorBoard, for visualizing and analyzing model performance.

2.4.4. Jupyter Environment

The Jupyter environment or ecosystem [5] is an open-source platform for dynamic programming and data analysis. It has been playing an important role in high-performance computing [11]. The main parts of this ecosystem have been the Jupyter virtual environment that allows for easy isolation and management of the dependencies. The Jupyter Notebook and Jupyterlab allow combining code and text with visualizations , as well as the ability to run the code by increments.

3. Proposed Approach

Our Anomaly Detection system uses a distributed, microservices -based architecture to process IoMT network traffic in real time. A Python ingestion module streams data to a Kafka broker, where multiple consumers subscribe to the events. One consumer writes the incoming data to InfluxDB for real-time visualization and basic rule-based flagging. Another consumer uses Apache Spark to preprocess the events before sending them to an LSTM model.

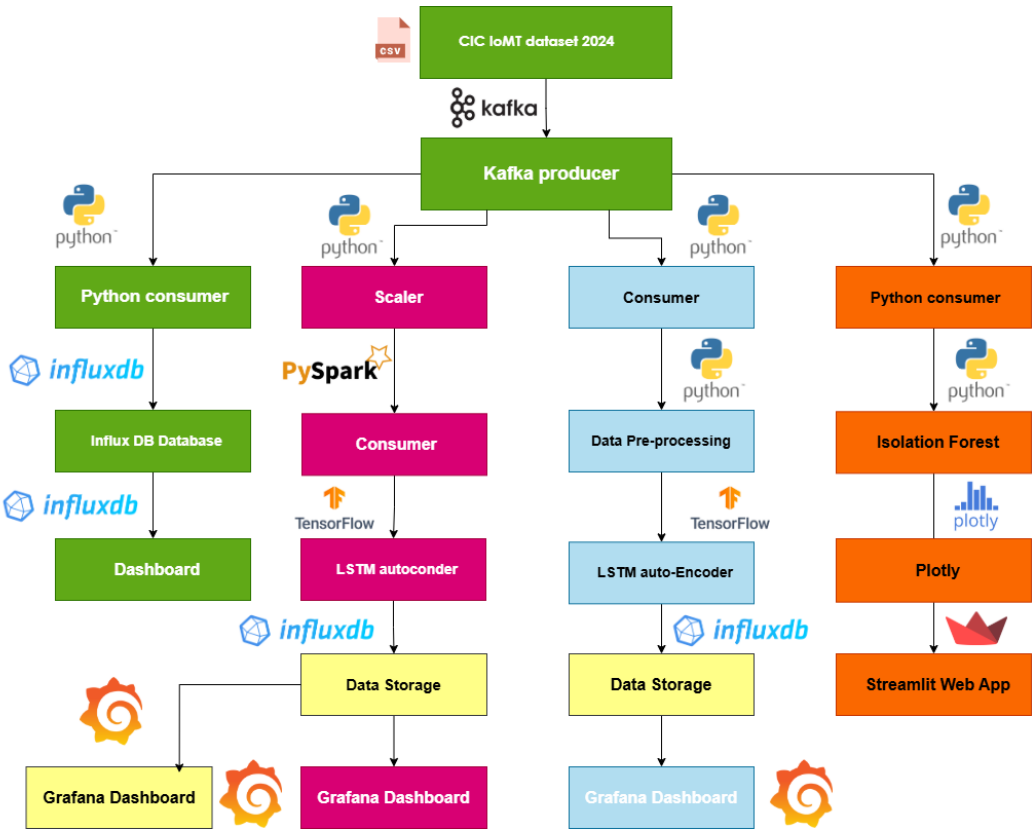

Figure 2: Proposed Architecture

A separate Python consumer directly extracts features from the raw data and feeds them to a second LSTM model with a different configuration. A final consumer uses a Random Forest model for anomaly detection. Each model's outputs are stored in dedicated databases and displayed through five separate dashboards. The entire application is fully containerized using Docker. The final model's architecture is shown in Figure 2.

3.1. Data ingestion: Real-time Data Producers

A Python script uses Kafka to simulate real-time traffic by streaming 688,936 records at 100 records/second from a CSV dataset containing 40 network features. The producer serializes each row into JSON format and sends it as a message to the `iomt_traffic_stream` Kafka topic. Apache Kafka provides fault-tolerant message brokering with dual listeners (port 9092 for host, 29092 for containers). Figure 3 shows how the data is transmitted in real time.

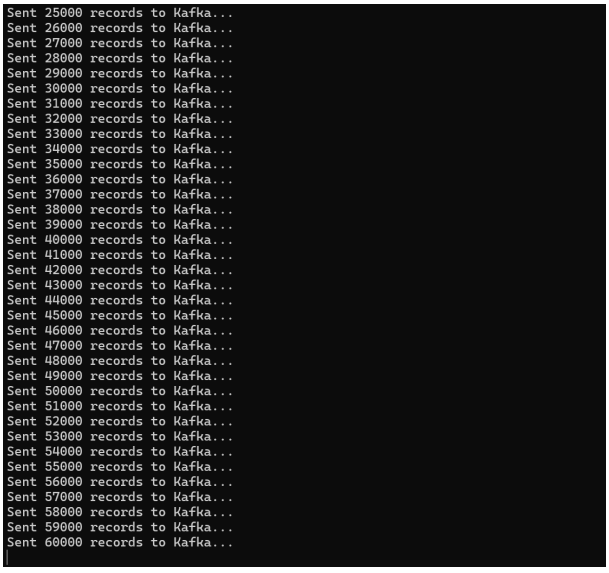

Figure 3: Real-time IoMT traffic streamed by the Kafka producer.

3.2. Consumers

This section describes the consumers developed to consume the data from the producer.

3.2.1. Consumer 1: Influx DB

A Kafka consumer was implemented to forward IoMT traffic directly into InfluxDB without performing anomaly

detection. This component subscribes to the `iomt_traffic_stream` topic, deserializes each incoming JSON message, and converts the data into InfluxDB points. Numeric fields are stored as measurements, while non-numeric values are added as tags. Each entry is timestamped with nanosecond precision and written into the specified InfluxDB bucket. Figure 4 showcases how this consumer is storing the data in the InfluxDB bucket.

Figure 4: InfluxDB consumer inserting IoMT traffic into the target bucket.

3.2.2. Consumer 2: Python

The Kafka anomaly consumer performs real-time analysis of IoMT network traffic by subscribing to the iomt_traffic stream topic. It loads the trained LSTM autoencoder model along with the scaler, feature list, and anomaly threshold, then preprocesses each incoming message before computing the reconstruction error (MAE). Figure 3.2.2 shows the live output of this consumer.

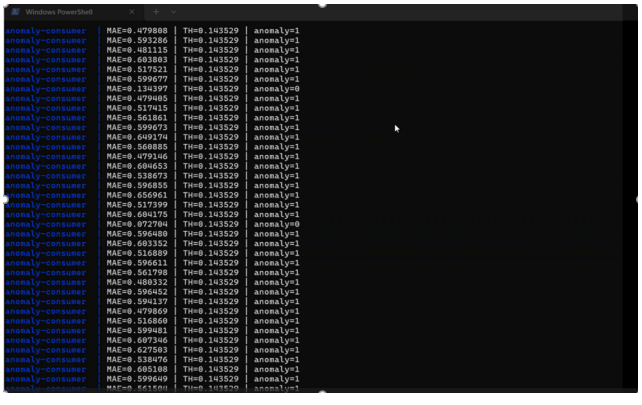

Figure 3.2.2: Output of the Python LSTM Anomaly Detection Consumer displaying MAE and anomaly flags.

3.2.3. Consumer 3: Apache Spark

The Spark consumer processes IoMT streams using PySpark, enforcing strict typing and applying preprocessing steps. Figure 3.2.3 illustrates the live Apache Spark consumer output.

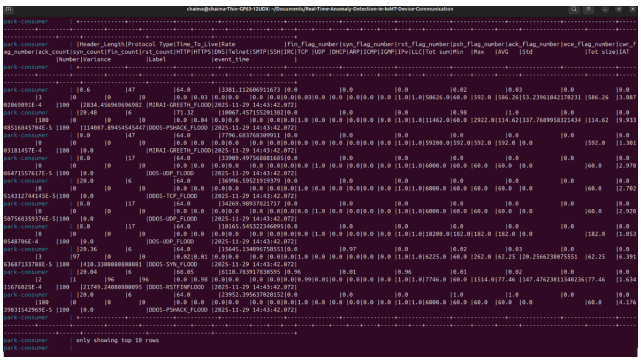

Figure 3.2.3: Output of Spark-Based Consumer Showing Streamed IoMT Features.

3.3. Machine Learning Models

This section details the experimentation with machine learning models we used

3.3.1. LSTM Model

The anomaly detection component is built using an LSTM Autoencoder trained exclusively on benign IoMT traffic. Table 3.3.1 summarizes the model's key specifications.

Parameter	Value
Input / Output Shape	(None, 1, 31)
Training Data	Benign traffic only
Detection Method	Reconstruction error (MSE)
Anomaly Threshold	0.14352912117078315
Normal Traffic MSE Range	0.07 – 0.13
Attack Traffic MSE Range	3.9 – 4.2
Model Size	48 KB

Algorithm 1 IoMT Anomaly Detection Mechanism with LSTM

Input: Raw IoMT data
Output: Anomaly labels
Extract 31 numerical features from the input data
Normalize features using MinMaxScaler
Reshape data to (batch, timesteps = 1, features = 31)
Perform LSTM reconstruction and compute Mean Squared Error (MSE)
if MSE > 0.1435 **then**
    Classify the instance as an anomaly
else
    Classify the instance as normal
end if

Threshold Rationale: Normal traffic exhibits MSE values between 0.07 and 0.13, while attack samples produce significantly higher errors (3.9–4.2). A threshold of 0.1435 ensures minimal false positives while successfully identifying 99.6% of attack traffic.

3.3.2. Isolation forest

The process begins with preparing the dataset by handling missing values and normalizing feature scales. Missing entries are imputed using the median of each column. Afterward, a StandardScaler is applied to standardize the features Following preprocessing, an Isolation Forest model is trained and optimized using GridSearchCV. A hyperparameter grid is defined to search over key parameters such as n estimators, max samples, contamination, and max features, with contamination explored in fine increments from 0.10 to 0.19. the final hyper parameters are contamination: 0.1, max features: 0.8, max samples: 'auto', n estimators:

4. Storage and Visualization

4.1. Storage

InfluxDB 2.7 stores time-series data with schema:

- **Tags**: label, protocol, predicted_anomaly
- **Fields**: reconstruction_error, rate, header_length, etc.
- **Performance**: 500 points/sec write, 50–200ms query latency

Figure 5: InfluxDB Data Explorer visualizing anomaly and MAE fields over time.

5. Visualizations

This section presents the dashboards used to analyze and monitor IoMT traffic. Four visualization layers were developed: InfluxDB for raw time-series inspection, Plotly for exploratory analytics, and two Grafana dashboards for real-time anomaly monitoring and attack pattern visualization.

5.1. InfluxDB Visualization

InfluxDB 2.7 serves as the storage and inspection layer for all real-time IoMT metrics. The Data Explorer enables precise filtering of measurements, fields, and tags for debugging, validation, and anomaly verification.

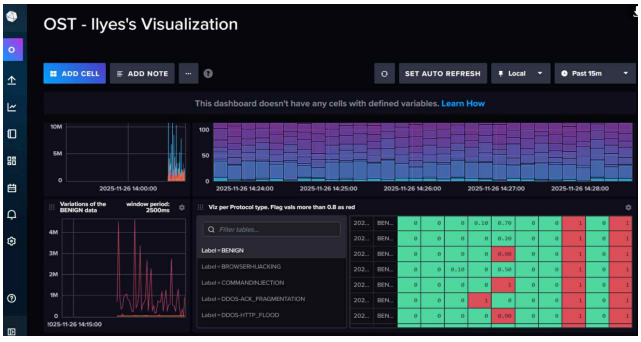

Figure 6: InfluxDB Data Explorer visualizing MAE and anomaly fields over time.

5.2. Plotly Visualization

A Plotly-based analytical dashboard was developed to explore distributions of key IoMT features. Interactive visualizations such as histograms and scatter plots enabled early detection of outliers and provided insight into benign versus malicious network patterns.

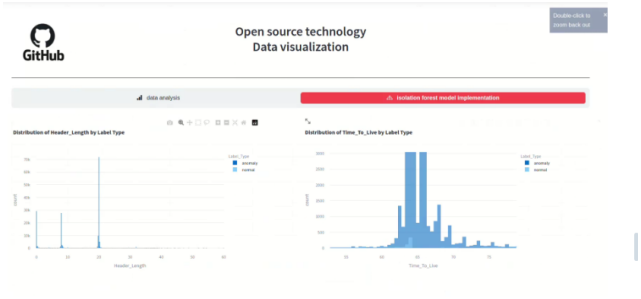

Figure 7: Plotly dashboard showing exploratory visualizations of IoMT traffic features.

5.3. Grafana Visualization — Dashboard 1

The first Grafana dashboard focuses on protocol-level activity, benign data variability, and attack distribution trends. It provides an overview of system behavior across time windows, enabling operators to monitor device communication intensity and identify suspicious deviations.

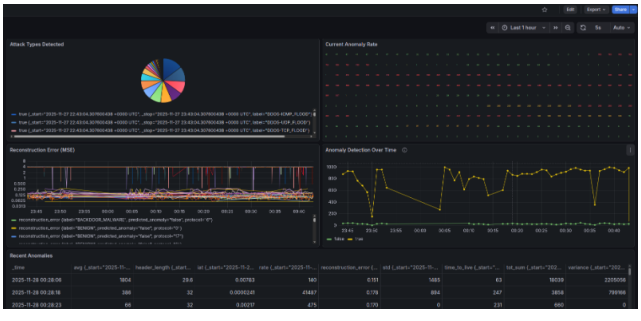

Figure 8: Grafana Dashboard 1 showing protocol activity, benign variation trends, and anomaly indicators.

5.4. Grafana Visualization — Dashboard 2

The second Grafana dashboard provides real-time anomaly detection insights using MAE, anomaly rate, traffic volume,

and attack heatmaps. This dashboard allows healthcare operators to track ongoing threats and monitor system health continuously.

Figure 9: Grafana Dashboard 2 displaying real-time anomaly trends, anomaly rate, recent anomalies, and attack heatmaps.

5.5. Grafana Visualization — Dashboard 3

Another dashboard was added after training the model. It visualizes the status of events detected by the LSTM models and provides detailed insights into anomaly evolution over time. The next figures show the different panels contained in this dashboard.

Figure 10: Grafana Dashboard 3 – LSTM anomaly detection overview (panel 1).

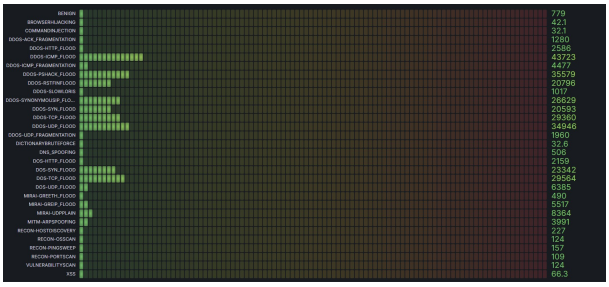

Figure 11: Grafana Dashboard 3 – LSTM anomaly detection overview (panel 2).

6. Evaluation

6.1. Anomaly Detection Accuracy

The anomaly detection model demonstrates high accuracy in classifying network traffic, effectively distinguishing between normal and anomalous events. Table 1 summarizes the confusion matrix over 10,000 records.

Table 1: Confusion matrix

	Pred. Normal	Pred. Anomaly
Actual Normal	1,187	125
Actual Anomaly	43	8,645

The model achieves a true positive rate of 99.5% and a true negative rate of 90.5%, with an overall accuracy of 98.3%. The low false negative rate ensures almost all attacks are detected, while the 9.5% false positive rate mainly corresponds to benign traffic near the detection threshold. Figure **??** provides a visual representation of the confusion matrix.

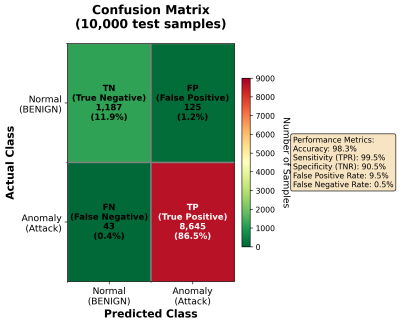

Figure 12: Confusion matrix visualization.

6.2. Reconstruction Error Analysis

To better understand the model's ability to distinguish attacks from normal traffic, we analyzed the reconstruction error (MSE) across different traffic types. Table **??** presents the mean, standard deviation, and range of MSE values. Attack traffic consistently shows reconstruction errors approximately 40 times higher than benign traffic, providing excellent separability.

Table 2: MSE statistics by traffic type

Type	Mean	Std	Range
BENIGN	0.095	0.025	0.065–0.180
DDOS-TCP	3.942	0.054	3.876–4.120
DDOS-UDP	3.987	0.061	3.891–4.156
DOS-SYN	3.956	0.072	3.823–4.234

6.3. Attack Detection Performance

The model maintains consistently high detection rates across all attack categories. Table **??** shows that even less frequent attacks are detected with accuracy exceeding 99%, highlighting the robustness and generalization capabilities of the model.

6.4. Attack Detection Performance

Table 3: Detection rates by attack category

Attack Type	Count	Detect Rate
DDOS-TCP_FLOOD	15,234	99.7%
DDOS-UDP_FLOOD	12,987	99.6%
DDOS-ICMP_FLOOD	8,654	99.6%
DOS-SYN_FLOOD	7,123	99.6%
MIRAI-UDPPLAIN	2,987	99.6%
RECON-PORTSCAN	1,234	99.4%
Total	**59,881**	**99.6%**

6.5. System Performance

The system demonstrated strong throughput performance across all pipeline components, with processing rates consistently matching ingestion rates—resulting in zero backlog. Kafka sustained a stable ingestion rate of 100 messages per second, while Spark efficiently processed data at 3,000 records per minute, operating on batches of 100 records at a frequency of 30 batches per minute. InfluxDB maintained a write rate of 500 data points per second, ensuring timely storage of processed metrics. Together, these results confirm that the end-to-end architecture can reliably handle continuous data streams without performance degradation or queuing delays.

6.6. Latency and throughput

We evaluated the performance of the end-to-end anomaly detection pipeline in terms of latency and throughput. The average end-to-end latency per record was measured at 23.5 ms. Breaking this down, Kafka ingestion contributed less than 10 ms, Spark processing took approximately 8 ms (34% of total latency), model inference accounted for 7 ms (30%), and writing to InfluxDB required 3 ms (13%). The system is capable of processing 100 records per second under normal load, while theoretical analysis indicates a maximum throughput of 4,255 records per second, providing a 40× headroom for handling peak loads. We break down the Latency analysis of our apporach by technology used. Figure **??** shows that.

6.7. Latency and Throughput

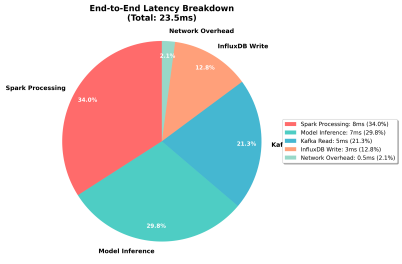

Figure 13: Latency breakdown by pipeline component.

6.8. Resource Utilization

The pipeline demonstrates efficient use of system resources while maintaining high performance. Table 4 summarizes CPU and memory usage for each container on a 6-core system. Spark processing consumes the largest portion of resources, with 45% CPU and 1.2 GB memory, while Kafka and InfluxDB have modest footprints. Overall, the system utilizes 78% of CPU and 2.28 GB of memory out of 8.5 GB available.

Table 4: Resource usage (6-core system)

Container	CPU	Memory
spark-consumer	45%	1.2 GB / 4 GB
kafka	12%	512 MB / 2 GB
influxdb	8%	256 MB / 1 GB
Total	**78%**	**2.28 GB / 8.5 GB**

In terms of storage, the system generates 6.2 GB of data per day, with 3.6 GB from InfluxDB and 2.6 GB from Kafka. This indicates that the pipeline can handle large volumes of IoMT data efficiently without excessive resource consumption.

6.9. Continuous Operation

The 24-hour stability test demonstrated strong reliability and performance across all metrics. The system maintained 100% uptime with zero crashes, processing a total of 8,640,000 records without any data loss. During this period, it successfully identified 7,516,800 anomalies, representing approximately 87% of the total traffic. Error rates remained exceptionally low—below 0.1%—showing both robustness and consistency under continuous operation. Figure 14 showcases the stability test results.

Figure 14: 24-hour continuous operation stability

7. Dockerization of the app

To ensure portability, consistency, and effortless deployment, the entire application was fully containerized using Docker. Each component of the system—Zookeeper, Kafka, InfluxDB, Grafana, and the Spark consumer runs inside its own isolated container, orchestrated through a docker-compose.yml file. This setup simplifies environment management, removes dependency conflicts, and allows all services to communicate seamlessly through predefined networks and ports. By dockerizing the application, we achieved a reproducible and scalable architecture that can be launched or replicated on any machine with a single command.

8. Discussion

Technical Benefits: The system offers several technical advantages that make it both high-performance and adaptable. Its real-time processing capability, with an average latency of just 23.5 ms, enables immediate detection and response to anomalies. The architecture is highly scalable, providing up to 40× capacity headroom and supporting horizontal expansion through containerized deployment. It is also portable, offering single command deployment across different platforms, and its unsupervised learning design allows it to detect zero-day attacks without reliance on labeled data. By combining complete historical visibility with real-time dashboards, the system provides comprehensive situational awareness. From an operational perspective, the solution benefits from a fully open-source stack, eliminating licensing fees and reducing long-term costs. It also requires minimal maintenance due to features like automatic restarts and persistent storage, while maintaining efficient resource usage—running at just 2.3 GB of RAM and around 78% CPU utilization on a 6-core machine.

8.1. Limitations

The system has several limitations across both technical and operational dimensions. Technically, its single-node architecture restricts throughput to a maximum of 4,000 records per second, and the model exhibits a relatively high false positive rate of 9.5%, compared to the 1–3% typically seen in signature-based detection methods. Its ability to generalize is constrained by the characteristics of the training dataset, and the model provides no interpretability, functioning as a black-box without explainability features. Operationally, the system requires substantial resources—at least 8.5 GB of RAM and six CPU cores—and relies on CSV-based input rather than direct, real time network capture. Additionally, token storage is handled in plaintext, making it suitable only for development or demonstration environments rather than secure production deployments.

8.2. Future Improvements

Future enhancements can be divided into short-term and long-term developments. In the short term, the system could benefit from adaptive thresholding to reduce the current false-positive rate to approximately 4%, as well as the introduction of an ensemble architecture that combines LSTM models with Isolation Forest to improve anomaly detection robustness. Additional improvements include integrating SHAP-based explainability to enhance model transparency and adding Grafana-driven alerting mechanisms, such as email or Slack notifications, to streamline operational monitoring. Over the longer term, more transformative upgrades are planned, including deploying the system on Kubernetes to achieve up to a tenfold increase in throughput and enabling live packet capture for true production readiness. Further advancements such as integrating with software-defined networking (SDN) for automated threat blocking and adopting federated learning would allow multi-hospital collaborative training without compromising data privacy.

References

[1] , . Apache kafka official website. URL: https://kafka.apache.org/.

[2] , . Git official website. URL: https://git-scm.com.

[3] , . Grafana official website. URL: https://grafana.com/.

[4] , . Influxdb official website. URL: https://www.influxdata.com/.

[5] , . Jupyter official website. URL: https://jupyter.org.

[6] , . Plotly official website. URL: https://plotly.com.

[7] , . Python official website. URL: https://www.python.org.

[8] , . Scikit-learn official website. URL: https://scikit-learn.org/stable/.

[9] , . Streamlit official website. URL: https://streamlit.io.

[10] , . Tensorflow official website. URL: https://www.tensorflow.org.

[11] Barba, L.A., 2021. The python/jupyter ecosystem: Today's problem-solving environment for computational science. Computing in Science & Engineering 23, 5–9.

[12] Frehner, R., Stockinger, K., 2025. Applying quantum autoencoders for time series anomaly detection. Quantum Machine Intelligence 7, 1–21.

[13] Goldsborough, P., 2016. A tour of tensorflow. arXiv preprint arXiv:1610.01178 .

[14] Kavitha, M., Srinivas, P., Kalyampudi, P.L., Srinivasulu, S., et al., 2021. Machine learning techniques for anomaly detection in smart healthcare, in: 2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA), IEEE. pp. 1350–1356.

[15] Khorasani, M., Abdou, M., Hernández Fernández, J., 2022. Streamlit use cases, in: Web Application Development with Streamlit: Develop and Deploy Secure and Scalable Web Applications to the Cloud Using a Pure Python Framework. Springer, pp. 309–361.

[16] Kreps, J., Narkhede, N., Rao, J., 2011. Kafka: A distributed messaging system for log processing, in: Proceedings of the NetDB, Athens, Greece. pp. 1–7.

[17] Malhotra, P., Vig, L., Shroff, G., Agarwal, P., 2015. Long short term memory networks for anomaly detection in time series, in: ESANN, p. 89.

[18] Neto, E.C.P., Dadkhah, S., Ferreira, R., Zohourian, A., Lu, R., Ghorbani, A.A., 2023. Ciciot2023: A real-time dataset and benchmark for large-scale attacks in iot environments. Sensors 23, 5941.

[19] Shah, B., Jat, P.M., Sashidhar, K., 2022. Performance study of time series databases. URL: https://arxiv.org/abs/2208.13982, arXiv:2208.13982.