Received XX Month, XXXX; revised XX Month, XXXX; accepted XX Month, XXXX; Date of publication XX Month, XXXX; date of current version XX Month, XXXX.

Digital Object Identifier 10.1109/XXXX.2022.1234567

Classification of Anomalies in Microservices Using an XGBoost-based Approach with Data Balancing and Hyperparameter Tuning

Luís M. Barata[1,2], Member, IEEE, Eurico Lopes[2], Pedro R. M. Inácio[1], Senior Member, IEEE, AND Mário M. Freire[1,3], Member, IEEE

[1]Instituto de Telecomunicações, Department of Computer Science, Universidade da Beira Interior, 6201-001 Covilhã, Portugal
[2]Department of Computer Science, Escola Superior de Tecnologia, Instituto Politécnico de Castelo Branco, 6000-767 Castelo Branco, Portugal
[3]NOVA Laboratory for Computer Science and Informatics (NOVA LINCS), Department of Computer Science, University of Beira Interior, 6201-001 Covilhã, Portugal

Corresponding author: Luís M. Barata (email: luis.miguel.barata@ubi.pt).

This work is funded in part by FCT/MECI through national funds and, when applicable, co-funded by EU funds under UID/50008: Instituto de Telecomunicações; and in part by the WATERMARK project (Watermark-Based Algorithms for Trustworthy Media Authentication and Robust Certification in Public Administration), Project No. 2024.07356.IACDC, supported by "RE-C05-i08.M04 - Support the launch of a program of R&D projects aimed at the development and implementation of advanced systems in cybersecurity, artificial intelligence, and data science in public administration, as well as a scientific training program," under the Recovery and Resilience Plan (PRR), as part of the funding agreement signed between the Recovery Portugal Task Force (EMRP) and the Fundação para a Ciência e a Tecnologia (FCT); and also in part supported by UID/04516/NOVA Laboratory for Computer Science and Informatics (NOVA LINCS) with the financial support of FCT.IP.

ABSTRACT

Microservice architecture has emerged as a leading paradigm for decomposing large monolithic applications into smaller, autonomous services. Although this approach offers many advantages, its complexity, distributed nature, and substantial scale create significant challenges for monitoring and anomaly detection. The vast volume of generated data further exacerbates computational load and detection latency, complicating the identification of anomalies. This study analyses the impact of data balancing and hyperparameter tuning on anomaly classification in microservices and introduces *ADMXGB - Anomaly Detection in Microservices using XGBoost*, a XGBoost-based framework tailored for anomaly detection in microservices that seamlessly integrates data balancing with hyperparameter tuning. We propose guidelines to determine appropriate threshold values that balance sensitivity with false positives, and show that the framework is model-agnostic, enabling integration with different machine learning algorithms beyond XGBoost. Validation was performed using a four-stage process encompassing preprocessing, training, validation, and testing. ADMXGB demonstrated improvements in both Accuracy and F1-Score, reaching 99.96% in both metrics on the TraceRCA dataset, outperforming the baseline XGBoost method by a margin of 1.46% in Accuracy and 45.62% in F1-Score. Moreover, ADMXGB achieves reductions in execution time (-86.3%) and memory usage (-21.7%), while maintaining an acceptable CPU overhead. These findings highlight the robustness of ADMXGB in delivering high-accuracy classification in a microservice environment.

INDEX TERMS Anomaly Detection, Data Balancing, Oversampling, Undersampling, Hybridsampling, Microservices, Hyperparameter Optimization

I. INTRODUCTION

THE rapid adoption of cloud services, particularly Infrastructure as a Service (IaaS) and Platform as a Service (PaaS), has been strongly influenced by the emergence of Microservices Architecture (MSA) [1]. MSAs focus on developing modular services with clearly defined interfaces, allowing independent development, deployment, and scaling [2]–[4]. This design contrasts with monolithic architec-

tures and is widely supported by containerization platforms such as Docker and Kubernetes, which simplify orchestration and promote consistency across environments [5].

Microservice-based systems provide clear advantages in scalability and modularity, yet they present unique challenges for anomaly detection. Their distributed nature, high dimensionality, and interdependencies make faults difficult to localize and detect. Conventional rule-based monitoring often fails under such dynamic conditions [1].

To address these challenges, machine learning (ML) techniques have gained traction. However, anomaly detection in this context faces two persistent problems: the extreme class imbalance in telemetry data, where anomalies are rare, and the sensitivity of ML models to hyperparameter configurations. While data balancing strategies (e.g., oversampling, undersampling, hybridsampling) can mitigate bias [6], and hyperparameter optimization can enhance model performance, these techniques are often applied in isolation, and their computational cost is frequently overlooked.

This study addresses that gap by proposing a framework that jointly explores the impact of data balancing and hyperparameter tuning on anomaly detection in microservices, while also analyzing the execution footprint of these techniques.

This paper makes the following original contributions:

- We propose ADMXGB, an anomaly detection framework for microservices that aims to improve the detection accuracy and to reduce computational cost. The proposed framework introduces a configurable latency threshold to improve computational efficiency during preprocessing;
- We conduct an extensive evaluation on the TraceRCA dataset, where ADMXGB outperforms existing models in multiple evaluation metrics, including classification accuracy and computational efficiency;
- We implement a comprehensive training pipeline combining data balancing strategies (oversampling, undersampling, hybridsampling) with systematic hyperparameter optimization through Grid Search, Random Search, and Bayesian optimization (HyperOpt), validated via 10-fold cross-validation;
- We identify possible guidelines to determine appropriate threshold values (e.g., $\mu + 2\sigma$ to $\mu + 3\sigma$) that balance sensitivity with false positives, and show that the framework is model-agnostic, enabling integration with different machine learning algorithms beyond XGBoost.

The paper is organized as follows: Section II reviews relevant literature; Section III presents the methodology and ADMXGB framework; Section IV describes the dataset, experiments, and evaluation; and Section V concludes with future research directions.

II. RELATED WORK

Microservices architecture has become foundational in modern cloud-native systems due to its scalability and modularity. However, its distributed and asynchronous nature introduces significant challenges for anomaly detection, particularly when faults propagate across service dependencies. Traditional approaches based on rule-based heuristics and static thresholds are no longer sufficient, prompting the exploration of machine learning (ML) and deep learning (DL) techniques.

Recent research has increasingly adopted automated methods for anomaly detection, including graph neural networks [7], federated learning [8], and sequence-aware architectures [9]. However, two technical challenges remain under-addressed: (i) the severe class imbalance in anomaly data, and (ii) the dependence on optimal hyperparameter settings for complex models.

Data balancing is critical to improving model sensitivity in rare-event classification tasks. Techniques such as SMOTE [10], hybridsampling [11], and Denoise Selection Sampling [12] have been proposed to correct skewed distributions. These techniques aim to prevent overfitting to the majority class and increase recall for minority (anomalous) instances.

Hyperparameter optimisation also plays a central role in improving classification performance and model generalisation. Automated approaches like Bayesian optimisation [13], GridSearchCV [14], and end-to-end tuning frameworks [15] have outperformed manual tuning, particularly in high-dimensional feature spaces.

While several studies address either data balancing or hyperparameter tuning in isolation, relatively few works combine both strategies in the context of microservice anomaly detection. Notable exceptions include [9], [11], [16], which report improved F1-scores through joint optimisation. However, most of these efforts are either focused on IoT networks or lack reproducible datasets.

The TraceRCA [39] dataset has also become foundational for benchmarking root cause analysis (RCA) in microservices. Recent works have leveraged this dataset to evaluate graph-based and interpretable learning approaches. For instance, Sleuth [24] employs graph neural networks (GNNs) to generalize RCA across systems, while GRACE [26] uses a GCN-based architecture for interpretable predictions. Benchmark frameworks like RCAEval [25] position TraceRCA among key datasets for evaluating RCA systems. In addition, approaches such as TraceStream [40] and TICAD [23] improve RCA using feedback-driven clustering and contextual invocation patterns. Zhang et al. [36] combine multi-dimensional trace inputs with performance localization, and anomaly detection techniques using PyOD [21] further showcase the utility of the dataset in unsupervised learning.

While recent research has advanced anomaly detection in microservices, most contributions focus on either class balancing or hyperparameter tuning in isolation. Only a

TABLE 1. Summary of Related Work in Microservice Anomaly Detection using Data Balancing or Hyperparameter Optimisation

Work	Year	Dataset	DB [a]	HT [b]	P [c]	R [d]	A [e]	F1 [f]	Brief Description
Vigoya et al. [17]	2021	IoT-23 [18]	✓	✓	85	88	87	86	Evaluates multiple machine learning algorithms (MLP, SVM, Random Forest) with data balancing and hyperparameter tuning.
Benalddi et al. [19]	2022	UNSW-NB15 [20]	✓	✓	–	–	–	–	Addresses class imbalance through data augmentation and employs hyperparameter tuning for model optimisation.
Landim et al. [21]	2022	TraceRCA	–	–	–	93	–	89	Uses PyOD for unsupervised anomaly detection on real-world microservice traces.
Bugshan et al. [22]	2023	Not publicly available	–	✓	88	90	89	89	Automates hyperparameter tuning and ensemble voting.
Du et al. [23]	2023	TraceRCA	–	–	–	–	–	–	Detects contextual anomalies using trace invocation patterns.
Gan et al. [24]	2023	TraceRCA	–	–	–	–	–	–	Introduces GNN-based Sleuth model, generalizable without retraining across microservices.
Nobre [14]	2023	Not publicly available	✓	✓	89	92	91	91	Employs data balancing techniques and hyperparameter tuning via GridSearchCV.
Pham et al. [25]	2023	TraceRCA	–	–	–	–	–	–	Benchmarks RCA methods and integrates TraceRCA into RCAEval framework.
Ren et al. [26]	2023	TraceRCA	–	–	96	95	–	95	Uses interpretable GCN model to locate microservice root causes in trace data.
Vigoya et al. [16]	2023	CIDAD [16]	✓	✓	85	88	87	86	Introduces the CIDAD dataset for CoAP-based IoT anomaly detection with data balancing and hyperparameter tuning.
Xiong et al. [12]	2023	BGL and HDFS logs [27], [28]	✓	–	–	–	–	–	Utilises Denoise Selection Sampling (DSS) to oversample minority classes.
Zhou et al. [29]	2023	Not publicly available	–	✓	–	–	–	–	Combines optimization with real-time detection layers.
Ge et al. [9]	2024	Not publicly available	✓	✓	91	92	92	91	SRdetector model predicts sequences and flags deviations.
Haq [10]	2024	NSL-KDD [30]	✓	✓	–	–	–	–	Applies SMOTE to handle imbalance before training classifiers.
Li et al. [11]	2024	ToN_IoT [31]	✓	✓	89	91	90	90	Integrates undersampling and oversampling with parallel CNNs.
Mante et al. [32]	2024	IoT-Botnet [33]	–	✓	–	–	99	–	Hyperparameter tuning and feature selection techniques are employed to enhance performance.
Steidl et al. [34]	2024	DeathStarBench [35]	–	✓	–	–	–	88	Optimizes alert thresholds for runtime anomalies.
Zhang et al. [36]	2024	TraceRCA	–	–	–	–	–	–	Uses multi-dimensional trace analysis with GNNs for root cause localization.
Fusco et al. [37]	2025	Not publicly available	✓	–	–	–	–	–	Addresses class imbalance using limited and imbalanced datasets.
Mathews et al. [15]	2025	CICIDS2017 [38]	–	✓	–	–	91	90	Automates entire ML lifecycle from tuning to deployment.
Raeiszadeh et al. [8]	2025	Not publicly available	✓	✓	89	93	91	91	Trains async GNN models across distributed clients.
Wang et al. [7]	2025	Not publicly available	✓	✓	–	–	–	92	Combines vectorized event paths with hybrid GNN detection.

[a]DB – Data Balancing, [b]HT – Hyperparameter Tuning, [c]P - Precision (%), [d]R - Recall (%), [e]A - Accuracy (%), [f]F1 - F1-Score (%).

limited set of works attempt to integrate both, and even fewer evaluate the computational trade-offs of these strategies. These gaps motivate ADMXGB, which uniquely combines latency thresholding, balancing, and systematic tuning in a unified framework validated on TraceRCA.

Table 1 summarises recent contributions that incorporate either or both of these strategies, highlighting their datasets, performance metrics, and methodological focus.

III. PROPOSED METHODOLOGY

ADMXGB is an anomaly detection framework based on the XGBoost algorithm [41], designed to improve classification performance and resource efficiency in microservice environments. It combines three main components: (i) latency-based data filtering, (ii) class balancing, and (iii) hyperparameter optimization, all integrated in a structured four-phase pipeline shown in Fig. 1.

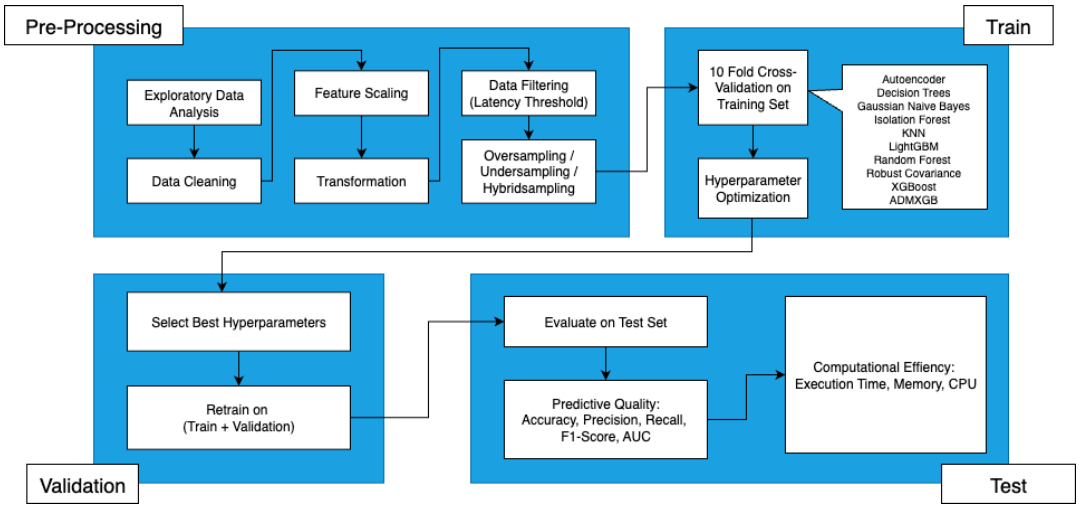


FIGURE 1. Overview of the ADMXGB four-phase pipeline. Each phase highlights the main steps involved in anomaly detection, from data preprocessing and training with hyperparameter tuning to model validation and final evaluation on test data

A. Framework Overview

The ADMXGB pipeline (Algorithm 1) comprises four phases: preprocessing, training and tuning, validation and test.

In the preprocessing phase, the dataset is filtered using a latency threshold τ, removing low-latency traces to reduce noise and overhead. Standard exploratory data analysis, feature scaling, and categorical encoding are applied. To clarify the filtering step, consider a dataset with latency values [2, 3, 8, 15, 100]. Applying a threshold $\tau = 10$ removes all traces with latency below 10 ms, resulting in the subset composed of [15, 100]. These filtered traces are then processed for balancing and later partitioned into training, validation, and test subsets. This simple example demonstrates how τ acts as a noise filter by discarding low-latency values that are unlikely to represent anomalous behavior.

An anomaly is defined as a significant latency deviation relative to the system baseline behaviour. During preprocessing, anomalies are identified based on latency spikes that align with known fault injection events in the TraceRCA dataset. Latency serves as the primary metric for labelling, providing the ground truth used for supervised training and evaluation. This approach ensures consistency in anomaly identification and reflects real-world scenarios in which increased latency typically indicates service degradation or failure. To mitigate class imbalance, three resampling strategies are explored: oversampling (e.g., SMOTE [42]), undersampling, and hybridsampling, which combines both. Data balancing must be applied carefully, particularly in the context of highly imbalanced data. Oversampling may introduce noise or lead to overfitting, while undersampling can discard important information [43], [44]. Therefore, the choice of balancing strategy must be aligned with the dataset characteristics and the tolerance to noise of the model.

During training, models are optimized using 10-fold cross-validation on the training set. Hyperparameter tuning is performed using one of three strategies: Grid Search (which exhaustively evaluates all possible combinations within a predefined grid), Random Search (which randomly samples hyperparameter values), or Bayesian Optimization (HyperOpt [45], which uses a probabilistic model to guide the search). The best hyperparameter configuration is selected based on the highest average F1-Score.

Algorithm 1 ADMXGB – Anomaly Detection in Microservices using XGBoost

Require: Dataset D, latency threshold τ, hyperparameter space H
Ensure: Optimized model M^*

Phase 1: Data Preprocessing
Filter traces in D with latency $< \tau$ ▷ Remove all traces whose observed latency is strictly lower than τ
Perform data cleaning and exploratory analysis
Apply feature scaling and encode categorical variables
Apply data balancing (SMOTE, undersampling, or hybridsampling)

Phase 2: Model Training
for each hyperparameter setting $h \in H$ **do**
    Split D into training set D_{train} and validation set D_{val}
    Train model M_h on D_{train}
    Evaluate M_h on D_{val} using F1-Score
end for
Select optimal hyperparameters h^* with highest F1-Score

Phase 3: Model Validation
Retrain model M^* using $D_{\text{train}} \cup D_{\text{val}}$ and h^*

Phase 4: Model Testing
Evaluate M^* on D_{test} using Accuracy, Precision, Recall, F1-Score, and AUC
return M^*

In the validation phase, the best configuration is used to retrain the model on the full training and validation set.

The final phase evaluates the performance of each model on a hold-out test set using the classification and computational metrics described in the Section B.

XGBoost was selected as the core algorithm of ADMXGB because it is an ensemble-based method that combines multiple weak learners into a robust predictive model. This ensemble strategy allows XGBoost to achieve high accuracy while maintaining efficiency through optimized memory management and parallel execution. Unlike traditional tree-based models, XGBoost integrates regularization to reduce overfitting and supports handling of sparse input, making it suitable for large-scale telemetry data from microservice environments. Its balance of scalability, interpretability, and computational efficiency makes it a more complete solution compared to alternatives such as Random Forest, LightGBM, or deep Autoencoder models.

B. Evaluation Protocol

To ensure fair comparison across models, we apply a uniform evaluation protocol based on 10-fold cross-validation and test set isolation. Unlike standard K-fold workflows, our method separates the test partition from all training and tuning processes. Algorithm 2 outlines a four-stage procedure for model selection and evaluation, combining 10-fold cross-validation with an explicit train/validation/test split.

Algorithm 2 Model Selection and Evaluation using 10-Fold Cross-Validation

Require: Dataset D, hyperparameter space $\mathcal{H}$, number of folds $K = 10$

Ensure: Final model M^* and test performance P_{test}

Step 1: Data Splitting

Split D into:
D_{train} (70%), D_{val} (20%), D_{test} (10%)

Step 2: Hyperparameter Tuning on D_{train}

for each $h \in \mathcal{H}$ **do**
  Initialize performance list $L \leftarrow [\,]$
  Partition D_{train} into $K = 10$ folds: $F_1, F_2, \ldots, F_{10}$.
  for $i = 1$ to 10 **do**
    $T \leftarrow D_{\text{train}} \setminus F_i$, $V \leftarrow F_i$
    Train model $M_{h,i}$ on T
    Evaluate $M_{h,i}$ on V to obtain $p_{h,i}$
    Append $p_{h,i}$ to L
  end for
  Compute mean performance $\bar{p}_h \leftarrow \text{mean}(L)$
end for
Select $h^* = \arg\max_h \bar{p}_h$

Step 3: Final Model Training

Retrain M^* on $D_{\text{train}} \cup D_{\text{val}}$ using h^*

Step 4: Final Evaluation

Evaluate M^* on D_{test} to obtain P_{test}

return M^* and P_{test}

First, the dataset D is partitioned into training (D_{train}), validation (D_{val}), and test (D_{test}) sets. The test set is held out entirely for final evaluation. Note that D_{test} is initialized during the first data split and is never subject to threshold filtering or balancing, ensuring unbiased evaluation of the final model. Algorithm 2 does not operate directly on the raw dataset D. Instead, it receives as input the preprocessed dataset D', which results from Algorithm 1 (ADMXGB) after applying latency thresholding, data cleaning, feature scaling, and balancing. Next, hyperparameter tuning is performed on D_{train} using 10-fold cross-validation. Each configuration $h \in \mathcal{H}$ is evaluated across folds, and the average performance $\bar{p}_h$ is computed. The best configuration h^* is selected. Then, the final model M^* is trained using $D_{\text{train}} \cup D_{\text{val}}$ and h^*. Finally, M^* is evaluated on D_{test}, yielding the test performance P_{test}. This approach ensures reliable model selection while preventing data leakage.

This pipeline is applied to ADMXGB and all baseline models. These include classical algorithms (e.g., Decision Trees, Naive Bayes, KNN), as well as advanced ensemble learners (e.g., Random Forest, LightGBM, XGBoost) and deep models (Autoencoders). The objective function in hyperparameter tuning maximizes the average F1-score to handle the skewed class distribution. Default hyperparameter configurations and the ranges used in tuning are detailed in Tables 2 and 3.

The performance of the different models was evaluated using a range of metrics, including Accuracy, Precision, Recall, F1-Score, and AUC [54], as well as execution time, RAM usage, and CPU utilization. The results obtained under various data balancing and hyperparameter optimization conditions are comprehensively summarized in Tables 4–6, enabling a direct comparison across methods, evaluation metrics, and computational resource consumption. This comprehensive evaluation tests the generalization capability of each model, reducing the risk of overfitting while enhancing predictive accuracy and reliability. This layered design, which combines data filtering, balancing, nested validation, and final isolation of test sets, supports reproducibility and highlights the novel structure of the ADMXGB training pipeline.

IV. RESULTS AND DISCUSSION

A. Dataset

The dataset made available by the authors of TraceRCA [39], a practical root-cause localisation method introduced in 2021, was used. Since TraceRCA does not provide explicit anomaly labels, we annotated anomalies by aligning fault injection logs with trace timestamps. Latency spikes were identified using a statistical criterion: traces with latency exceeding the mean μ plus three standard deviations ($\mu+3\sigma$) of the baseline distribution were labeled as anomalous. This 3σ rule has been widely used in anomaly detection to distinguish abnormal values from natural fluctuations [56], [57].

TABLE 2. Default hyperparameters for Anomaly Detection Methods

Method	Default hyperparameters
Autoencoder [46]	activation='relu', optimizer='adam', loss='mean_squared_error', epochs=100, batch_size=32
Decision Tree [47]	criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, max_features=None
Gaussian Naive Bayes [48]	priors=None, var_smoothing=1e-9
Isolation Forest [49]	n_estimators=100, max_samples='auto', contamination='auto', max_features=1.0
K-Nearest Neighbors [50]	n_neighbors=5, algorithm='auto', metric='minkowski', p=2
LightGBM [51]	num_leaves=31, learning_rate=0.1, n_estimators=100, min_child_samples=20, subsample=1.0
Random Forest [52]	n_estimators=100, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, max_features='auto'
Robust Covariance [53]	support_fraction=None, covariance_threshold=None
XGBoost [41]	n_estimators=100, learning_rate=0.1, max_depth=6, colsample_bytree=1.0, subsample=1.0, gamma=0, reg_alpha=0, reg_lambda=1
ADMXGB	n_estimators=100, learning_rate=0.1, max_depth=6, colsample_bytree=1.0, subsample=1.0, gamma=0, reg_alpha=0, reg_lambda=1, threshold=10

TABLE 3. Values used for Random and Grid Search methods

Method	Values
Autoencoder [46]	encoding_dim: [8, 16, 32, 64], epochs: [30, 50, 70], batch_size: [16, 32, 64]
Decision Tree [47]	criterion: ['gini', 'entropy'], max_depth: [3, 5, 10, 15], min_samples_split: [2, 5, 10]
Gaussian Naive Bayes [48]	var_smoothing: [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]
Isolation Forest [49]	n_estimators: [50, 300, 10], max_samples: [0.1, 1.0], contamination: [0.01, 0.5], max_features: [0.1, 1.0]
K-Nearest Neighbors [50]	n_neighbors: [3, 5, 7, 9], weights: ['uniform', 'distance'], p: [1, 2]
LightGBM [51]	n_estimators: range(50, 301, 50), learning_rate: [0.01, 0.05, 0.1, 0.2, 0.3], num_leaves: [20, 50, 100, 150], max_depth: [3, 5, 10, 15]
Random Forest [52]	n_estimators: [50, 300, 10], max_depth: [3, 15, 1], max_features: [0.1, 1.0], min_samples_split: [2, 10, 1]
Robust Covariance [53]	contamination: [0.01, 0.5], support_fraction: [0.1, 1.0]
XGBoost [41]	n_estimators: [50, 100, 150, 200, 250, 300], max_depth: [3, 5, 7, 10, 15], learning_rate: [0.01, 0.05, 0.1, 0.2, 0.3], subsample: [0.6, 0.7, 0.8, 0.9, 1.0], colsample_bytree: [0.6, 0.7, 0.8, 0.9, 1.0]
ADMXGB	n_estimators: [50, 100, 150, 200, 250, 300], max_depth: [3, 5, 7, 10, 12, 15], learning_rate: [0.01, 0.05, 0.1, 0.2, 0.3], subsample: [0.6, 0.7, 0.8, 0.9, 1.0], colsample_bytree: [0.6, 0.7, 0.8, 0.9, 1.0]

Most existing studies based on the TraceRCA dataset primarily utilize metrics tailored for RCA, so to provide a fair and transparent comparison, we benchmark ADMXGB against nine widely adopted anomaly-detection baselines that are dataset-agnostic and have public implementations.

While we acknowledge the importance of multi-dataset validation, the TraceRCA dataset was selected due to its real-world characteristics, comprehensive labelling, and heterogeneous trace patterns. Additionally, its substantial size (5.5 GB) and volume (over 85 million entries) offer a realistic and challenging benchmark for anomaly detection methods. Its widespread adoption in recent anomaly detection studies further reinforces its suitability as a benchmark for evaluating microservices-based methods.

To better understand the characteristics of the dataset, several preliminary checks were conducted. No missing values were identified, thus eliminating the need for imputation. However, an anomaly was detected: 37 records contained negative latency values, which required careful handling to preserve data integrity. Feature selection was applied, retaining the following attributes: *latency*, *source*, *target*, and *succ*. To ensure compatibility with algorithms that require numerical inputs (e.g., XGBoost, LightGBM, and ADMXGB), the categorical attributes *source*, *target*, and *succ* were mapped as integer values. Additionally, the class distribution was analysed to assess potential imbalance (see Table 4).

The latency distribution in TraceRCA exhibits a pronounced right skew typical of microservice workloads. The mean latency is $9.758\,\text{ms}$ with a standard deviation of $161.587\,\text{ms}$, the median is $1\,\text{ms}$, and the 95th percentile is $34\,\text{ms}$. Consequently, more than 95% of traces fall below $34\,\text{ms}$, which we consider the normal operating range. Adopting a statistical criterion, we set the anomaly threshold at $\mu + 3\sigma \approx 494.520\,\text{ms}$. Although this criterion does not assume normality, it effectively captures rare latency spikes. In the full dataset ($N = 85{,}561{,}389$), this rule flags approximately 0.05% of traces as anomalous, underscoring the rarity of extreme delays.

This evaluation helped determine whether techniques such as resampling or class weighting would be necessary to mitigate skewed class distributions.

TABLE 4. Performance Comparison (Mean Train Accuracy, Test Accuracy, and F1-Score) of Imbalanced vs. Balanced Sampling Methods. The base performance for each model is reported on imbalanced data (IB), with the values in square brackets indicating the difference when applying balanced-oversampling (BDO), balanced-undersampling (BDU), and balanced-hybridsampling (BDH) (full version available in [55]).

Method	Mean Train Accuracy IB [BDO, BDU, BDH]	Test Accuracy IB [BDO, BDU, BDH]	F1-Score IB [BDO, BDU, BDH]
	Without hyper-parameter Tuning (WHT)		
Autoencoder	94.12 [+0.68, +1.28, +1.73]	95.15 [+1.05, +0.27, +0.63]	1.15 [+0.30, -0.75, -0.69]
Decision Trees	99.10 [-6.60, -7.10, -7.70]	99.95 [-9.66, -10.09, -9.35]	6.56 [-5.57, -5.57, -5.55]
Gaussian Naive Bayes	98.45 [-8.25, -8.95, -7.95]	99.82 [-10.02, -11.07, -9.97]	26.30 [-23.72, -25.41, -25.31]
Isolation Forest	95.60 [-4.10, -5.40, -3.60]	94.55 [-3.65, -4.85, -2.75]	13.48 [-9.40, -12.48, -12.49]
KNN	99.88 [-0.88, -8.88, -0.68]	99.95 [-0.19, -9.21, -0.32]	99.94 [-87.50, -98.95, -90.93]
LightGBM	99.20 [-1.40, -8.15, -1.10]	99.94 [-4.59, -8.94, -2.04]	15.86 [-14.31, -14.78, -14.37]
Random Forest	98.30 [-0.30, -1.20, -0.30]	99.10 [-1.60, -2.75, -1.70]	31.10 [+7.32, -1.72, +3.97]
Robust Covariance	90.52 [-0.42, -0.12, +1.08]	90.10 [-0.90, -0.30, +0.60]	6.44 [-5.65, -5.57, -5.41]
XGBoost	98.50 [-3.10, -7.16, -3.50]	99.91 [-4.91, -9.14, -5.31]	52.10 [-50.32, -51.05, -51.09]
ADMXGB	**99.96** [-0.16, -0.30, -0.35]	99.67 [**+0.13**, -0.05, +0.06]	31.02 [**+11.38**, -1.73, +6.51]

TABLE 5. Performance Comparison (Precision, Recall and AUC) of Imbalanced vs. Balanced Sampling Methods. The base performance for each model is reported on imbalanced data (IB), with the values in square brackets indicating the difference when applying balanced-oversampling (BDO), balanced-undersampling (BDU), and balanced-hybridsampling (BDH) (full version available in [55]).

Method	Precision IB [BDO, BDU, BDH]	Recall IB [BDO, BDU, BDH]	AUC IB [BDO, BDU, BDH]
	Without hyper-parameter Tuning (WHT)		
Autoencoder	0.58 [+0.17, -0.38, -0.35]	54.29 [-32.33, -36.41, -36.12]	74.73 [-15.63, -18.06, -17.73]
Decision Trees	31.64 [-31.14, -31.14, -31.13]	3.66 [+91.48, +95.72, +90.48]	51.83 [+40.88, +42.79, +40.54]
Gaussian Naive Bayes	25.53 [-24.18, -25.08, -25.03]	27.12 [+3.30, +14.88, +8.38]	64.10 [-1.00, +0.10, -0.20]
Isolation Forest	12.22 [-10.02, -11.71, -11.72]	15.04 [+13.06, +25.06, +32.14]	58.55 [+6.45, +3.45, +0.75]
KNN	99.90 [-92.19, -99.40, -94.72]	99.98 [-67.78, -3.27, -65.47]	53.30 [+12.70, +39.28, +13.79]
LightGBM	31.02 [-30.24, -30.48, -30.27]	10.65 [+61.08, +87.93, +59.35]	55.32 [+28.16, +39.47, +33.38]
Random Forest	40.00 [-6.48, -7.78, -1.20]	25.44 [+19.56, +1.56, +6.56]	80.20 [+5.30, -0.10, +3.90]
Robust Covariance	4.10 [-3.70, -3.66, -3.58]	15.00 [+10.10, +21.11, +31.11]	59.12 [+1.08, +1.03, +1.21]
XGBoost	75.01 [-74.11, -74.48, -74.50]	39.91 [+42.49, +59.15, +57.59]	88.20 [-0.20, +6.71, +5.80]
ADMXGB	18.36 [**+9.73**, -1.20, +5.05]	**99.96** [-13.55, -0.02, -5.39]	99.65 [-6.54, **+0.11**, -2.50]

TABLE 6. Performance Comparison (Execution Time, Memory Usage and CPU Usage) of Imbalanced vs. Balanced Sampling Methods. The base performance for each model is reported on imbalanced data (IB), with the values in square brackets indicating the difference when applying balanced-oversampling (BDO), balanced-undersampling (BDU), and balanced-hybridsampling (BDH) (full version available in [55]).

Method	Execution Time (s) IB [BDO, BDU, BDH]	Memory Usage (GB) IB [BDO, BDU, BDH]	CPU Usage (%) IB [BDO, BDU, BDH]
	Without hyper-parameter Tuning (WHT)		
Autoencoder	32866 [-2959, -3271, -2316]	15.62 [+0.47, -7.36, -0.19]	20.07 [+3.09, **-14.01**, +15.53]
Decision Trees	3105 [-996, **-2594**, +821]	7.97 [-3.48, -3.54, -3.50]	14.38 [+1.12, -7.95, +8.63]
Gaussian Naive Bayes	2725 [-1414, -1953, -624]	1.15 [**-0.05**, 0.00, **-0.05**]	9.44 [-2.04, -2.34, -1.44]
Isolation Forest	1982 [+118, -1102, +2040]	2.60 [-0.35, -1.10, -0.56]	7.22 [+1.78, -0.42, -0.21]
KNN	4550 [-1095, -3330, -2650]	9.40 [-2.60, -4.25, -2.80]	11.10 [+0.10, -2.70, -2.00]
LightGBM	6123 [-1633, -4473, -2973]	3.50 [-0.30, +0.30, +0.30]	12.50 [+0.90, -2.30, +1.70]
Random Forest	7100 [-980, -2661, -1980]	4.31 [-0.06, -0.11, -0.01]	18.80 [-1.80, -5.70, -2.10]
Robust Covariance	1432 [+548, -292, +8]	1.94 [-0.22, -0.29, +0.05]	6.55 [+0.70, -0.44, +0.95]
XGBoost	5450 [-2262, -3063, -2120]	2.90 [-0.25, +1.32, -0.40]	23.20 [-3.30, -3.49, -2.50]
ADMXGB	755 [-223, **-632**, +3016]	3.34 [-0.54, **-1.03**, -0.30]	60.20 [+18.30, **-41.90**, +0.60]

TABLE 7. Class Imbalance Calculation

Class type	Count	Ratio (%)
Anomaly	41,532	0.048517
Normal	85,561,389	99.951483

B. Testbed and Reproducibility

All experiments were implemented in Python (v3.12.3), using the following key libraries: Scikit-learn (v1.5.2) for general machine learning tasks, Pandas (v2.2.3) for data manipulation, NumPy (v2.0.2) for numerical operations, XGBoost (v2.1.1) and LightGBM (v4.1.0) for gradient boosting models, and Imbalanced-learn (v0.12.4) for sampling techniques (SMOTE, RandomUnderSampler, SMOTEENN). Hyperparameter optimisation was performed using GridSearchCV, RandomizedSearchCV, and Hyperopt (v0.2.7) with TPE as the search algorithm.

The experiments were executed on a local testbed with an Intel® Core™ i7-8700 CPU @ 3.20GHz (12 threads), 64 GB RAM, and Ubuntu 22.04.4 LTS 64-bit. All code was run using a controlled virtual environment (venv) to ensure consistency across runs.

The complete source code, configuration files, trained models, and preprocessing scripts are publicly available in our GitHub repository [58]. The dataset used (TraceRCA) was preprocessed using timestamps provided in the fault injection logs to label anomalous traces. Our processed version of the dataset is also available via Zenodo [59], ensuring that all results can be reproduced and verified.

C. Impact of Latency Threshold

As previously described in Section III, the latency threshold τ was varied to assess its effect on performance and efficiency. The ADMXGB model was evaluated across a range of values for τ: 0 (no filtering), 5, 10, 20, 50, 100, 200, 500, and 1000 ms. Table 8 presents the results in terms of F1-Score, accuracy, execution time, and best hyperparameters (for Grid Search with data balanced via undersampling).

A particularly interesting observation is related to execution time. Contrary to expectations, lower thresholds, which correspond to significantly larger datasets, do not always result in longer training times. For example, with $\tau = 1000$, the training set contains only 68,039 samples, yet the execution time reaches 3,197 seconds. In contrast, at $\tau = 10$, the model processes more than 8.9 million records but completes training in just 1,154 seconds.

This counterintuitive result can be attributed to the increased complexity of the models trained on smaller datasets. At higher thresholds (e.g., $\tau = 1000$ and $\tau = 500$), the selected hyperparameters include deeper trees (e.g., `max_depth=7`) and higher learning rates. These configurations, when applied to limited and less diverse data, can lead to overfitting and longer convergence times, as the model struggles to learn generalized patterns.

In contrast, with lower thresholds and more comprehensive datasets, the model converges faster using shallower trees and smaller learning rates. The larger dataset size provides richer, more representative patterns, allowing the learning process to be both more stable and efficient.

These findings demonstrate that applying a latency threshold is not merely a filtering mechanism but also a determinant of learning efficiency. Models trained on heavily filtered data (high τ) may experience inflated training times, even with smaller datasets, due to hyperparameter-driven complexity and reduced data variance.

D. Impact of Balancing the Dataset

Dataset balancing improves model performance, especially with class-imbalanced data. Results show that models react differently to balancing strategies. *Oversampling* boosts minority class representation, with LightGBM, KNN, and ADMXGB showing notable gains in Recall and F1-score. However, it increases computational cost, particularly for resource-heavy models like Autoencoders.

In contrast, *undersampling* reduces the dataset size by removing majority class samples, easing resource use but often lowering F1-score and AUC for models like Decision Trees and Random Forests. ADMXGB stays relatively stable, while KNN and LightGBM perform variably.

Hybridsampling, combining both methods, proves most effective. It enhances performance while mitigating individual drawbacks. ADMXGB and LightGBM reach near-optimal AUC and F1-scores, with XGBoost and Random Forests also benefiting, though at higher resource cost. Overall, it improves model robustness and generalisation.

Although dataset balancing is a common strategy to mitigate the bias caused by the extreme class imbalance [60], [61], the results indicate that for certain algorithms, such as Decision Tree, KNN, Gaussian Naive Bayes, and Isolation Forest, the F1-score values were higher when using the original, imbalanced dataset. While this might seem contradictory, it can be explained by the fact that some methods tend to overfit to synthetic patterns introduced by balancing techniques, particularly oversampling [62]–[64].

E. Impact of Hyperparameters Optimization

Hyperparameter tuning using *Grid Search*, *Random Search*, and *HyperOPT* raises Precision, Recall, F1-Score, and AUC toward theoretical limits, complementing data balancing. *Grid Search* exhaustively tests every parameter set and yields top performance, but its exponential runtime is prohibitive on large data. *Random Search* and *HyperOPT* explore only part of the space, sacrificing a little Accuracy for much lower computation; HyperOPT's Bayesian focus often rivals Grid on complex tasks.

Experiments show tuning particularly aids imbalanced data: ADMXGB excels with Grid and HyperOPT; LightGBM and XGBoost improve under all three. Autoencoders and Isolation Forests gain most from Grid yet incur heavy

TABLE 8. Impact of Latency Threshold (τ) on ADMXGB Performance

Threshold	Dataset Size	Size After Excl.	Train Acc (%)	Test Acc (%)	Mean F1 (%)	Exec. Time (s)
	Best Hyperparameters					
0	84,104,058	84,104,021	99.84	99.84	99.84	1154.83
	colsample_bytree: 0.6, learning_rate: 0.2, max_depth: 3, n_estimators: 200, subsample: 0.7					
5	84,104,058	15,024,905	99.89	99.88	99.88	1588.61
	colsample_bytree: 0.6, learning_rate: 0.1, max_depth: 5, n_estimators: 300, subsample: 0.7					
10	84,104,058	8,944,558	99.98	99.98	99.98	1664.96
	colsample_bytree: 0.6, learning_rate: 0.01, max_depth: 3, n_estimators: 50, subsample: 0.6					
20	84,104,058	6,746,550	99.74	99.70	99.71	1891.87
	colsample_bytree: 0.6, learning_rate: 0.3, max_depth: 3, n_estimators: 250, subsample: 0.6					
50	84,104,058	3,597,243	99.63	99.62	99.62	1959.85
	colsample_bytree: 0.6, learning_rate: 0.01, max_depth: 7, n_estimators: 50, subsample: 1.0					
100	84,104,058	1,020,751	98.65	98.64	98.66	2949.95
	colsample_bytree: 0.8, learning_rate: 0.1, max_depth: 3, n_estimators: 150, subsample: 0.9					
200	84,104,058	431,762	96.80	96.70	96.78	2653.91
	colsample_bytree: 1.0, learning_rate: 0.05, max_depth: 5, n_estimators: 150, subsample: 1.0					
500	84,104,058	161,243	93.44	92.45	92.71	3402.82
	colsample_bytree: 0.8, learning_rate: 0.05, max_depth: 7, n_estimators: 200, subsample: 0.6					
1000	84,104,058	68,039	86.45	86.12	86.78	3197.49
	colsample_bytree: 0.8, learning_rate: 0.05, max_depth: 5, n_estimators: 150, subsample: 0.6					

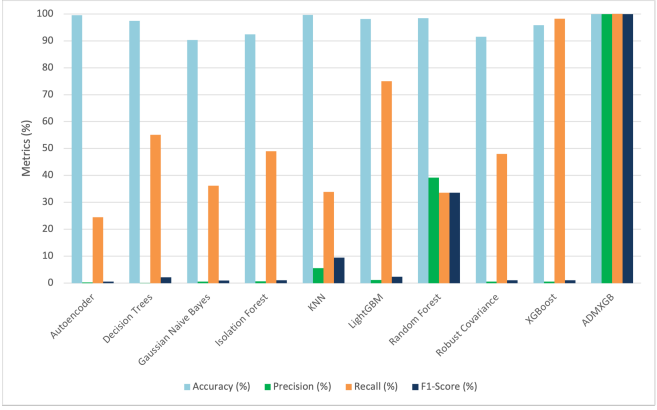


FIGURE 2. Comparative performance metrics (Accuracy, Precision, Recall, F1-Score) of all evaluated methods trained on hybrid-balanced data with hyperparameter tuning via Grid Search. In this scenario, ADMXGB demonstrates the most significant performance gains over the baseline methods.

costs, while Decision Trees and Gaussian Naive Bayes change little, reflecting weak tuning dependence.

Selecting a tuning method therefore means balancing accuracy, runtime, and resources, which is crucial for real-time anomaly detection and other large-scale applications.

F. Validation

The experimental results show that the proposed ADMXGB framework outperforms the Accuracy figures commonly reported in the literature (see Table 1). Whereas most recent studies on microservice anomaly detection remain in the range of 85%–99% Accuracy and Recall, ADMXGB reaches: up to **99.96%** test Accuracy, **99.95%** Precision, **99.99%** Recall, a **99.97%** F1-Score and an AUC of **99.96%**. These values are obtained under the following configurations:

- *Grid Search + hybridsampling*: Accuracy=99.96%, F1-Score=99.97%, AUC=99.96%;
- *Random Search + hybridsampling*: Precision=99.95% (+81.6% over the imbalanced baseline);
- *Random Search + oversampling (SMOTE)*: Recall=99.99% (+93.2% over the imbalanced baseline);
- *No tuning + imbalanced dataset*: Train-Accuracy=99.96% (upper bound on fitting capacity).

ADMXGB consistently achieves high Accuracy, Precision, and Recall (see Figure 2), maintaining near-real-time execution even under higher CPU and memory usage, making it well-suited for rapid anomaly detection. In comparison, XGBoost and LightGBM offer competitive Accuracy but require extensive tuning, increasing computational costs in large-scale settings. Autoencoders and Isolation Forests, while effective in specific cases, are memory-intensive and less suitable for resource-constrained scenarios. Balancing methods, including oversampling, undersampling, and hybrid strategies, further refine model outcomes. Hybrid approaches, in particular, effectively merge the strengths of both techniques, benefiting ADMXGB and LightGBM. Hyperparameter tuning enhances model performance but varies in computational cost: Grid Search is thorough yet resource-heavy, Random Search is faster but less comprehensive, and HyperOpt balances accuracy and efficiency more effectively.

Hyperparameter optimisation and sampling strategies act synergistically on ADMXGB performance:

1) **Precision boost.** On the raw, highly imbalanced TraceRCA data the model achieves only 18.36% Precision;

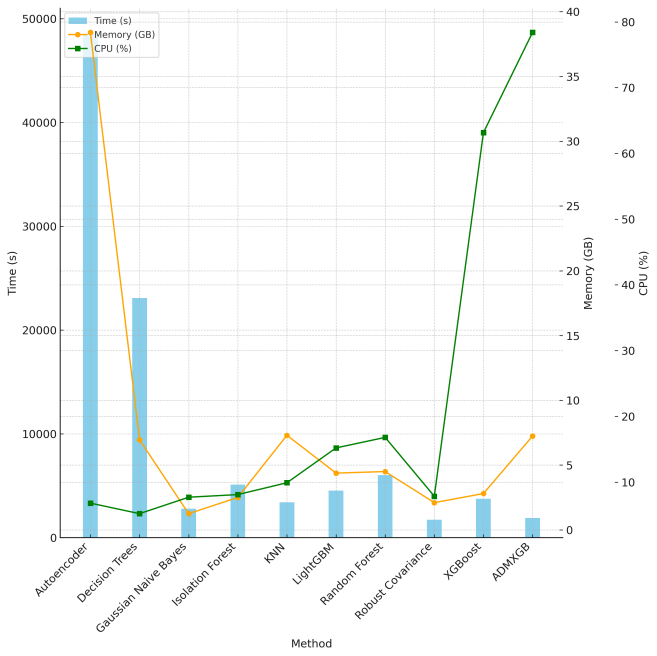


FIGURE 3. Resource usage of methods using data balanced with hybridsampling and hyperparameter tuning via Hyperopt

Random Search + hybrid sampling lifts this to 99.95%, an improvement of **+81.6%** ;

2) **Recall recovery.** While aggressive tuning on the unbalanced set may drive Recall down to 6.82%, applying SMOTE before training restores it to 99.99% (**+93.2%**);
3) **Global balance (F1-Score, AUC).** *Grid Search + hybrid sampling* aligns both classes almost perfectly, pushing F1-Score and AUC to 99.97% and 99.96%, respectively. HyperOPT achieves the same AUC with only one third of the search budget, confirming the efficiency of Bayesian hyperparameter optimisation;
4) **Resource footprint.** Balancing adds negligible memory overhead ($<$ 1 GB) and hyperparameter optimisation adds modest CPU overhead (20%) when Bayesian optimisation is used; exhaustive Grid Search can be over one order of magnitude slower and should therefore be reserved for offline re-training.

Among the tested methods, ADMXGB demonstrates a particularly strong balance between Accuracy, Recall, and execution time, making it a suitable candidate for near real-time anomaly detection in microservices. This performance is primarily attributed to two synergistic mechanisms. First, the introduction of a configurable latency threshold (τ) enables the model to disregard low-latency traces that typically introduce noise and false positives. Second, the systematic integration of data balancing strategies with hyperparameter optimisation, validated through 10-fold cross-validation, enhances model generalization and stability. Furthermore, ADMXGB leverages hybrid sampling to address class imbalance and employs efficient optimisation algorithms such as HyperOpt to maintain resource efficiency. These mechanisms are particularly effective in the context of the TraceRCA dataset, where subtle anomaly patterns challenge traditional models. ADMXGB's gradient boosting structure proves especially robust under undersampling and hybrid sampling conditions, achieving consistently high precision and F1-Score while maintaining practical execution times (see Figure 3).

G. Computational Performance Comparison

Following recommendations from the AI-Centric Computing Continuum Systems framework [65], we extended the evaluation beyond accuracy-related metrics to include system-level indicators such as CPU utilization, memory footprint, and execution time. These additional parameters are reported in Table 6 and Figure 3, providing a comprehensive view of the operational cost of ADMXGB compared to baseline models. The results confirm that ADMXGB maintains competitive efficiency while significantly improving anomaly detection performance.

Execution time, memory usage, and CPU utilization show clear trade-offs between algorithms, sampling strategies, and tuning methods. ADMXGB is fastest with undersampling (BDU) and minimal tuning, reaching **64 s** (Random Search), but oversampling (BDO) or hybridsampling (BDH) with exhaustive tuning can exceed **41,000 s**. Tree-based models generally take 2,000–8,000 s, while Autoencoders are highly variable, sometimes extreme. In memory, Gaussian Naive Bayes and Robust Covariance use under **2 GB**, tree-based models remain moderate (2–7 GB), and Autoencoder/ADMXGB can peak very high (e.g., **63.49 GB** for Autoencoder with BDO and Random Search) due to dataset size and search space. For CPU, ADMXGB often reaches the highest loads (up to **95.40%**) in balanced datasets with exhaustive tuning, whereas Decision Trees and Gaussian Naive Bayes stay below 10%, making them preferable for resource-constrained environments.

H. Implications and Practical Impact

The ADMXGB framework addresses key operational challenges in microservices environments by offering a practical and lightweight anomaly detection mechanism. The use of latency thresholding reduces CPU and memory usage, making it viable for near real-time deployment in edge or containerized infrastructures. Moreover, the use of data balancing and hyperparameter tuning allows the model to adapt to highly imbalanced and dynamic workloads, which are frequent in production microservice systems.

Our method is especially suited for scenarios where telemetry is abundant but computational efficiency is a priority, such as continuous delivery pipelines or autoscaling decision-making processes. While the evaluation was conducted on a single real-world dataset, the methodological design is dataset-agnostic and can be replicated on other telemetry-rich environments.

In practice, the latency threshold τ should be calibrated to the application baseline performance. A recommended guideline is to compute the mean latency μ during stable operation and set τ between $\mu + 2\sigma$ and $\mu + 3\sigma$, depending on the acceptable trade-off between sensitivity and false positives. This ensures that thresholding remains aligned with system-specific performance.

Future extensions will validate ADMXGB on different microservices platforms and explore its integration with AIOps pipelines for automated root-cause identification and anomaly response. In this context, ADMXGB can be embedded into modern monitoring architectures based on observability tools like Prometheus, Jaeger, or OpenTelemetry. The model may act as a dedicated anomaly detection microservice, processing telemetry in real-time or batch mode and emitting alerts to external systems such as Grafana or Slack.

The performance of ADMXGB can be explained by the combined effect of three factors: (i) latency threshold filtering, which removes low-latency noise and reduces false positives, (ii) class balancing, which prevents bias toward the majority class and enhances recall for anomalous cases, and (iii) systematic hyperparameter tuning, which improves generalization by exploring a broader parameter space. Together, these elements create a synergy that allows ADMXGB to outperform baselines not only in accuracy-related measures but also in terms of stability and efficiency, as shown in Tables 4-6.

Furthermore, ADMXGB can be coupled with *GambitRCA* [66], a complementary framework for root-cause analysis based on cooperative game theory. In this combined pipeline, ADMXGB first detects anomalous traces, which are then processed by GambitRCA to identify the most probable components responsible for the incident. This layered approach enhances the reliability and interpretability of anomaly management in complex microservice systems.

I. Threats to Validity

Several factors may affect the validity of our findings. **Internal validity** may be limited by the use of a single dataset (TraceRCA); however, its real-world origin, fault diversity, and volume mitigate this concern. **External validity** could be impacted by architectural differences in other microservice systems. Future work will address this by including other datasets. **Construct validity** depends on the chosen evaluation metrics, which, although standard, may not capture interpretability or robustness. **Conclusion validity** is supported by reproducibility, 10-fold cross-validation, and multiple optimisation strategies, reducing bias and overfitting risks.

V. CONCLUSION

Effective anomaly detection in microservices requires balancing data handling, hyperparameter tuning, and resource efficiency. Our results show that hybridsampling achieves the highest Recall, F1-Score, and AUC across models like ADMXGB, LightGBM, and XGBoost, while keeping high values of Accuracy. Oversampling improves minority-class representation, it increases memory usage, especially for Autoencoders. Undersampling, though faster, risks losing critical data. However, combining it with ADMXGB offers a practical trade-off for real-time applications.

For hyperparameter tuning, Grid Search is thorough but computationally expensive. Random Search provides a cost-effective alternative, while HyperOpt optimally balances accuracy and efficiency, especially for large-scale systems. Our method adapts well to different strategies, maintaining high detection rates even in the presence of class imbalance. Ultimately, practitioners must balance recall, precision, and resource constraints. Undersampling with ADMXGB suits fast detection, while hybridsampling with advanced optimisation (e.g., HyperOpt) maximises performance.

While this work focused on XGBoost, the thresholding and balancing components of ADMXGB are model-agnostic and can be applied to other learners such as LightGBM, Random Forests, or deep neural networks, further broadening the applicability of the framework.

Although the results are promising, this study presents some limitations. The analysis was conducted using an extensive and realistic, yet single real-world dataset (TraceRCA), which may limit generalizability across different microservice platforms and workloads. Moreover, while supervised learning yields high accuracy, it depends heavily on labelled data, which is often scarce or expensive to obtain in operational environments. Future work will aim to extend this approach to multiple datasets, explore unsupervised or semi-supervised labelling techniques, and integrate more detailed telemetry data (e.g., logs, metrics, traces) to support not only detection but also accurate root cause localisation in complex environments.

REFERENCES

[1] Brandón, M. Solé, A. Huélamo, D. Solans, M. S. Pérez, and V. Muntés-Mulero, "Graph-based root cause analysis for service-oriented and microservice architectures," *Journal of Systems and Software*, vol. 159, p. 110432, Jan. 2020.

[2] C. Fetzer, "Building Critical Applications Using Microservices," *IEEE Security Privacy*, vol. 14, pp. 86–89, Nov. 2016.

[3] S. Newman, *Building Microservices*. O'Reilly Media, Inc, 1st edition ed., 2015.

[4] M. Fowler, "Microservices Definition." Available at https://martinfowler.com/articles/microservices.html (2024/04/20).

[5] D. Inc., "Docker: Accelerated, Containerized Application Development." Available at https://www.docker.com/ (2024/04/20).

[6] S. Wolfe, "Amazon's one hour of downtime on Prime Day may have cost it up to $100 million in lost sales." Available at https://www.businessinsider.com/amazon-prime-day-website-issues-cost-it-millions-in-lost-sales-2018-7 (2024/04/21).

[7] P. Wang, X. Zhang, and Z. Cao, "Anomaly detection for microservice system via augmented multimodal data and hybrid graph representations," *Information Fusion*, vol. 118, p. 103017, 2025.

[8] M. Raeiszadeh, A. Ebrahimzadeh, R. H. Glitho, J. Eker, and R. A. F. Mini, "Asynchronous real-time federated learning for anomaly detection in microservice cloud applications," *IEEE Transactions on Ma-*

chine Learning in Communications and Networking, vol. 3, pp. 176–194, 2025.

[9] H. Ge, X. Ji, F. Peng, R. Chen, N. Xiang, K. Zhang, and W. Wu, "Srdetector: Sequence reconstruction method for microservice anomaly detection," *Electronics*, vol. 14, no. 1, 2025.

[10] M. S. Haq, "Data-centric analysis of security and privacy of containerized applications," *The University of Texas at San AntonioProQuest Dissertations Theses*, 2024.

[11] X. Li, P. Wen, P. Chen, J. Chen, X. Wen, and Y. Xia, "An effective parallel convolutional anomaly multi-classification model for fault diagnosis in microservice system," *Software Quality Journal*, 2024.

[12] W. Xiong, W. Chen, J. Liu, and K. Zhao, "An anomaly detection framework for system logs based on ensemble learning," in *PRICAI 2023: Trends in Artificial Intelligence*, vol. 14325 of *Lecture Notes in Artificial Intelligence*, Springer, 2023.

[13] M. Steidl, M. Leitner, P. Urbanke, M. Gattringer, M. Felderer, and S. Ristov, "Understanding microservice runtime monitoring data for anomaly detection with structural equation modeling," in *Proceedings of the Product-Focused Software Process Improvement Conference* (D. Pfahl, J. Gonzalez Huerta, J. Klünder, and H. Anwar, eds.), vol. 15452 of *Lecture Notes in Computer Science*, (Cham), Springer, 2025. PROFES 2024.

[14] J. Nobre, E. J. S. Pires, and A. Reis, "Anomaly detection in microservice-based systems," vol. 13, no. 13, p. 7891. Number: 13 Publisher: Multidisciplinary Digital Publishing Institute.

[15] D. R. Mathews, "Application service resilience in cloud: An end-to-end perspective," *IISC Thesis*, 2025.

[16] L. Vigoya, A. Pardal, D. Fernandez, and V. Carneiro, "Application of machine learning algorithms for the validation of a new CoAP-IoT anomaly detection dataset," vol. 13, no. 7, p. 4482. Number: 7 Publisher: Multidisciplinary Digital Publishing Institute.

[17] L. Vigoya, D. Fernandez, V. Carneiro, and F. J. Nóvoa, "IoT dataset validation using machine learning techniques for traffic anomaly detection," vol. 10, no. 22, p. 2857. Number: 22 Publisher: Multidisciplinary Digital Publishing Institute.

[18] S. Garcia, "Iot-23: A labeled dataset for iot network intrusion detection," 2020. Accessed: 2025-04-29.

[19] H. Benaddi, M. Jouhari, K. Ibrahimi, J. Ben Othman, and E. M. Amhoud, "Anomaly detection in industrial IoT using distributional reinforcement learning and generative adversarial networks," vol. 22, no. 21, p. 8085. Number: 21 Publisher: Multidisciplinary Digital Publishing Institute.

[20] N. Moustafa and J. Slay, "Unsw-nb15 dataset," 2015. Accessed: 2025-04-29.

[21] L. Landim, L. Barata, and E. Lopes, "A simple approach to detect anomalies in microservices-based systems using PyOD," in *Proceedings of the 22.ª Conferência da Associação Portuguesa de Sistemas de Informação (CAPSI'2022)*, pp. 177–186, Nov. 2022.

[22] N. Bugshan, I. Khalil, and A. Kalapaaking, "Intrusion detection-based ensemble learning and microservices for zero touch networks," *IEEE Communications Magazine*, 2023.

[23] Q. Du, L. Zhao, F. Tian, and Y. Han, "Trace-based anomaly detection with contextual sequential invocations," in *International Conference on Database and Expert Systems Applications*, 2023.

[24] Y. Gan, G. Liu, X. Zhang, Q. Zhou, and J. Wu, "Sleuth: A trace-based root cause analysis system for large-scale microservices with graph neural networks," in *Proceedings of the 28th ACM Symposium on Operating Systems Principles*, 2023.

[25] L. Pham, H. Zhang, H. Ha, F. Salim, and X. Zhang, "Rcaeval: A benchmark for root cause analysis of microservice systems with telemetry data," in *ACM Companion of ICSE*, 2025.

[26] R. Ren, Y. Wang, F. Liu, Z. Li, and G. Tyson, "Grace: Interpretable root cause analysis by graph convolutional network for microservices," in *IEEE/ACM ICSE 2023*, 2023.

[27] A. Oliner and J. Stearley, "Bgl log dataset," 2007. Accessed: 2025-04-29.

[28] W. Xu *et al.*, "Hdfs log dataset," 2009. Accessed: 2025-04-29.

[29] S. Zhou, X. Zhan, L. Li, and Y. Liu, "Effective anomaly detection for microservice systems with real-time feature selection," *Asia-Pacific Software Engineering Conference (APSEC)*, 2023.

[30] M. Tavallaee *et al.*, "Nsl-kdd dataset," 2009. Accessed: 2025-04-29.

[31] N. Moustafa, "Ton_iot datasets," 2019. Accessed: 2025-04-29.

[32] J. Mante and K. Kolhe, "Ensemble of tree classifiers for improved ddos attack detection in the internet of things," *Mathematical Modelling of Engineering Problems*, vol. 11, no. 9, pp. 2355–2367, 2024.

[33] N. Moustafa and J. Slay, "Iot-botnet dataset," 2019. Accessed: 2025-04-29.

[34] M. Steidl, B. Dornauer, and M. Felderer, "How industry tackles anomalies during runtime: Approaches and key monitoring parameters," in *50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2024.

[35] S. Gao *et al.*, "Deathstarbench: A benchmark suite for cloud microservices," 2020. Accessed: 2025-04-29.

[36] C. Zhang, Z. Dong, X. Peng, and B. Zhang, "Trace-based multi-dimensional root cause localization of performance issues in microservice systems," *Proceedings of the IEEE International Conference on Software Engineering*, 2024.

[37] P. Fusco, A. Montefusco, G. P. Rimoli, F. Palmieri, and M. Ficco, "Tinyml-based intrusion detection system for handling class imbalance in iot-edge domain using siamese neural network on mcu," in *Computational Intelligence in Pattern Recognition*, Springer, 2025.

[38] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Cicids 2017 dataset," 2017. Accessed: 2025-04-29.

[39] Z. Li, J. Chen, R. Jiao, N. Zhao, Z. Wang, S. Zhang, Y. Wu, L. Jiang, L. Yan, Z. Wang, Z. Chen, W. Zhang, X. Nie, K. Sui, and D. Pei, "Practical Root Cause Localization for Microservice Systems via Trace Analysis," in *Proceedings of the 2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pp. 1–10, June 2021. ISSN: 1548-615X.

[40] T. Zhou, C. Zhang, X. Peng, Z. Yan, and P. Li, "Tracestream: Anomalous service localization based on trace stream clustering with online feedback," in *IEEE 34th ISSRE*, 2023.

[41] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.

[42] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," vol. 16, pp. 321–357, 2002.

[43] G. Haixiang, Y. Li, S. Shang, H. Ming, Y. Wang, and B. Wu, "Learning from class-imbalanced data: Review of methods and applications," *Expert Systems with Applications*, vol. 73, pp. 220–239, 2017.

[44] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[45] J. Bergstra *et al.*, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms." http://hyperopt.github.io/hyperopt/, 2013. Available at http://hyperopt.github.io/hyperopt/.

[46] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, vol. 2, pp. 4–11, 2014.

[47] J. R. Quinlan, *Induction of decision trees*, vol. 1. Springer, 1986.

[48] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pp. 338–345, Morgan Kaufmann Publishers Inc., 1995.

[49] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422, IEEE, 2008.

[50] F. Angiulli and C. Pizzuti, "Fast outlier detection in high dimensional spaces," in *Proceedings of the European Conference on Principles of Data Mining and Knowledge Discovery*, pp. 15–27, Springer, 2002.

[51] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Proceedings of the Advances in Neural Information Processing Systems*, pp. 3146–3154, 2017.

[52] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[53] P. J. Rousseeuw and K. Van Driessen, "Fast and robust fixed-point algorithms for independent component analysis," *Technometrics*, vol. 41, no. 3, pp. 212–223, 1999.

[54] G. Developers, "Classification: Precision and Recall | Machine Learning Crash Course | Google Developers." Available at https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall (2024/04/12).

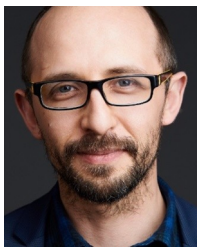
[55] L. M. Barata, "Admxgb - anomaly detection data tables." https://github.com/luisbarata-ubi/anomally-detection/tree/main/data, 2025. Accessed: 2025-07-29.
[56] V. Barnett and T. Lewis, *Outliers in Statistical Data*. Wiley, 3rd ed., 1994.
[57] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, pp. 15:1–15:58, July 2009.
[58] L. M. Barata, "Anomaly detection github repository - luís barata," 2025. Accessed: 2025-05-14.
[59] L. M. Barata, "Classified version of the tracerca dataset," May 2025.
[60] Y. Zhang, Y. Chen, J. Wang, and Z. Pan, "Unsupervised deep anomaly detection for multi-sensor time-series signals," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 2, pp. 2118–2132, 2023.
[61] D. F. N. Oliveira, L. F. Vismari, A. M. Nascimento, J. R. de Almeida, P. S. Cugnasca, J. B. Camargo, L. Almeida, R. Gripp, and M. Neves, "A new interpretable unsupervised anomaly detection method based on residual explanation," *IEEE Access*, vol. 10, pp. 1401–1409, 2022.
[62] J. Wang, J. Liu, J. Pu, Q. Yang, Z. Miao, J. Gao, and Y. Song, "An anomaly prediction framework for financial IT systems using hybrid machine learning methods," vol. 14, no. 11, pp. 15277–15286.
[63] B. Krawczyk, "Learning from imbalanced data: open challenges and future directions," vol. 5, no. 4, pp. 221–232.
[64] V. J. Hodge and J. Austin, "A Survey of Outlier Detection Methodologies," *Artificial Intelligence Review*, vol. 22, pp. 85–126, Oct. 2004.
[65] P. K. Donta, Q. Zhang, and S. Dustdar, "Performance measurements in the ai-centric computing continuum systems," *arXiv preprint arXiv:2506.22884*, 2025.
[66] L. M. Barata, E. Lopes, P. R. M. Inácio, and M. M. Freire, "Gambitrca: An approach for anomaly root- cause identification in microservices using game theory." Under revision, 2025.

LUIS M. BARATA was born in Castelo Branco, Portugal in 1977. Graduated in Computer Engineering at the Superior School of Technology, Polytechnic Institute of Castelo Branco, and MSc in Software Development and Interactive Systems in the same institution with a master thesis named "Informatics at the Service of the Elderly - an emerging theory," combining the use of IT by the elderly using a Grounded Theory approach. Currently pursuing a Ph.D. in computer engineering at the University of Beira Interior's Department of Computer Science in Covilhã, Portugal, his research focus has shifted to microservice architectures, capturing anomalies and determining their cause. Topics such as predictive algorithms, machine learning, and federated learning have recently attracted considerable attention.

Eng. Luis Barata is currently a Level 2 Engineer by the Portuguese Order of Engineers in the College of Computer Science and holds the title of Specialist in Development of Information Systems (Computer Science), awarded by Polytechnic Institute of Castelo Branco, Polytechnic Institute of Santarém, Polytechnic Institute of Guarda, Portuguese Order of Engineers (OE) and Portuguese Information Systems Association (APSI).

EURICO LOPES Computer Engineering, IST, Lisbon. MSc Computer Science, Essex University, Uk. PhD Information Management, Leeds Metropolitan University, UK. Professor at IPCB, Portugal. Whilst teaching and researching, Eurico has been working as a consultant with local enterprises. Since the 1990s, Eurico has developed work as a researcher in computer engineering and information systems. Prior to this, Eurico assisted the banking sector and the industry as an engineer, programmer, systems analyst, and trainer, and latterly as a project leader. Whilst teaching and researching, Eurico has established various collaborations, including several major projects with local enterprises, involving technology transfer across the organization for the development and implementation information systems.

For many years Eurico has been a member of Ordem dos Engenheiros, Senior Member in the IEEE and ACM; also working with CCISP (Conselho Coordenador Institutos Superior Politécnicos) as a consultant on ICT and network projects, including work on PRODEP I & II. Eurico Lopes was convened as an observer in the users panel on FCCN (Fundação Computação Cientifica Nacional) and as a member of the Installation Committee Technological and Management School in Castelo Branco and Idanha-a-Nova. Since April 2013, Eurico has been appointed as an Expert at the Agencia de Avaliação e Acreditação do Ensino Superior (Agency for Assessment and Accreditation of Higher Education - A3ES). Research Interests: Project Management, Business Analytics, Decision Support Systems, Research Methodology and Systems Thinking.



PEDRO R. M. INÁCIO (Senior Member, IEEE) is an associate professor of the Department of Computer Science at the University of Beira Interior (UBI), which he joined in 2010. Lectures subjects related with information assurance and (cyber)security, to graduate and undergraduate courses, namely to the B.Sc., M.Sc. and Ph.D. programmes in Computer Science and Engineering. He holds a 5-year B.Sc. degree in Mathematics/Computer Science and a Ph.D. degree in Computer Science and Engineering, obtained from UBI, Portugal, in 2005 and 2009 respectively. The Ph.D. work was performed in the enterprise environment of Nokia Siemens Networks Portugal S.A., through a Ph.D. grant from the Portuguese Foundation for Science and Technology.

He is an IEEE senior member, an ACM professional member and a researcher of the Instituto de Telecomunicações (IT). His main research topics are information assurance and security, computer based simulation, and network traffic monitoring, analysis and classification. He frequently reviews papers for IEEE, Springer, Wiley and Elsevier journals. He is a member of the Technical Program Committees of flagship national and international workshops and conferences, such as ACM SAC, IEEE NCA, IFIPSEC or ARES. He is also a Senior Editor for IEEE Access.

MÁRIO M. FREIRE (Member, IEEE) received a five-year BSc degree in Electrical Engineering and a two-year MSc degree in Systems and Automation in 1992 and 1994, respectively, from the University of Coimbra, Portugal. He received the PhD degree in Electrical Engineering in 2000 and the Habilitation title in Computer Science in 2007 from the University of Beira Interior (UBI), Portugal. He is a full professor of Computer Science at UBI, which he joined in October 1994. He started his research activities in October 1991 at the University of Coimbra, having worked within the European Project EEC RACE 2011 TRAVEL, within which he carried out a one-month internship in April 1993 at the ALCATEL SEL AG (now Nokia Networks) Research Center in Stuttgart, Germany. His main research interests fall within the area of computer systems and networks, including network and system virtualization, cloud and edge computing, computer security and privacy, and applications of AI in computer systems and networks. He is the coauthor of six international patents, co-editor of eight books published in the Springer LNCS book series, and co-author of more than 130 papers in international journals and conferences. He serves on the editorial board of the ACM SIGAPP Applied Computing Review, as an associate editor of the Wiley Security and Privacy Journal and the Wiley International Journal of Communication Systems, and as an editor of IEEE Communications Surveys and Tutorials in 2007–2011. He served as a technical program committee member for several IEEE international conferences and is co-chair of the track on Computer Networking of ACM SAC 2025. Dr. Mário Freire is a chartered engineer by the Portuguese Engineering Association and is a member of the IEEE Computer Society and the Association for Computing Machinery.