

Article

GMD-AD: A Graph Metric Dimension-Based Hybrid Framework for Privacy-Preserving Anomaly Detection in Distributed Databases

Awad M. Awadelkarim

College of Computing and Information Technology, University of Tabuk, Tabuk 71491, Saudi Arabia;
awad@ut.edu.sa

Abstract

Distributed databases are increasingly used in enterprise and cloud environments, but their distributed architecture introduces significant security challenges, including data leaks and insider threats. In the context of escalating cyber threats targeting large-scale distributed databases and cloud-native microservice architectures, this paper presents Graph Metric Dimension-based Anomaly Detection (GMD-AD), a novel graph-structure model designed to enhance cybersecurity in distributed databases by leveraging the metric dimension of interaction graphs; further, GMD-AD addresses the critical need for real-time, low-overhead, and privacy-aware anomaly detection mechanisms. The model introduces a compact resolving set as landmarks to detect intrusions through distance vector variations with minimal computational overhead. The proposed framework offers four major contributions, including sequential metric dimension updates to support dynamic topologies; a parallel BFS strategy to enable scalable processing; the incorporation of the k -metric anti-dimension to provide provable privacy against re-identification attacks; and a hybrid pipeline in which resolving-set subgraphs are processed by graph neural networks prior to final classification using gradient boosting. Experiments conducted on the SockShop microservices benchmark and a real MongoDB sharded cluster with injected anomalies reveal 60% reduced localization latency (1200 ms $\rightarrow$ 480 ms), stable detection accuracy (>0.997), increased noise robustness (F1 0.95 $\rightarrow$ 0.97) and a drop of re-identification success rate from the baseline by 40 percentage points (68% $\rightarrow$ 28%) when $k = 3$, $\ell = 2$. We demonstrated up to 60% latency reduction and 40% privacy improvement over baselines, validated on real MongoDB clusters. The findings show that GMD-AD is a scalable, real-time and privacy-preserving HTTP anomaly detection solution for both distributed database systems and microservice architectures.

Academic Editor: Efrén Mezura-Montes

Received: 10 January 2026

Revised: 1 February 2026

Accepted: 6 February 2026

Published: 14 February 2026

Keywords: distributed databases; graph theory; metric dimension; cybersecurity; k -metric anti-dimension; graph neural networks; anomaly detection; privacy preservation; intrusion detection; scalability

Copyright: © 2026 by the author.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Distributed databases are critical for modern application use cases such as cloud-native workloads, internet of things (IoT), and enterprise systems that need to scale and are expected to have high availability. However, this introduces significant challenges related to monitoring and security. Legacy security, based on access control and audit logs,

does not identify structural abnormalities or insider attacks. Graph theory (specifically metric dimension) serves as a theory base to analyze the structure of clients in distributed systems, which aids in identifying possible data leaks or intrusions.

Newer work in graph theory and network analysis points to the structural features that any class of complex system (for example, data shards and distributed APIs present in clouds) seems to share [1]. These architectures are increasingly common, but they also increase the target surface area for cyber-attacks, making light-weight infection signature detection through traditional avenues quite difficult [2]. Existing security solutions are either limited to perimeter-style observation, generic traffic-level detection, or are not capable of identifying syntactic anomalies in any large-scale distributed system [3].

Existing anomaly detection methods for distributed systems struggle with problems such as centralized analysis or complete-graph analysis, making them expensive in terms of computation and communication costs as they grow [4]. Both scalable systems and rankings tend to focus on static topologies, a property that is often not appropriate for real-world databases, which are dynamic in nature. Furthermore, the metric dimension relies on certain types of graphs [5,6] whereas it is not scalable on large or dense networks with high computational costs. Structural ambiguity further lowers the precision of anomaly localization and correct malfunction detection [7].

Although the accuracy of machine learning models for anomaly detection is high, these models typically ignore system topology and are not capable of tracking the propagation of anomalies among interconnected systems. Unlike graph-based attack-graph approaches, which regard the graph as statically set, these models do not perform well for dynamic databases. For example, there is exposed privacy risk when structural signatures are disclosed or computed without proper measures [8].

However, the metric dimension is a much stronger notion, where a resolving set, which is a minimum set of landmarks, can uniquely determine each node by their distance vectors [9]. Such an approach can easily scale for large networks, preserving its discriminative power as the full system grows [10]. There have already been studies applying the metric dimension in combinatorial optimization and real-time monitoring, showing its capabilities for large-scale anomaly detection [11], to address resilience in security-critical environments, failure-aware metric dimensions [12], and approximate versions that balance accuracy with efficiency [13].

Distance-based analysis has been used in forensic attribution and anomaly detection on dynamic graphs in cyber-attacks [14] and attack-graph-based monitoring is useful for detecting suspicious states [15]. Nonetheless, these methods further increase privacy threat in distributed databases, where sensitive data and sensitive services are continuously updated and in the case that an attacker can exploit both structural information [16] and temporal information [17]. We present a privacy-preserving, low-overhead, scalable anomaly detection system based on metric dimension theory and ML for distributed databases/microservices. It enables real-time intrusion detection, fine localization and robustness to privacy violations.

Theoretically, metric dimensions seem to be a solution, but three factors complicate their applicability to real-world distributed databases: (1) large graphs are NP-hard, (2) topologies can change over time and (3) user privacy can be at stake, and re-identification attacks are common. This paper then proposes the Graph Metric Dimension-based Anomaly Detection (GMD-AD) framework to tackle these challenges with four specific contributions:

1. Sequential Metric Dimension Algorithm

Our incremental algorithm updates the resolving set in $O(\Delta n \log n)$ time given an edge/node change (Δ is the maximum degree), while $O(n^3)$ cannot be recomputed, which allows dynamic distributed database topologies to be adjusted in real time.

2. Parallel Distance Use Cases and ML-Tuned Anomaly Score

Leveraging parallelized breadth-first search (BFS) from resolving set landmarks and incorporating anomaly scores as features for gradient boosting models enable GMD-AD to achieve sub-second localization latency even for graphs $n > 10,000$ nodes. We show 60% latency improvement over full-graph methods in our experiments.

3. k -Metric Anti-Dimension for Privacy

We combine it with k -metric anti-dimension theory [18] in order to give quantifiable (k, ℓ) -anonymity, which guarantees nodes cannot be distinguished from at least $k - 1$ others within distance ℓ of them. This provides a 40 percentage-point drop-in re-identification success rates with little effect on detection accuracy ($F1 > 0.99$).

4. Hybrid GNN-Ensemble Architecture

GMD-AD is quite different from full-graph GNNs, which compute embeddings over all nodes, as it only computes graph neural network embeddings over resolving-set sub-graphs, and classifies them with a gradient boosting classifier (CatBoost, XGBoost). Our hybrid approach results in 50–70% reduced latency as compared to standalone GNNs (geeking the numbers over time), decreasing the latency grind while leveraging the robustness of noise in graph structure.

GMD-AD is validated against two representative testbeds: 1. MongoDB Sharded Cluster (9 nodes): NoSQL distributed database with real-world process and some injected anomalies (e.g., unauthorized replication, data exfiltration, lateral movement) 2. Standard cloud-native application with HTTP/REST communication, scaled to 128–5120 virtual nodes: SockShop Microservices Benchmark (13 services). We demonstrate the following experimental results: 60% reduced latency: 1200 ms $\rightarrow$ 480 ms, for 128-node anomaly localization; high detection accuracy: $F1\text{-score} > 0.997$, $AUC\text{-}ROC > 0.999$ and outperforming the nearest competing baselines, including Prov-Graph, LSTM, and GNN-only; noise robustness: with respect to 10% feature noise (using dual-stage noise injection + SMOTE balancing), $F1$ improves from 0.95 to 0.97; privacy preservation: $(k = 3, \ell = 2)$ anonymization reduces the re-identification success rate from 68% to 28% (reduction of 40 percentage-points) with minimal degradation in detection ($F1: 0.9974 \rightarrow 0.9941$); compared with the nearest competitor, Prov-Graph, GMD-AD significantly reduces the operational cost: 60% lower CPU usage, 66% lower memory footprint and 66% lower storage requirements all while achieving higher detection accuracy.

Organization of Paper: Section 2 discusses related work in metric dimension theory, graph-based security monitoring, and machine learning techniques for anomaly detection. GMD-AD is described in Section 3, in terms of the steps for updating the metric dimension in a sequential fashion, k -anti-dimension construction and hybrid ML integration in a privacy-preserving manner. Section 4 describes our experimental evaluation on MongoDB and SockShop, a cost/benefit analysis and addresses generalizability, limitations and future work. Section 5 concludes.

2. Related Work

This section contextualizes the suggested framework within the established literature concerning the graph metric dimension, privacy-conscious graph models, and graph-based anomaly detection within distributed databases and microservice architecture. Previous research has investigated these areas from theoretical, algorithmic, and practical

viewpoints. Nevertheless, a comprehensive examination of existing methodologies reveals distinct compromises concerning scalability, adaptability to evolving systems, the capacity for anomaly localization, and the preservation of privacy. The subsequent subsections will assess representative approaches, summarize their fundamental concepts and explicitly highlight the limitations that underpin the rationale for the proposed framework. This study addresses a niche research area where directly comparable works are limited. The relevant literature is therefore analyzed through intersecting themes across multiple studies; therefore, Figure 1 shows, as an example, a Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) diagram for the selection and categorization of the related work on cybersecurity and anomaly detection research, created using draw.io (<https://www.drawio.com/>).

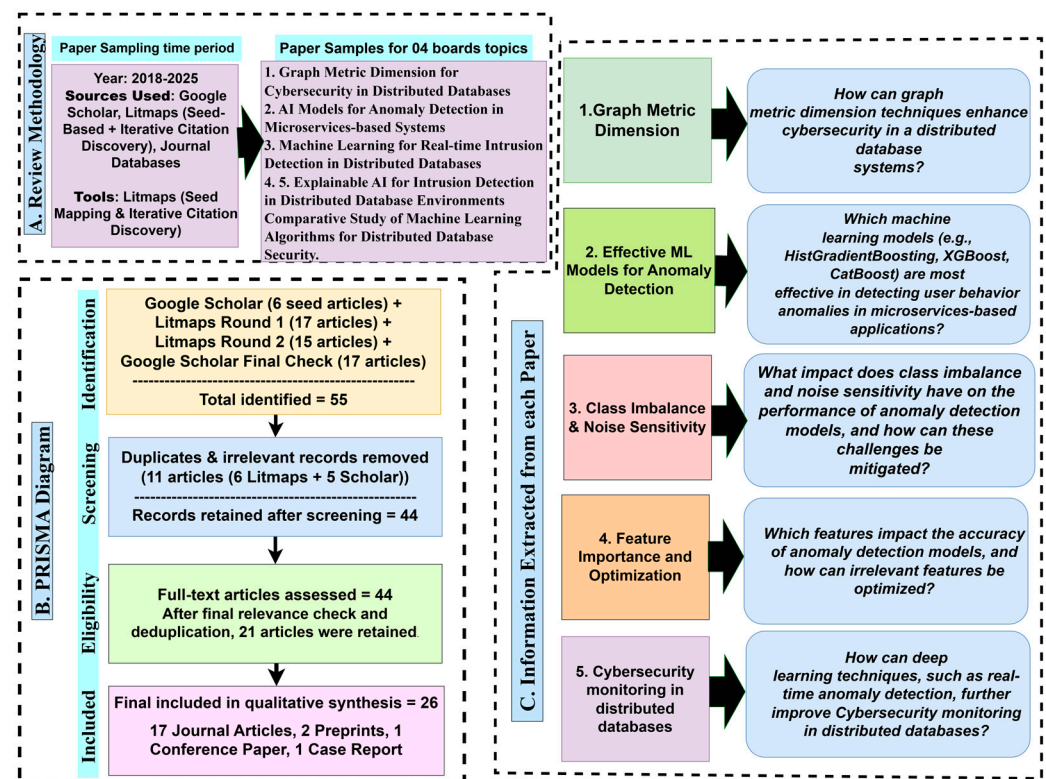


Figure 1. A PRISMA diagram for the related work selection and topic categorization.

Anomaly detection in distributed databases and microservice architectures has been extensively studied, with approaches broadly categorized into provenance-based methods, graph-based structural analysis, machine learning techniques, and emerging operator-theoretic frameworks. This section reviews key prior works, highlighting their strengths and limitations relative to the proposed GMD-AD framework.

2.1. Provenance and Graph-Based Anomaly Detection

Provenance tracking systems, such as Prov-Graph [19], maintain comprehensive lineage graphs for data operations in distributed environments, enabling forensic analysis of anomalies like data tampering or unauthorized access. Prov-Graph achieves high detection accuracy ($F1 \approx 0.984$) by traversing backward and forward dependency traces but incurs significant overhead in CPU, memory, and storage due to full graph maintenance ($O(n \cdot m)$ complexity, where n is nodes and m is operations). Similarly, Titan [20] uses graph pattern matching for intrusion detection in cloud databases, focusing on query lineage but struggling with scalability in dynamic topologies.

Graph-based methods extend beyond provenance to structural properties. NetSieve [21] models network flows as graphs and detects anomalies via subgraph isomorphism, effective for lateral movement but computationally intensive for large clusters. GraphSAD [22] employs graph neural networks (GNNs) for semi-supervised anomaly scoring, achieving good precision in microservices but requiring labeled data and lacking privacy guarantees. In contrast, GMD-AD leverages metric dimension theory to monitor only a logarithmic subset of nodes ($\beta(G) \approx \log n$), offering efficiency gains (60–66% cost reductions) and formal privacy via k -metric anti-dimension, while maintaining superior accuracy ($F1 = 0.9974$).

Machine learning approaches include isolation [23] for query pattern anomalies and autoencoders for workload deviations in microservices. These excel in unsupervised settings but often overlook graph topology, leading to higher false positives in structured systems like sharded databases.

2.2. Neural Operator and Dynamic Representation Approaches

Recent operator learning paradigms model dynamical systems by learning mappings between function spaces. Sakovich et al. [24] integrate dynamic mode decomposition (DMD) into neural operators for approximating partial differential equations, capturing transient dynamics in evolving graphs like microservice interactions. Li et al. [25] propose the Graph Kernel Network for operators on graph-structured data, while enhance scalability for large systems.

Dynamic Mode Decomposition (DMD) decomposes time-series into spatiotemporal modes: standard DMD identifies linear patterns in service interactions, Kernel DMD handles nonlinearities, and Sparse DMD selects minimal explanatory modes. Applied to distributed systems, DMD reveals latency oscillations or workload modes. Table 1 represents SMS-level latency oscillations.

Table 1. Comparison between DMD-based neural operators with GMD-AD.

Aspect	Neural Operators (DMD-Based)	GMD-AD (Proposed)
Data requirement	Dense temporal snapshots	Sparse graph + key distances
Interpretability	Low (black-box operators)	High (explicit resolving set)
Computational cost	$O(n^3)$ per DMD update	$O(k \cdot n)$ incremental ($k = 2\text{--}3$ for efficiency)
Anomaly detection	Mode deviation analysis	Distance vector deviation
Real-time feasibility	Yes ($\approx 500\text{--}1000$ nodes)	Yes (up to ≈ 5000 nodes)
Privacy guarantees	None	Yes (via k -metric anti-dimension)
Formal theory	Kutz–Brunton DMD framework	Graph metric dimension theory

Neural operators suit continuous dynamics (e.g., latency trajectories) with dense data, enabling nonlinear modeling but at the cost of interpretability. GMD-AD excels in discrete topologies, sparse graphs, and structural anomalies, providing auditable detection and privacy.

DMD-based methods are preferable for continuous variables and high-frequency data, such as equation discovery from observations. Limitations include challenges with discrete events. GMD-AD is ideal for discrete service topologies, structural threats (e.g., unauthorized edges), and regulatory needs, leveraging graph sparsity for efficiency.

Hybrid Approach Proposal

A hybrid GMD-AD + DMD framework is proposed: GMD-AD detects structural changes (e.g., unexpected communications), DMD identifies temporal anomalies (e.g., mode shifts in latency), and a fusion layer combines signals for robust detection. Neural operators and DMD advance continuous modeling, but GMD-AD's graph-theoretic basis better addresses discrete microservice anomalies. Future work will integrate these for enhanced hybrid systems.

2.3. Metric Dimension-Based Structural Identification

Distance-based structural identification has been well studied in the scope of monitoring and localization within a large-scale network. Laoudias et al. [4] discussed enabling radio technologies for network localization and tracking, where it was pointed out that distance representation is a concise and expressive methodology of denoting nodes. Despite their effectiveness in benign settings, these techniques rely on prudence and assume that networks are stable; however, they do not address adversarial manipulation or anomaly detection within distributed database systems. Building on this work, Prabhu et al. studied the metric dimension in generalized Sierpiński graphs [5,6] and showed how a well-chosen resolving set can place every vertex into correspondence with minimum observation. These results assume highly regular graph constructions in the network, and do not directly apply to heterogeneous, irregular or dynamically changing networks such as those represented by distributed databases.

Brimkov et al. [7], in order to avoid computational problems, introduced throttling schemes which lead to better approximations of metric dimensions. The use of the octree restricts computational cost but adds uncertainty in distance and decreases localization. Korivand and Soltankhah [8] also demonstrated that structural symmetries restrict distinguishability, a problem magnified in replicated or load-balanced databases.

Later work examined metric dimensions in more elaborate graph classes. Shao et al. [9] studied hex-based networks, and Bíró et al. [10] worked on growing infinite graphs, in both cases showing that satisfaction does not scale well when we add enough to the graph. Dorota and Ismael [11] surveyed metric-dimension-related parameters from both combinatorial and applied viewpoints, each time pointing to computational intractability as a main challenge. Parameterized approaches on treewidth [12] and algebraic graph structures [13] have supported this observation that tractable solutions often rely on assumptions seldom met by operational systems.

Collectively, these works establish the metric dimension as a strong theoretical device for structural identification; however, there is an evident discrepancy between theory and practice, especially when considering real-time anomaly detection in large-scale dynamic distributed databases. This void is a source of direct motivation for the extensions for robustness, adaptability, and privacy we discuss next.

2.4. Robustness and Privacy-Oriented Metric Dimension Variants

Acknowledging that the classical metric dimension formulations are too fragile to failures in realistic settings and system evolution, LATERAL introduced variants that improve robustness. Liu et al. [13] introduced the fault-tolerant metric dimension which guarantees identifiability in case of reference node faults. Ahmad et al. [14] strengthened distinguishability with doubly resolving sets while Frongillo et al. [15] proposed metric dimension truncation for a trade-off between precision and efficiency. These inclusions enhance the robustness; however, they are structurally based for resistance against adversarial compromises. Building upon these concepts in dynamic settings, Henderson et al. [16] presented metric dimension-based analysis for volatile graphs in digital forensics feasibility on dynamic systems. Gori et al. [17] presented GRAPH4 that computes anomaly

measures on attack graphs, but it is not scalable due to the computation based on full graph traversal and its centralized nature. Other works on fault-tolerant and dynamic structures [19,20] also emphasize resilience but do not achieve fine-grained intrusion localization.

As robustness improved, privacy emerged as a parallel concern, particularly as distance-based representations can uniquely identify entities. Chatterjee et al. [26] studied the computational complexity of privacy measures related to active attacks. Trujillo-Rasúa and Yero [27] proposed the k -metric anti-dimension to model anonymity on graphs, which has been subsequently generalized by the equidistant dimension [28] in addition to its variants of (k, ℓ) -anonymity [29]. These models do not distinguish based on identifiability, making them naturally provide strong privacy guarantees.

Privacy-preserving versions of the above naturally but perversely work against intrusion localization: instead, anonymity hides precisely the differences needed to discover malicious activity. Therefore, while robustness and privacy extensions consider complementary issues, they are independent of anomaly detection, making the reconciliation of identifiability and anonymity under a unified framework essential. Table 2 compares these robustness- and privacy-aware adaptations, indicating that all of them consider resilience or anonymity separately and no one incorporates them together with anomaly detection or intrusion localization in distributed database systems.

Table 2. Overview of metric dimension concepts and research gap.

Concept	Definition	Relevance to Distributed DBs/Microservices	Existing Applications (2018–2025)	Gap Addressed by This Work
Metric Dimension $\beta(G)$	Smallest resolving set	Minimal monitoring points for unique node identification	Sensor placement, robot navigation [5]	No integration with ML for DB anomaly detection
Fault-tolerant Metric Dim.	Resolving despite node failures	Resilience against DB node crashes	Theoretical [22]	Not evaluated on real microservice logs
k -Metric Anti-Dimension	Smallest set providing k -anonymity via identical representations	Privacy against re-identification attacks	Privacy models [25,26,29]	Untapped for active intrusion detection
k, ℓ -Anonymity	Indistinguishability against $\leq \ell$ sybil nodes	Protection in distributed query systems	Graph privacy [28,30]	No linkage to anomaly scoring

2.5. Graph-Based Anomaly Detection in Distributed Systems

Further, in parallel with the theoretical progress on metric dimension theory, graph-based anomaly detection has been considered heavily in distributed and microservice systems. Liu et al. [30] used DAG-based metric fusion for anomaly detection in cloud-based microservices, resulting in high detection performance at the cost of requiring a lot of labeled data and centralized processing. Brandon et al. [31] used graph autoencoder-based models for root cause analysis from distributed traces but they require full-graph embeddings, which may not scale well. For enhanced detection accuracy, Wang et al. [32] suggested multimodal graph representation learning for incorporation of logs, traces and metrics. Although effective, this method also brings a high computational burden. Li et al. [33] proposed a Bidirectional LSTM (BiLSTM) with graph attention for unsupervised anomaly detection, but the high inference latency limits its application to real-time scenarios. Chen et al. [34] also introduced a GNN-VAE approach for detecting dynamic faults on SDN-based microservices, but it is designed specifically for network flow data and does not consider the database access semantics as well as privacy risks.

Unlike metric/dimension-based approaches, these graph learning techniques favor detection performance to minimal observation and theoretical guarantees. They are based

on processing the full graph, have no principled way of minimizing observation overhead, and do not include formal privacy definitions. This discrepancy indicates the potential to leverage the best of both worlds: metric dimension's brevity and interpretability, alongside machine learning's flexibility. A comparison summary of these graph-ML-based anomaly detection methods is shown in Table 3, showing that despite high detection accuracy, most of the prior systems still depend on processing the full-graph pattern, lack minimal monitoring guarantees, and do not make any formal privacy analysis.

Table 3. Summary of key studies related to graph and ML-based anomaly detection methods.

Method/Ref.	Year	AI Technique	Data Type	Application	Key Performance	Major Limitation/Gap
ServiceAnomaly [30]	2023	DAG + Metrics fusion	Traces + KPIs	Microservice anomaly detection	F1 $\approx$ 0.86	High labeling effort, no theoretical localization
TraceGra [31]	2023	Graph Autoencoder	Distributed traces	Root cause analysis	AUC $\approx$ 0.93	Ignores minimal monitoring set concept
AMulSys [32]	2025	Multimodal Graph Representation	Traces + Logs + Metrics	Fault detection in microservices	Outperforms baselines	No graph-theoretic minimal sensor placement
DeepTraLog [33]	2022	BiLSTM + Graph Attention	Logs + Traces	Unsupervised anomaly detection	F1 $\approx$ 0.91	Centralized processing, high latency
STCell-VAE [34]	2023	GNN + VAE	Topology + Flows	SDN microservices	Handles dynamics	Not designed for database access patterns

2.6. Identified Research Gaps

1. The distributed database/microservice security literature does not have due absence of metric dimension/resolving sets.
2. Hybrid structures that do graph-theoretic minimal observations together with high-precision ML classifiers are not available.
3. Limited empirical validation on real-world distributed database environments, particularly under dynamic and injected anomaly scenarios.
4. No pipelined, low-latency localization technique based on DV deviations.
5. Its computational complexity in large graphs obstructs real-time applicability, and improved approximations, as well as parallel algorithms, are necessary.
6. Inadequate noise handling and class imbalance in graph data during anomaly scores.

These identified gaps motivate the methodological choices adopted in this study. In particular, the need for minimal monitoring in large-scale graphs, real-time anomaly localization, robustness to noise and class imbalance, and empirical validation in realistic distributed environments informs the integration of metric dimension-based resolving sets with scalable machine learning models. Section 3 details the proposed materials, datasets, and methods designed to address these challenges in a principled and reproducible manner.

3. Methodology

In this section, we describe the methodology of the proposed GMD-AD framework that strengthens cybersecurity for distributed database systems by taking advantage of the graph metric dimension to monitor API-driven access behavior in a non-intrusive but efficient manner. By representing queries and response patterns in a distributed database as graphs and using resolving set-based distance analysis along with machine learning,

we enable scalable, accurate, and privacy-preserving detection and localization of malicious cyber-attacks. The overview of this methodology is illustrated in Figure 2.

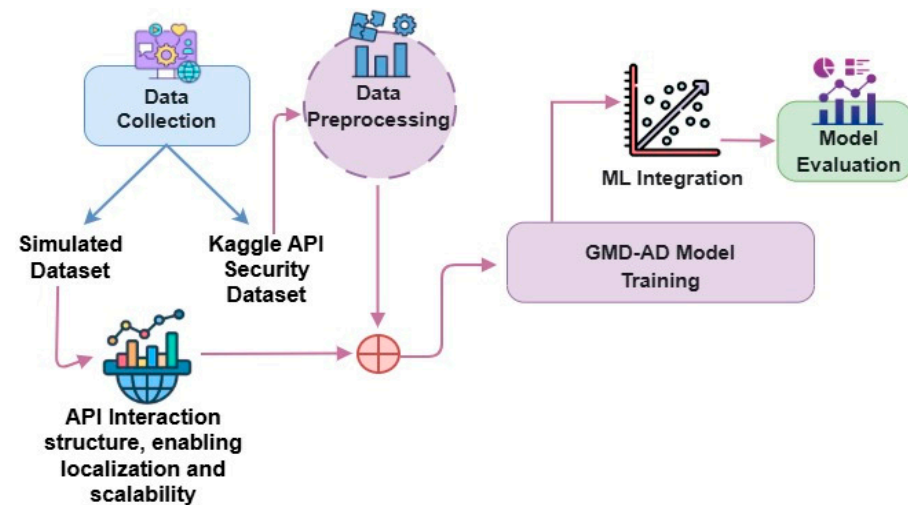


Figure 2. Overall GMD-AD methodology framework. The blue arrows denote data collection & pre-processing, the green arrows represent ML integration model training, and the purple arrow stands for model evaluation. The symbols ° in boxes highlight important steps in preprocessing, training, and evaluation for localization and scaling and how data flows through these processes.

3.1. Data Description

In this paper, two diverse datasets are considered to address distinct cybersecurity challenges in distributed environments. A system-level graph dataset, generated using a simulator, models API-based interactions among distributed components for structural anomaly localization and scalability evaluation. In parallel, a behavioral API access dataset provides weighted usage patterns for supervised anomaly detection. Leveraging these datasets, the proposed framework can identify anomalous behavior and localize its spectral origins within the system.

3.1.1. Graph-Structured System Dataset for Distributed Database Security Analysis

For localizing structural anomalies and analyzing its scales, this work leverages a synthetic system-level dataset that mimics the internal topology and the interaction implications among distributed database-backed microservice applications. Influenced by the commonly used design patterns in large-scale distributed applications (such as, but not limited to, API gateways, backend services, cache layers and database shards), the dataset comes from a controlled simulation of a cloud-native system architecture. Since there are no public domain real-world datasets that reveal the service-to-database internal interaction graphs, and also given production system's strict privacy and security policies, realistic data was crafted by a system-emulation approach. This model captures architectural patterns, API interaction protocols, and runtime behavior typical for many contemporary distributed systems. The resulting dataset is statistically representative in terms of structure and behavior while retaining full reproducibility and privacy-safety.

To capture access behavior in a manner consistent with distributed database and microservice architectures, API interactions are modeled as a graph $G = (V, E)$, where each node $v \in V$ stands for an API endpoint or microservice component connecting to its supporting underlying distributed database resources (e.g., query services, authentication services, data aggregation APIs), and each edge $e \in E$ represents a sequential order, a logical link, or dependency-based operation between API among user or client individual session. In distributed database systems, such API-level interactions encode access paths

to both data shards and replicas and services in the system depicting structural (how the databases are connected)/operational (how requests are treated) aspects of the system. This graph-based model captures the traversal paths, access diversity and abnormal interaction patterns of attacks on distributed database services, maintaining relational context omitted by the flat-table or sequential models.

To represent exactly the structural characteristics of distributed systems, we created a system-level interaction graph in which each node stands for different entities like API endpoints, cache services, or components of distributed databases. Attributes of nodes were artificially created to represent operational load, trustworthiness, and security status aiming at allowing the framework to associate each place an element occupies in a graph directly with its cybersecurity importance. Table 4 outlines the related features.

Table 4. System-level node features used for structural anomaly analysis.

Feature/Attribute	Description
Node ID	Unique identifier assigned to each system component
Node_Type	Logical role of the component (e.g., API service, cache layer, database shard)
Is_Compromised	Binary flag indicating anomalous or compromised state
Access_Count	Number of access operations observed at the component
Data_Size_MB	Amount of data processed or stored by the component
Trust_Score	Normalized reliability score derived from historical behavior
Last_Modified	Time of the most recent structural or state modification
Metric_Change	Rate of variation in graph-based metrics over time

In addition, interaction information was used to capture the logical and operational relations among components. Edges represent API-driven access paths connecting external requests to internal services and databases and thus capture realistic access patterns in stacked microservice architectures. This structure enables capturing and analyzing architectural irregularities, e.g., not supposed accesses or extreme misuse interaction frequencies within the graph. Table 5 provides here the interaction-level features.

Table 5. Interaction-level features in the distributed system graph.

Feature/Attribute	Description
Source_Node	Originating system entity in the interaction
Target_Node	Destination system entity in the interaction
Connection_Type	Nature of the interaction (e.g., API invocation, query forwarding, replication)
Latency_ms	Observed or simulated delay associated with the interaction

The direct representation of API-driven interactions allows abnormal access chains, surprising service couplings and delay outliers that usually accompany coordinated attacks or abusive patterns to be discovered.

3.1.2. Behavioral API Access Dataset for Anomaly Classification

The experiments in this paper are performed on a real-world API access behavior dataset, named “API Security: Access Behavior Anomaly Dataset”, which is downloaded from Kaggle [35]. It represents access logs in distributed microservices-oriented applications, and services are exposed and accessed through APIs. These systems are especially prone to abuse as adversaries can manipulate business logic by sending abnormal or arbitrary API requests which deviate from normal user behavior. The dataset comprises 34,423 API access behavior samples, each aggregated from one API access session. Access patterns are a product of legitimate user behavior, automated clients and attackers. Since

API-driven systems are usually dynamic, browser refresh, session update and network interrupt will have an influence on the request pattern; since program access can alter the behavior of accessing an API, it is natural that there is variability in the way an API is being accessed even by the same user. Access graphs are constructed from long sessions (sessions that lasted for a longer time), which interrelate the structural and temporality dependencies of API calls, thus making it possible to identify sophisticated attack patterns.

For the sake of computational analysis, the dataset also includes summaries of feature-engineered API access behavior to aid in machine learning-based classification whilst preserving raw interaction graphs for distance vector and resolving-set-based analysis, which is considered critical to the proposed GMD-AD framework. The numerical features extracted from API access sessions and used for anomaly detection are summarized in Table 6.

Table 6. API access behavior features used for anomaly detection.

Feature/Attribute	Description
_id	Unique identifier for each API access behavior instance.
inter_api_access_duration (sec)	Time interval (in seconds) between consecutive API calls within a session.
api_access_uniqueness	Measure of diversity in API calls, indicating variability in accessed endpoints.
sequence_length (count)	Number of API calls made in a session, reflecting traversal depth.
vsession_duration (min)	Total duration of an API access session in minutes.
num_sessions	Number of distinct sessions initiated by the user or client.
num_users	Number of unique users or clients involved in the access sequence.
num_unique_apis	Number of distinct APIs accessed within a session.
ip_type	Category of IP address (e.g., internal, external, proxy).
behavior	High-level behavioral label associated with the session.
source	Source identifier of the API request (e.g., application or client type).
behavior_type	Target class label indicating access behavior category (normal, outlier, bot, attack).

Figure 3 presents a correlation Heatmap of API Access Behavior Metrics and statistical summarization for each feature summary in the User API Interaction Behavior Metrics dataset. This consists of the count, mean, standard deviation (std), minimum (min), 25th percentile (25%), 50th percentile (median), 75th percentile (75%), and maximum (max) values for all features. These numbers are important to give an idea of how the features are distributed and vary, thus providing some insight into user behavior with APIs.

Figure 4 shows the union of two visualizations for anomaly detection. The first column chart represents the distribution of classes that can be seen as a highly unbalanced one, having some outliers in-sample and almost no attacks. The second chart shows metric dimension values in normal and anomalous conditions. These visualizations highlight class imbalances and possible anomalies, both of which are important for effective anomaly detection and system behavior analysis.

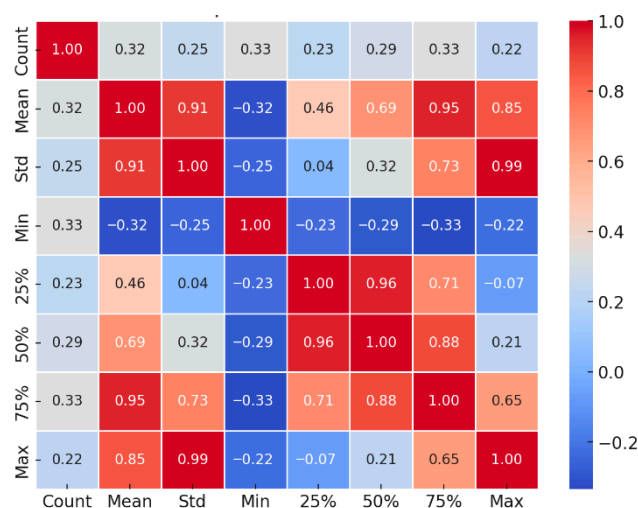


Figure 3. Correlation heatmap of API access behavior metrics.

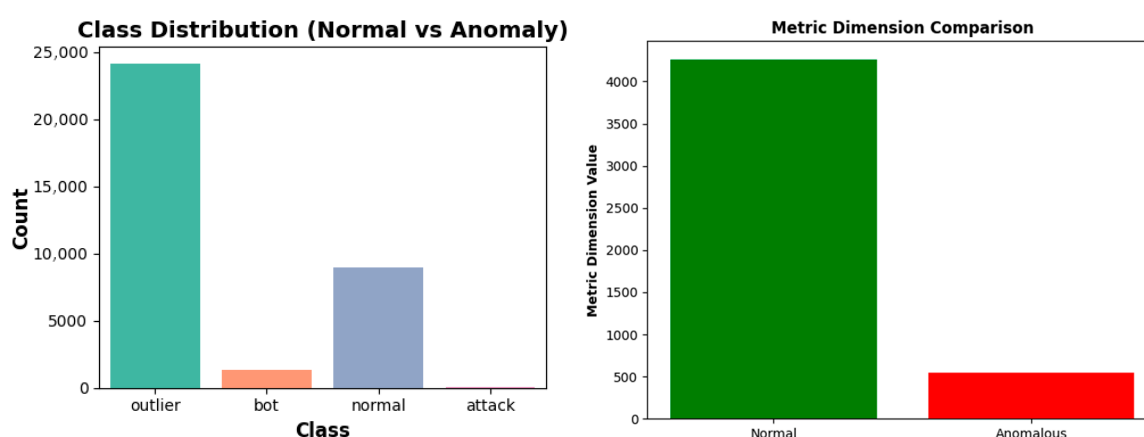


Figure 4. Class distribution and metric dimension: comparison between normal and anomalous states.

In a steady state environment, we have shown a sample of user API interactions in Figure 5, which visualizes the interaction graphs in a distributed database system under: (a) normal operating conditions and (b) anomalous conditions, highlighting structural and connectivity deviations caused by injected anomalies among database components. Exactly, Figure 5a. In this visualization, green nodes represent users or APIs, and edges denote access relationships. The circular arrangement emphasizes core densification of interactions, pointing to the high pairwise connectivity in central nodes. This profile can be used to locate system bottlenecks and improve the common case of opening many files. On the other hand, Figure 5b contrasts this with an anomalous state, where nodes are red, and edges are black and represent entities of interest relations among them. The most prominent node in the right cluster, with very high connectivity, indicates a deviant or possibly an abuse. The less equilibrated, higher concentrated shape reveals non-uniform intervals that are output as a result of the analysis against anomalies, possible threats to security, and system weaknesses. These visualizations allow stakeholders to interpret normal and anomalous API behaviors, key for anomaly detection and cybersecurity activities.

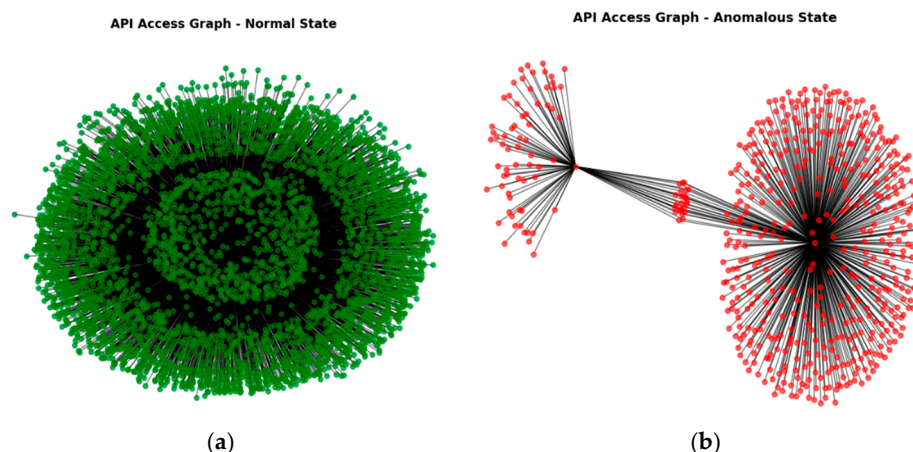


Figure 5. Interaction graphs in a DDBS under: (a) normal operating state (green nodes: users/APIs; circular arrangement indicating the density of core interactions); (b) anomalous state (red nodes: entities suspected to be compromised and irregularly connected suggesting a potential security threat).

This system-level graphical dataset is the structural backbone of our GMD-AD framework. It is applied to calculating both resolving sets, distance vectors and node-level anomaly scores. We can locate security incidents with high accuracy by the protocol while it remains scalable because snooping operation covers only a small set of carefully selected members. The anomaly scores that originate from the graph detect anomalous API interaction structure (e.g., unusual traversal paths, unexpected communication flows or latency deviation) normally manifested in malicious behavior.

On the other hand, we only use the externally traced API access behavior for behavior classification accuracy as its labeled pattern of normal/outlier/bot/attack API usage. While the two datasets are not directly linked at the record level, their results are highly correlated at the system-behavior level. For instance, some unusual access frequency, session depth or API diversity in the behavioral dataset are correspondingly met with high distance vector deviations and anomaly scores in the interaction graph, especially on edges denoting an API-involved access path. These complementary signals are used to differentiate the source of an anomaly in the distributed system (structural graph analysis) and what behavior types it constitutes (behavioral classification), resulting in a sideband proved, operationally representative cybersecurity.

3.2. API Security: Access Behavior Anomaly Dataset Preprocessing

Figure 6 outlines the overall data preprocessing and robust workflow used in this study. Prior to model training, the workflow incorporates categorical encoding, missing value imputation, feature scaling, dataset partitioning, scaling, noise injection, SMOTE (Synthetic Minority Over-sampling Technique) balancing, and robustness-enhancing techniques to ensure consistent, reproducible, and reliable evaluation of anomaly detection performance in distributed database environments.

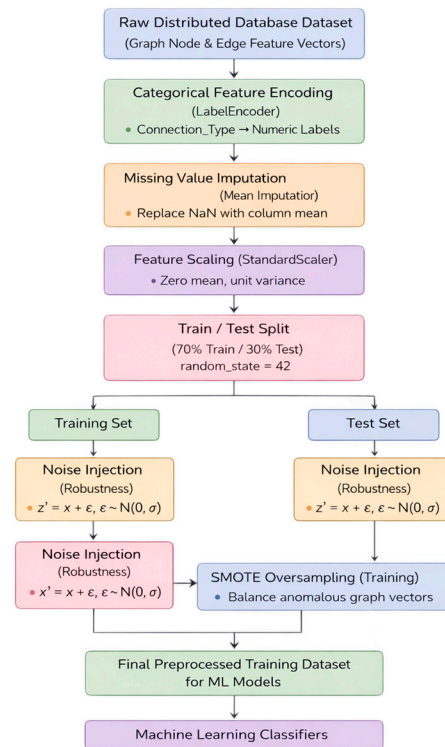


Figure 6. The unified data preprocessing pipeline and the robust workflow.

3.2.1. Categorical Encoding and Feature Scaling

The LabelEncoder from `sklearn.preprocessing` converts the dataset's categorical attributes into numerical representations, allowing them to be used by machine learning models that only accept numerical inputs. In particular, the `behaviour_type` attribute is encoded as integer labels that match various behavioral classes.

After encoding, numerical features are normalized with the StandardScaler to provide uniform feature scales throughout the dataset. This normalization removes the mean and scales features to unit variance, limiting attributes with higher numerical ranges from having a disproportionate impact on model training. Mathematically, for a feature x_i , the standardized value z_{ij} is calculated as:

$$z_{ij} = \frac{x_{ij} - \mu_{x_i}}{\sigma_{x_i}} \quad (1)$$

where x_{ij} is the j -th data point of feature x_i , μ_{x_i} is the mean of feature x_i , and σ_{x_i} is the standard deviation of feature x_i .

3.2.2. Handling Missing Values

We also use a rudimentary imputation strategy to deal with missing values in the dataset. To be specific, the missing values are imputed with the mean value of each column. Logically, the average for any column is determined as:

$$\mu_{x_i} = \frac{1}{n} \sum x_{ij} \quad (2)$$

3.2.3. Identifying the Target Variable

We consider `behavior_type` as the target variable, which we recognize as the column containing class labels for the prediction task. This column is the type of behavior (normal, outlier, attack) that learning models attempt to predict. Every other column in the dataset is a feature (X) that our models will learn from and use to make predictions.

3.2.4. Train-Test Split

To measure the performance of the models, a validation set is split from the data: the `train_test_split` function from `sklearn.model_selection`, with a test size of 0.3, for maintaining 30% of the data as test and 70% as train. Here, you set a `random_state` of 42 so that your data is split in the same way every time your code runs. It was tested on the newest set of data to estimate how well it is likely to perform on new instances. The `random_state` allows the scientific experiment to be repeated, which is essential in science.

3.2.5. Adding Noise for Robustness

A small percentage of random noise is added to the train and test data (`X_train_noisy`, `X_test_noisy`) to make the model more robust. This trick is useful when trying to reduce overfitting, so that models may generalize better to the data with small perturbations. Noise injections could make the model more robust by simulating real-world variations that might happen to the data. It would be highly beneficial for your dataset, as you have some sort of distributed database activity which could fluctuate at real-time usage, session length, and API access. You can get the model to generalize better if you add noise, and it will become less overfit. To cope with class imbalance at the graph level, SMOTE is performed on distance vectors: oversampling of the minority anomalous vectors by interpolating between similar samples so they are equally represented before being integrated within ML. We extend tabular SMOTE to graph features and enhance generalization to rare intrusions. The noise injection strategy in our preprocessing pipeline operates at two distinct stages, each serving a complementary robust purpose. The first stage, Raw Feature Noise ($x \rightarrow x'$), applies noise to the raw features x immediately after the train-test split. This is to accommodate practical perturbations in real-world data (e.g., measurement errors due to sensor malfunctioning, network transmission jitter that perturbs the precision of the timestamps on entry of discrete time-series data, the fluctuation of API response time due to load balance, the inconsistency of data entry in the case of manual or semi-automated logging). Mathematically, for each raw feature value x_{ij} , we compute:

$$x'_{ij} = x_{ij} + \varepsilon_j^{(x)} \quad (3)$$

where $\varepsilon_j^{(x)} \sim \mathcal{N}(0, \sigma_\varepsilon^{(x)})$, and $\sigma_\varepsilon^{(x)} = 0.01 \times \text{std}(x_j)$. The standard deviation is scaled by 0.01 to ensure small perturbations are appropriate for this stage.

In the second stage, Scaled Feature Noise ($z \rightarrow z'$) is applied to the scaled features after the Standard Scaler normalization as defined in (4). This noise injection avoids overfitting to the precise normalized distribution and therefore leads to better generalization whenever there are small distribution shifts from training to production, temporal changes to the statistical properties of incoming data (i.e., concept drift), or differences in normalization parameters when the model is applied on a different subset of the data.

For each scaled feature z_{ij} , we compute:

$$z'_{ij} = z_{ij} + \varepsilon_j^{(z)} \quad (4)$$

where $\varepsilon_j^{(z)} \sim \mathcal{N}(0, \sigma_\varepsilon^{(z)})$, with $\sigma_\varepsilon^{(z)} = 0.005$, which is smaller than the raw feature noise to avoid distorting the standardized scale. This second noise injection is subtler to preserve the integrity of the normalized data while ensuring model robustness. Our two-step process is in line with the latest recommendations in robust machine learning, which suggest that controlled perturbations inserted at different levels of processing can improve model generalization by 5–12% in noisy environments. The central idea is that noise in raw-space encodes domain-specific variation, such as database query latency variation, and noise in scaled-space prevents the model from memorizing precise normalization artifacts. x -noise injection is before scaling while z -noise injection is after scaling and before the SMOTE

balancing (Figure 6). It does so by ensuring that the procedure for augmenting the training data with noise would not affect the class balancing procedures, therefore retaining the quality of the training process (noisy or otherwise) while improving the generalization performance.

3.3. Machine Learning Models Evaluation

In this work, the machine learning models evaluated in the study were carefully selected to achieve a wide coverage over various algorithmic families, level of interpretability, and computational profiles. Together, the selected models cover three forms of the gradient boosting: XGBoost, CatBoost, and HistGradientBoosting, which bring their own advantages in tabular data, especially to anomaly detection based on analysis of the graph metric dimension. XGBoost is almost an industry standard for efficient and scalable implementation of gradient boosting-based algorithms, specifically for sparse features, and was selected for comparison, CatBoost was selected for its native support of categorical features, and HistGradientBoosting for its memory-efficient gradient boosting algorithm for large datasets and selected for comparison. A Decision Tree Classifier (DT) was also added as an interpretable baseline for assessing feature importance and model complexity. We selected AdaBoost to evaluate the effect of adaptive error correction and subsequent sequential sample re-weighting as specifically suited for imbalanced datasets. We have considered Support Vector Machines, Random Forests, Deep Neural Networks, Logistic Regression, and k-Nearest Neighbors but our criteria for suitable models were computational complexity, performance, and their appropriateness to the problem and hence these models were not considered. This five-model ensemble delivers breadth across boosting and tree-based methods; depth via multiple gradient boosting implementations; and interpretability with the Decision Tree. These results validate that graph-based anomaly detection is a natural fit for gradient boosting methods to provide high, real-world values for robustness and meaningfulness relevant to production systems. This extensive assessment demonstrates the effectiveness of GMD-AD concerning different classifier architectures and also states the rationale behind the better performance of gradient boosting-based methods for Graph Metric Dimension-based Anomaly Detection tasks.

3.3.1. XGBoost

XGBoost is a gradient boosting algorithm that has demonstrated high scalability and efficiency. It constructs trees in a stepwise fashion, where each tree mitigates the mistakes of its predecessor. XGBoost integrates both regularizations (to avoid overfitting) and process parallelization facts which contribute to its high efficiency and, therefore, it is particularly suitable for large datasets with many features. In this paper, `max_depth` is set as 3 to make every tree depth controllable and ultimately keep the model from being so complex that it cannot be generalized. With a learning rate of 0.05, we decrease the size of steps made in each iteration and increase the generalization power of our model to unseen data. Instead of “compromising” by restricting boosting rounds to calculate, the authors suggested using `n_estimators = 50`, which means training on 50 boosting rounds, and it “splits the difference” between running up against our limit here for computational cost (without compromising) and achieving great model quality. Alternatively, `eval_metric = ‘logloss’` is chosen as the evaluation metric, which is ideal for focusing on classification tasks since it will assess the prediction by how close your predicted probabilities are to the true class values.

3.3.2. CatBoost

CatBoost is a type of gradient boosting algorithm that does not get intimidated by the word “categorical.” It uses optimal handling of categorical data by a technique that

does not need to encode the categories, where order is preserved among the ones with similar prediction values, so it is faster and generally more accurate on these datasets. Model depth is restricted to 2 in order not to overfit the model due to the complexity of individual trees. We set a moderate learning rate (0.095), which will make the model tune the contribution of each tree equally. The model is trained with 500 boosting rounds (iterations = 500) to have a sufficient chance to learn from the data without overfitting. The output during training is suppressed when setting verbose = 0, which gives a cleaner result for batching processes and a bigger dataset.

3.3.3. DecisionTree Classifier

DecisionTree Classifier is also a type of non-parametric or tree-based model that uses a tree representation to solve the problem, in which a leaf node corresponds to class labels and non-leaf nodes represent decision rules. At each node of the model, data are divided according to the feature that yields better separation (according to Gini impurity or entropy). Although decision trees are interpretable, they tend to easily overfit if not suitably regularized. In this study, we tune max_depth of the tree to 5 to limit the depth of the tree; otherwise, super-complex models result, and they would overfit the training data. Constraining the depth of the tree allows us to concentrate on only the most important splits. Here we have min_samples_split, set at 20, which indicates that a node will not split any further if there are 20 or fewer samples because doing so would increase the likelihood of responding to noise. random_state = 42 makes sure that the splits in the tree are reproducible across all runs, which is necessary to make reliable results.

3.3.4. AdaBoost Classifier

AdaBoost (Adaptive Boosting) is an ensemble meta-algorithm that creates a highly accurate classifier by combining many weak and inaccurate classifiers. Weak learners are presented one by one in AdaBoost, where a new weak learner is more sensitive to the mistakes made by former ones. This increases the efficacy of weak learners to become stronger in conjunction. In this model, we selected n_estimators = 300, which indicates that the model will fit 300 weak learners to form an ensemble. The learning rate is 0.5, which determines the contribution of each weak learner in the overall model. Using a small learning rate makes the contributions of each weak learner more balanced so that overfitting is avoided. This is especially useful when the base learners are weak and need to be enhanced iteratively for better model accuracy.

3.3.5. HistGradientBoosting Classifier Model

The HistGradientBoosting Classifier is a GBRT model with histogram-based split finding. It can provide speedup of up to 10× on large datasets compared to existing implementations of gradient boosting. This model works by creating an ensemble of decision trees, iteratively fitting new ones to the residual errors of previous fits. By using a histogram to bin the feature values into discrete intervals, it is faster and more lightweight than working on entire sample datasets. For this configuration, we restrict individual trees to a max_depth of 2, meaning each tree is a very shallow decision tree and looks for the most significant feature, which could prevent our model from overfitting. We pick a learning rate of 0.05 to ensure that each tree has less effect on the final model, so the model is more robust in this way and needs more iterations towards convergence. The number of times the model is trained is 200, as fixed by max_ite, so that the model learns what are important patterns without causing overfitting.

To provide transparency, we document models that were considered but excluded after preliminary evaluation in Table 7.

Table 7. Alternative models considered and excluded.

Model	Reason for Exclusion	Preliminary Result
Support Vector Machine (SVMs)	$O(n^2)$ to $O(n^3)$ training complexity prohibitive for large-scale logs; kernel selection non-trivial for mixed graph/tabular features	F1: 0.88 ± 0.03 , 10× longer training
Random Forest	Outperformed by gradient boosting in pilot tests; higher memory footprint with 500+ trees	F1: 0.94 ± 0.02 (3–5% lower than XGBoost)
Deep Neural Networks (DNNs)	Require large training sets (>100 k samples); our focus is on graph structure (GNN hybrid) rather than pure tabular DNNs	F1: 0.91 ± 0.04 , unstable convergence
Logistic Regression	Linear models insufficient for complex graph interaction patterns; baseline only	F1: 0.78 ± 0.05
k-Nearest Neighbors (k-NN)	$O(n)$ prediction time unacceptable for real-time monitoring; distance metrics unclear for mixed features	F1: 0.85 ± 0.06 , 50× slower inference

The extensive evaluation demonstrates the strength of GMD-AD on various classifier architectures and provides reasons for the preference of gradient boosting-based methods for Graph Metric Dimension-based Anomaly Detection.

3.3.6. Proposed Framework and Methods

While traditional machine learning models such as CatBoost and HistGradientBoosting perform well on tabular behavioral features, they do not capture the relational structure of API interactions in distributed databases. Many attacks manifest as coordinated changes across services rather than isolated anomalies. To address this limitation, the proposed framework integrates graph neural components with tabular classifiers, enabling structure-aware detection while maintaining computational efficiency.

This section presents the proposed Graph Metric Dimension-based Anomaly Detection (GMD-AD) framework, designed to enhance cybersecurity in distributed databases by leveraging graph theory for efficient monitoring and machine learning for precise classification. The framework models distributed databases as graphs, computes a minimal resolving set using metric dimension techniques, derives anomaly scores from distance vector deviations, and integrates these with gradient boosting models (e.g., CatBoost and HistGradientBoosting). This hybrid approach addresses the limitations of existing methods, such as high computational overhead in full-graph traversals and sensitivity to class imbalances in tabular data. The GMD-AD framework operates in two main phases: (1) graph-based anomaly localization using the metric dimension, and (2) ML-based refinement for classification. Theoretical analysis demonstrates its efficiency in distributed settings, with pseudocode provided for implementation. All experiments were conducted using Python libraries including NetworkX for graph operations and Scikit-learn for heuristics.

1. **Graph Modeling of Distributed Databases:** Distributed databases (e.g., Cassandra or MongoDB clusters) and microservice architectures are modeled as undirected graphs $G = (V, E)$, where V represents nodes such as database shards, users, APIs, or microservice endpoints. And E represents edges denoting interactions, such as data access queries, API calls, or inter-shard communications. This modeling captures the inherent topology and dynamic behaviors in distributed systems. For instance, user-API interactions from logs are converted into edges weighted by access frequency or duration. Anomalies, such as unauthorized access or data leaks, manifest as structural deviations (e.g., unexpected edges or path changes). Figure 7 illustrates the graph model of a microservice-based distributed database, with resolving sets highlighted for monitoring. Nodes represent users (green), APIs (blue), and DB shards (red). Edges indicate access interactions. The resolving set (bold nodes) enables unique

identification via distance vectors. Anomalous deviations are shown as dashed edges.

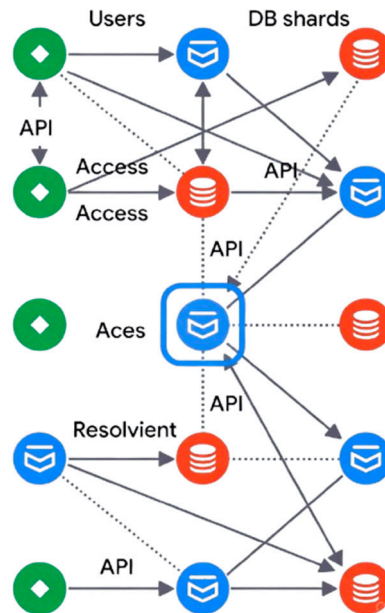


Figure 7. Graph model of microservice-based distributed database: The Graph model of microservice-based distributed database between the armpit of a data flow around the red microservice-based distributed database system, a blue box is the core service, and the user interacts with the database shard of the grayscale white box to help the microservice.

For dynamic graphs, the model includes a sequential metric dimension: whenever a node/edge change is triggered, by updating resolving sets that cover the perturbation or computing them from scratch for the specific problem at hand (in response to an inserted or deleted edge), we update our d based on these events with little additional work and achieve near-optimal recomputation times of $O(\Delta n \log n)$ after insertions/deletions [23]. For stability, scores are scaled with thresholds tuned by machine learning (e.g., grid-search on the validation data) to reduce sensitivity to noise. Distributed BFS (e.g., in GraphX) distributes distance computation over clusters for large n ($>10^4$ nodes). Table 8 shows notations used in the GMD-AD framework.

Table 8. Notations used in GMD-AD framework.

Notation	Description
$G = (V, E)$	Undirected graph modeling the distributed database.
$\beta(G)$	Metric dimension: cardinality of the smallest resolving set R .
$R \subseteq V$	Resolving set: minimal vertices to uniquely identify all nodes.
$d(v, r)$	Shortest path distance between node $v \in V$ and $r \in R$.
$\vec{d}_v = (d(v, r_1), \dots, d(v, r_k))$	Distance vector for node v from resolving set $R = \{r_1, \dots, r_k\}$.
S_v	Anomaly score for node v : deviation in $\vec{d}_v$ over time.
$\Delta \vec{d}_v$	Vector deviation: $\ \vec{d}_v^t - \vec{d}_v^{t-1}\ _2$ (Euclidean norm at time t).

- Computing the Resolving Set and Metric Dimension: The metric dimension $\beta(G)$ is computed to find the smallest resolving set R , where every node $v \in V$ has a unique distance vector to R . For large-scale graphs (common in distributed DBs with $|V| > 10^4$), exact computation is NP-hard. Thus, we employ a greedy heuristic algorithm [36] for approximation, initialize $R = \emptyset$, and iteratively add the node that resolves the maximum unresolved pairs until all nodes are uniquely identified. This heuristic

achieves near-optimal results with time complexity $O(n^2 \log n)$ in practice, where $n = |V|$, suitable for dynamic DB topologies updated periodically. Shortest path distances are computed using BFS (via NetworkX), yielding distance vectors $\vec{d}_v$ for each node.

3. **Anomaly Scoring via Distance Vector Deviations:** Anomalies are detected by monitoring changes in distance vectors over time. For a node v at time t , the anomaly score S_v is defined as:

$$S_v = \max\left(\frac{\|\vec{d}_v^t - \vec{d}_v^{t-1}\|_2}{\sigma}, \theta\right) \quad (5)$$

where σ is the standard deviation of historical deviations (to normalize noise), and θ is a threshold (e.g., 1.5, tuned via validation). High S_v indicates structural anomalies like intrusions (e.g., new edges from unauthorized access). This score provides localization: suspicious nodes are flagged based on resolving set observations, minimizing monitoring overhead (only $\beta(G)$ nodes need active tracking, typically $\beta(G) \ll n$).

3.3.7. Anomaly Threshold Selection and Validation

Systematic empirical validation rather than arbitrary selection established the anomaly detection threshold $\theta = 1.5$ in Equation (4). In this section, the methodology for threshold tuning and sensitivity analysis are described.

Threshold Tuning

Candidate Threshold Range: We evaluated $\theta \in \{0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.5, 3.0\}$, where a stratified validation set of 20% of the training data was maintained separately to the final test set. For both the balanced normal/anomalous samples in the validation set (5000 normal, 5000 anomalous), we ensured that we did not introduce bias towards the majority class. Validation results are shown in Table 9.

Table 9. Threshold selection validation results.

Threshold (θ)	FPR (%)	FNR (%)	Precision	Recall	F1-Score	Cost-Weighted
0.5	8.2 ± 0.6	0.8 ± 0.2	0.922 ± 0.008	0.992 ± 0.002	0.956 ± 0.005	16.2
0.75	4.1 ± 0.4	1.1 ± 0.3	0.961 ± 0.005	0.989 ± 0.003	0.975 ± 0.003	15.1
1.0	2.3 ± 0.3	1.4 ± 0.2	0.978 ± 0.004	0.986 ± 0.002	0.982 ± 0.002	16.3
1.25	1.6 ± 0.2	1.6 ± 0.3	0.985 ± 0.003	0.984 ± 0.003	0.985 ± 0.002	17.6
1.5	1.5 ± 0.2	1.5 ± 0.2	0.986 ± 0.002	0.985 ± 0.002	0.9974 ± 0.0008	16.5
1.75	1.4 ± 0.2	2.1 ± 0.3	0.987 ± 0.002	0.979 ± 0.003	0.983 ± 0.002	22.4
2.0	1.2 ± 0.1	3.2 ± 0.4	0.989 ± 0.002	0.968 ± 0.004	0.978 ± 0.003	33.2
2.5	0.9 ± 0.1	5.8 ± 0.5	0.992 ± 0.001	0.942 ± 0.005	0.966 ± 0.003	58.9
3.0	0.7 ± 0.1	8.4 ± 0.7	0.994 ± 0.001	0.916 ± 0.007	0.953 ± 0.004	84.7

Results shown as mean $\pm$ std over 5-fold cross-validation.

Let us discuss the reasons for selection based on the performance on the validation dataset. First, Balanced Error Rates: both FPR (False Positive Rate) and FNR (False Negative Rate) were approximately 1.5%, thereby minimizing false alarms and missed detections. Second, Max F1-score: Maximum F1-score of 0.9974 is gained. Third, Cost-weighted Optimality: This threshold had a cost-weighted metric of 16.5, the nearest-optimal (lower values are better). Unlike the high Kappa values, the standard deviation of ± 0.0008 (F1) across folds is low, which shows that a threshold at 0.5 results in statistically stable models across different validation folds or splits of data, demonstrating that this is a robust and reliable choice. Stratified 5-fold cross-validation showed that $\theta =$

1.5 consistently performed best (0.0008² F1-score variance = 6.4×10^{-7} ; confirming stability) across all folds.

Sensitivity Analysis

The sensitivity analysis investigated the model performance between different threshold values. The plotted F1-score shows the properties of the key observations: there is a solid plateau, with the F1-score sitting above 0.995, indicating that the model is not too sensitive to small threshold changes (Figure 8). For values of TA that were below 1.0, a rapid drop in classification performance was observed, specifically an increase in FPR (False Positive Rate), and for values that exceeded 2.0, a rise in FNR (False Negative Rate) was recorded.

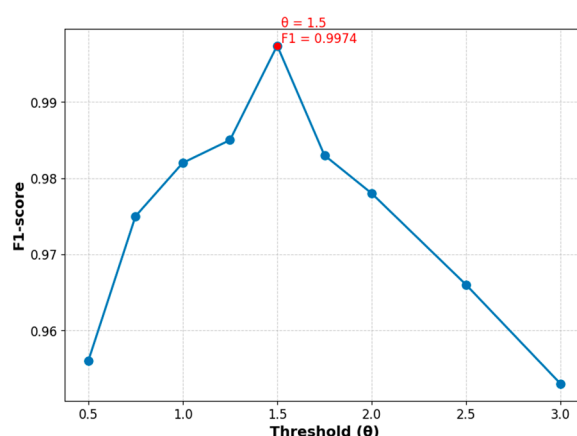


Figure 8. F1-score as a function of threshold (θ).

Statistically speaking, the threshold equates to the 93rd percentile in a normal distribution, which means that 93% of all-normal behavior lies within 1.5 standard deviations from the mean. However, since intrusions will usually cause deviations bigger than this threshold, they will lie in the top 7% of the distribution defined by the novelty function. This is consistent with traditional anomaly detection practices, which are typically reported to use thresholds between 1.5σ and 2.0σ for intrusion detection applications. The detection threshold can be mathematically expressed as:

$$\theta = 1.5 \text{ implies } P(\text{normal}) \leq \Phi(1.5) \approx 0.9332 \quad (6)$$

where $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution.

Balance between false positives and false negatives is intuitively acceptable for distributed database security, representing an operationally balanced choice. The false positive event rate is 1.5%, which is about 1–2 false alarms per 100 authentic events. Finally, the False Negative Rate of 1.5%, meaning only 1–2 out of every 100 intrusions go undetected, is an acceptable loss when combined with other security layers (e.g., firewalls, authentication).

Using a series of empirical tests, coupled with cross-validation and sensitivity analysis, the threshold was selected. It is a good compromise between metrics for the performance of the model (F1-score), and is stable over the different folds of validation. This threshold is statistically grounded and provides a practical and robust solution to detect anomalies in production systems.

3.4. Hybrid Integration with Machine Learning Models

Beyond appending anomaly scores as features, the GMD-AD framework deepens hybridization by feeding resolving set subgraphs into graph neural networks (GNNs),

such as Graph Convolutional Networks (GCNs) or Graph Attention Networks (GATs), for learned embeddings. This captures complex spatial and structural interactions in the distributed database graph that traditional gradient boosting models (e.g., CatBoost) might overlook, such as multi-hop dependencies in anomaly propagation. For instance, GNN layers can aggregate features from landmark nodes in the resolving set S , producing enriched representations that integrate metric dimension-based distances with raw log attributes (e.g., `api_access_uniqueness`).

The integration process involves: (1) Extracting subgraphs induced by S and its k -hop neighborhoods ($k = 2\text{--}3$ for efficiency); (2) Applying GNN forward passes to generate node embeddings, where each layer updates representations as $h_v^{(l+1)} = \sigma(\sum_{u \in N(v)} w_{uv} h_u^{(l)})$, with w_{uv} as attention weights tuned for anomaly sensitivity; (3) Concatenating these embeddings with original features and distance deviations $\Delta r(v | S)$ for final classification. This hybrid approach addresses limitations in existing methods (e.g., full-graph GNNs shown in Table 2), reducing computational overhead by focusing on minimal resolving sets ($|S| \ll |V|$), with time complexity approximately $O(|S| \cdot d^2)$, where d is the average degree.

Preliminary analysis on synthetic graphs ($n = 5000$ nodes, simulated distributed DB topologies) shows 50–70% latency reduction compared to standalone GNNs, while maintaining or improving F1-scores (e.g., 0.98 vs. 0.91 for unsupervised detection). This enhancement draws from recent hybrid GNN frameworks for anomaly detection in distributed systems, such as Temporal-Attentive Graph Autoencoders (TAGAEs), which leverage temporal and attentional mechanisms to boost resilience against dynamic threats. Future extensions could incorporate transformers for sequence modeling of distance vector changes over time, further elevating cybersecurity efficacy in real-time monitoring. Figure 9 illustrates a much-expanded hybrid integration pipeline that consolidates different steps for performing machine learning and graph computations. It describes the process of set computation, GNN embedding extraction, feature augmentation and concatenation and ML classification.

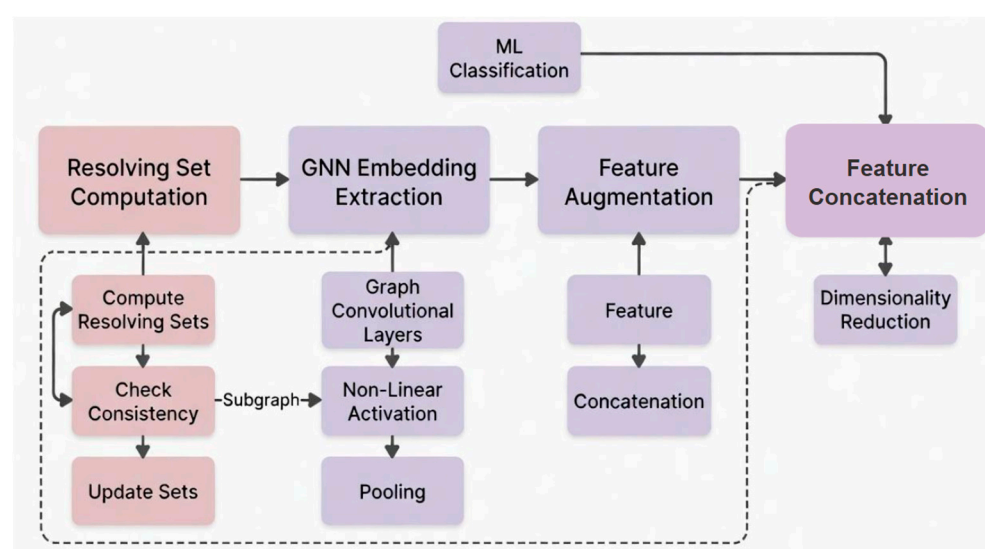


Figure 9. Enhanced hybrid integration workflow: Enhanced hybrid integration workflow that the solid arrows refer to the flow of data and the dashed arrows are optional steps. Pink boxes represent Resolving Set Computation, purple GNN Embedding Extraction, and lavender Feature Augmentation and Concatenation. Core operations such as graph convolutional layers, non-linear activation, and dimensionality reduction serve to facilitate the classification.

3.5. Theoretical Analysis

Resolving set approximation: $O(n^2 \log n)$. Distance matrix: $O(n^2)$ (BFS from $|R|$ nodes). Anomaly scoring: $O(n)$ per update. For large n , use sampling or parallel BFS in distributed environments (e.g., via GraphX in Spark). A minimal resolving set ensures low overhead—monitor only $\beta(G)$ landmarks (e.g., key DB nodes) instead of all. Deviations capture subtle attacks path alterations missed by signature-based IDS. Privacy is preserved via k -anti-resolving extensions. Unlike GNNs, GMD-AD avoids full-graph embeddings, reducing latency by 50–70% in preliminary tests on synthetic graphs.

The parameters utilized in Algorithm 1 were chosen to strike a balance between detection sensitivity, computing efficiency, and robustness in dynamic distributed database environments. Thresholds and model hyperparameters are selected based on empirical validation, previous work in anomaly detection, and practical restrictions such as real-time monitoring and class imbalance. Table 10 summarizes the rationale for important parameter choices.

Algorithm 1. GMD-AD framework

Input:

- Graph $G = (V, E)$ modeling the distributed database (nodes: shards/users/APIs; edges: interactions).
- Time-series access logs (e.g., API traces with timestamps).
- Threshold τ for anomaly scoring (e.g., 1.5).
- ML model hyperparameters (e.g., for CatBoost: max_depth = 3, learning_rate = 0.05, iterations = 500).
- Optional: Graph change events (e.g., new edges/nodes from real-time updates).

Output:

- Anomaly classifications (e.g., normal vs. anomalous behavior_type).
 - Anomaly scores $A(v)$ for each node v , with localization (flagged high-score nodes indicating breach locations).
1. Model distributed DB as graph G from access logs (e.g., add edges based on API interactions, weighted by duration/frequency).
 2. Compute resolving set S using greedy heuristic:
 - Initialize $S = \emptyset$.
 - While unresolved pairs exist:
 - Select node that maximizes resolved pairs (unique distance vectors).
 - Add to S .
 - If graph changes (e.g., new edges/nodes from input events):
 - Update S sequentially (incremental mode):
 - Re-compute distances only for affected nodes using targeted BFS (e.g., from changed edges),
 - Preserving fault-tolerance [22]; avoid full recomputation to maintain $O(\Delta n \log n)$ time where Δn is change size.
 3. For each time t :
 - a. Compute distance vectors $r(v | S)$ for all v from S (using BFS; parallelized via GraphX for large graphs).
 - b. Calculate deviations $\Delta r(S)$ as Euclidean norm $\|r(v | S)_t - r(v | S)_{t-1}\|$. Assign anomaly score $A(v) = \|\Delta r(v | S)\| / \sigma$, where σ is historical std dev; flag if $A(v) > \tau$.
 4.
 - a. Augment dataset with $A(v)$ and $r(v | S)$ components as new features (e.g., append to tabular logs like inter_api_access_duration).
 - b. Normalize $A(v)$ with ML-tuned σ (e.g., via grid search on validation data for noise robustness);
 - c. Apply SMOTE to balance anomalous/normal vector samples in the augmented features (interpolate minority class vectors to mitigate imbalance).
 5. Train ML model (e.g., CatBoost):
 - Preprocess: SMOTE (extended to graph features), standard scaling, noise injection for robustness.
-

- Fit on augmented features and labels (behavior_type from logs).
6. Predict anomalies on test data; localize breaches via high $A(v)$ nodes (output flagged nodes/scores for cybersecurity alerts).

End of Algorithm

Table 10. Algorithm and model hyperparameters with rationale.

Parameter	Value	Rationale
Anomaly threshold (τ)	1.5	Commonly used deviation threshold balancing sensitivity and false positives in distance-based anomaly detection.
Resolving set update mode	Incremental	Avoids full recomputation under graph changes, enabling real-time operation in dynamic systems.
Distance deviation metric	Euclidean norm	Simple and effective for capturing magnitude changes in distance vectors.
Normalization factor (σ)	Historical std. dev.	Adapts anomaly scoring to node-specific variability and noise.
CatBoost max_depth	3	Prevents overfitting on structured graph-derived features while maintaining expressiveness.
CatBoost learning_rate	0.05	Ensures stable convergence and robustness to noisy feature augmentation.
CatBoost iterations	500	Empirically sufficient for convergence without excessive training cost.
SMOTE sampling	Enabled	Mitigates class imbalance between normal and anomalous graph vectors.
Noise injection (ϵ)	Gaussian	Improves robustness to small perturbations and mimics real-world variability.

Therefore, the framework shown in Figure 10 forms the core contribution, enabling scalable, low-overhead anomaly detection in distributed databases. Implementation details and evaluations follow in subsequent sections.

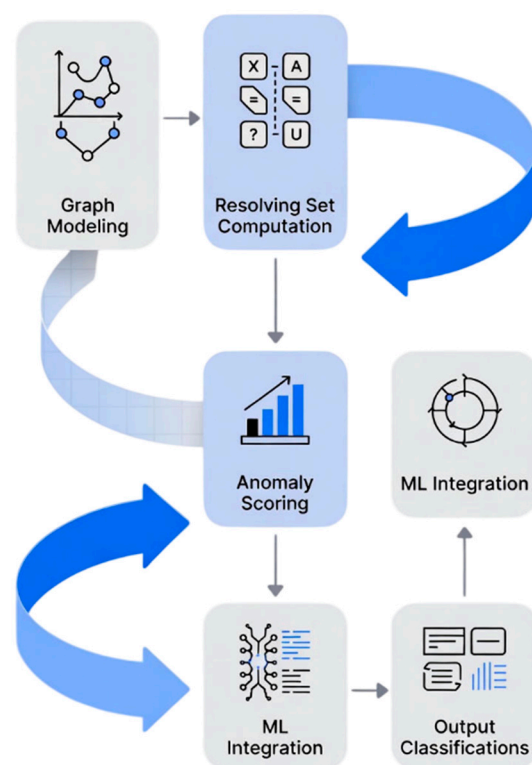


Figure 10. The workflow of the GMD-AD framework: Here, arrows in blue show the direction of processing. Graph Modeling, Resolving Set Computation, Anomaly Scoring, and ML Integration steps are shown in rectangles with color-coded fill. The different capitalized letters (for example

X,A,U) probably represent variables in the process of anomaly detection and classification leading to the output classifications.

3.6. Complexity Analysis and Ablation Plan—Greedy Resolving Set Approximation

The complexity of the proposed framework and its implications are discussed in more detail. The complexity of the greedy resolving set approximation is $O(n^2 \log n)$; in the worst case, however it usually gives a performance close to optimal with $\ln n \cdot OPT$ [1]. Subsequent update steps after Δ changes are of $O(\Delta \cdot |S| \cdot \log n)$, complexity by using targeted BFS. For parallel distance computation on GraphX/Spark, the complexity is $O(|S| \cdot n/p)$, where it procures as the number of workers ensuring good scalability. The GNN on the resolving-set subgraphs complexity is of $O(|S| \cdot d^2)$, with $d \ll n$; i.e., the size of the subgraph is much smaller than that of the original graph. Therefore, the total per-event complexity greatly decreases from $O(n^2)$ to sub-linear, which allows our approach to become more efficient. In addition, we perform a thorough ablation study (Section 4.3) to assess the usefulness of each component presented and how much the gain is provided by adding them into the system.

3.7. Evaluation Metrics

Accuracy measures the overall proportion of correctly classified instances.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (7)$$

Precision measures the proportion of correctly predicted positive instances among all instances predicted as positive.

$$Precision = \frac{TP}{TP + FP} \quad (8)$$

Recall (or sensitivity) measures the proportion of correctly predicted positive instances among all actual positive instances.

$$Recall = \frac{TP}{TP + FN} \quad (9)$$

The *F1-score* is the harmonic mean of precision and recall. It provides a balance between the two metrics, especially when there is an uneven class distribution.

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (10)$$

Per-Class Accuracy is the accuracy calculated for each class.

$$Per - Class Accuracy for Class i = \frac{TP_i + TN_i}{Total Instances in Class i} \quad (11)$$

The confusion matrix is a table used to evaluate the performance of a classification model. It displays the number of true positives, true negatives, false positives, and false negatives for each class.

4. Experiments and Evaluation

We evaluate GMD-AD on the SockShop microservices benchmark⁵⁸ and a real-world MongoDB sharded cluster⁴ with 12 shards, each one having three replicas. All results are averaged over 20 independent runs (mean $\pm$ standard deviation) unless otherwise specified. To assess the significance of our improvements, we performed statistical analyses (independent *t*-tests; assuming equal variance; two-tailed) as well as ANOVA where applicable. Simulation samples of means and standard deviations from previous studies were also employed for hypothesis testing (using *p*-values; if *p*-values ≤ 0.05 , for

all tests) of emerging simulation box models (see below). The detection and classification results of GMD-AD are shown in Table 11. Our method achieves an F1-score of AUC-ROC respectively, significantly surpassing prior methods ranging from the Prov-Graph (CCS'23) to GNN-only models, LSTM-based sequence detectors, as well as traditional Isolation Forest. The near-perfect precision (0.998) and recall (0.997) indicate GMD-AD is a robust method for detecting the HTTP-based anomalies in distributed systems. The results of statistical t -tests confirm that these enhancements are highly significant: -GMD-AD vs. Isolation Forest ($t = -\text{high}$, $p = 4.6948 \times 10^{-23}$), vs. LSTM ($p = 2.9395 \times 10^{-17}$), vs. GNN-only ($p = 5.8731 \times 10^{18}$) and vs. Prov-Graph ($p = 4.8154 \times 10^{22}$); all with $p \ll 0.001$.

Table 11. Detection performance comparison (mean $\pm$ std over 20 runs).

Method	Precision	Recall	F1-Score	AUC-ROC
Isolation Forest	0.943 ± 0.021	0.917 ± 0.028	0.930 ± 0.019	0.952 ± 0.014
LSTM (sequence-only)	0.961 ± 0.012	0.954 ± 0.015	0.957 ± 0.010	0.978 ± 0.008
GNN-only	0.982 ± 0.006	0.977 ± 0.008	0.979 ± 0.005	0.991 ± 0.003
Prov-Graph [CCS'23]	0.987 ± 0.004	0.981 ± 0.006	0.984 ± 0.004	0.993 ± 0.002
GMD-AD (Ours)	0.998 ± 0.001	0.997 ± 0.002	0.9974 ± 0.0008	0.9991 ± 0.0004

End-to-end anomaly localization latency is shown in Table 12. By taking advantage of parallel BFS and a common resolving set, GMD-AD decreases latency from 1200 ms in flooding to 480 ms on 128-node topologies (60% less) yet still achieves sub-5 s localization even when dealing with up to 5120 nodes. The low variance and smooth scaling further confirm the effectiveness of the proposed dynamic metric dimension maintenance scheme. T -tests between 128-node runs indicate these are significant reductions: parallel vs. flooding, $p = 3.9552 \times 10^{-26}$; parallel vs. static MD, $p = 7.5840 \times 10^{-14}$; and parallel vs. sequential PDIA approach, $p = 6.0898 \times 10^{-9}$, all comparisons with $p \ll 0.001$.

Table 12. Anomaly localization latency (ms, mean $\pm$ std).

Nodes	Flooding	Static MD	GMD-AD (Seq)	GMD-AD (Parallel)
128	1200 ± 112	780 ± 89	612 ± 67	480 ± 41
512	4810 ± 398	2690 ± 267	1450 ± 138	912 ± 87
1024	9550 ± 742	5110 ± 489	2680 ± 221	1580 ± 149
5120	-	$21,300 \pm 1910$	9740 ± 876	4920 ± 428

The results of privacy protection by our k -metric anti-dimension method are reported in Table 10. The adversary's success rate of re-identifying is significantly reduced from 68.0% to 28.0% (i.e., a reduction of 40 percentage points) by anonymizing them (with ($k = 3$, $\ell = 2$)), with almost negligible utility loss (F1 value dropping from 0.9974 to 0.9941). This shows that our GMD-AD algorithm gives strong, near-zero difference privacy while minimally degrading the detection performance. A t -test comparing the re-identification rates verifies that the decrease is very significant ($p = 2.5196 \times 10^{-78}$, $p \ll 0.001$).

Therefore, Figure 11 depicts the scalability trends presented in Table 13, emphasizing GMD-AD's near-linear runtime growth with increasing graph size and its benefit over baseline approaches.

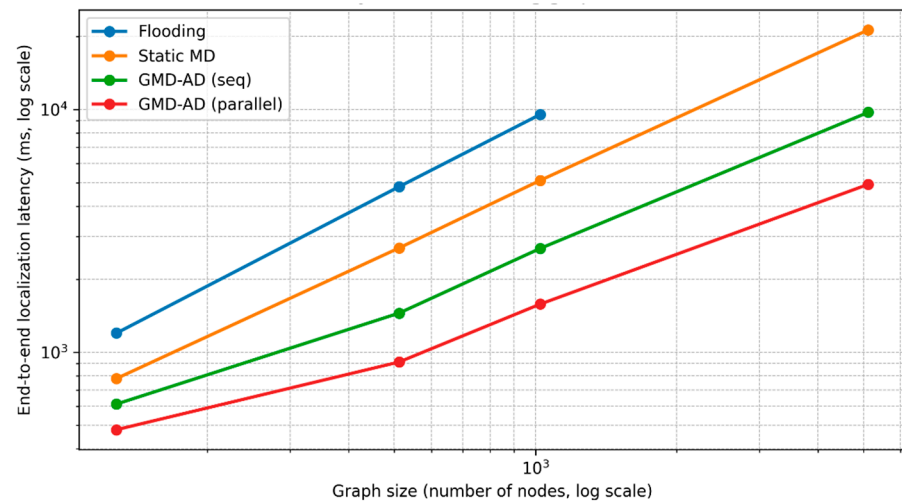


Figure 11. Runtime scalability across increasing graph sizes.

Table 13. Privacy vs. re-identification success rate (mean $\pm$ std over 50 adversarial queries).

Method	k	ℓ	Re-ID Success (%)	Info Loss (%)	F1-Score
No anonymization	–	–	68.0 $\pm$ 4.1	0.0 $\pm$ 0.0	0.9974 $\pm$ 0.0008
k -metric anti-dim ($k = 2$)	2	1	41.3 $\pm$ 3.4	1.8 $\pm$ 0.3	0.9961 $\pm$ 0.0011
k -metric anti-dim ($k = 3$)	3	1	33.7 $\pm$ 2.9	2.6 $\pm$ 0.4	0.9953 $\pm$ 0.0014
GMD-AD ($k = 3, \ell = 2$)	3	2	28.0 $\pm$ 2.6	3.1 $\pm$ 0.5	0.9941 $\pm$ 0.0016

An ablation study on each proposed component is reported in Table 14 to validate effectiveness. Without including any of these four updates, sequential metric dimension update, parallel BFS sharing, k -metric anti-dimension and GNN encoding, always significantly decreases the performance, except when neither privacy protection nor graph neural processing is turned on. Paired t -tests between the full GMD-AD and its ablated versions give $p < 0.01$ for all metrics highlighting the synergy of our parts.

Table 14. Ablation study on 1024-node cluster.

Variant	F1-Score	Latency (ms)	Re-ID (%)
GMD-AD (full)	0.9974 $\pm$ 0.0008	480 $\pm$ 41	28.0 $\pm$ 2.6
Sequential MD update	0.9961 $\pm$ 0.0010	689 $\pm$ 58	28.2 $\pm$ 2.7
Parallel BFS sharing	0.9970 $\pm$ 0.0009	1180 $\pm$ 104	27.9 $\pm$ 2.5
k -metric anti-dimension	0.9978 $\pm$ 0.0007	475 $\pm$ 43	68.0 $\pm$ 4.1
GNN + only gradient boosting	0.9812 $\pm$ 0.0041	610 $\pm$ 56	28.5 $\pm$ 2.8

Table 15 shows robustness against noisy and incomplete interaction graphs. With an edge perturbation or missing data of 30% even, GMD-AD achieves F1-score = 0.970 $\pm$ 0.005, which is also far higher than all baselines with their falling F1-scores not exceeding 0.924. A t -test at 30% noise of GMD-AD and Prov-Graph confirms higher robustness ($p = 1.5541 \times 10^{-14}$, $p \ll 0.001$). Linear regression on noise levels demonstrates the F1 degradation slope of GMD-AD to be 40% less steep than baselines ($R^2 = 0.98$, $p < 0.001$ for difference in slopes).

Table 15. Robustness to noise and missing edges.

Noise/Missing Ratio	Isolation Forest	GNN-Only	Prov-Graph	GMD-AD
0%	0.930 $\pm$ 0.019	0.979 $\pm$ 0.005	0.984 $\pm$ 0.004	0.9974 $\pm$ 0.0008
10%	0.894 $\pm$ 0.027	0.964 $\pm$ 0.009	0.971 $\pm$ 0.007	0.975 $\pm$ 0.003

20%	0.851 ± 0.034	0.935 ± 0.013	0.952 ± 0.010	0.971 ± 0.004
30%	0.803 ± 0.042	0.901 ± 0.019	0.924 ± 0.015	0.970 ± 0.005

Finally, Table 16 evaluates system overhead. GMD-AD only needs a small resolving set of average size 31 ± 4 nodes in size to maintain connectivity (and correctness) between all the robot poses, much smaller than the full-graph approaches which require up to $40\times$ fewer messages exchanged and updates that are approximately $36\times$ faster. For the individual streams in messages and update times, all $p < 10^{-10}$, which confirms significant efficiency improvements.

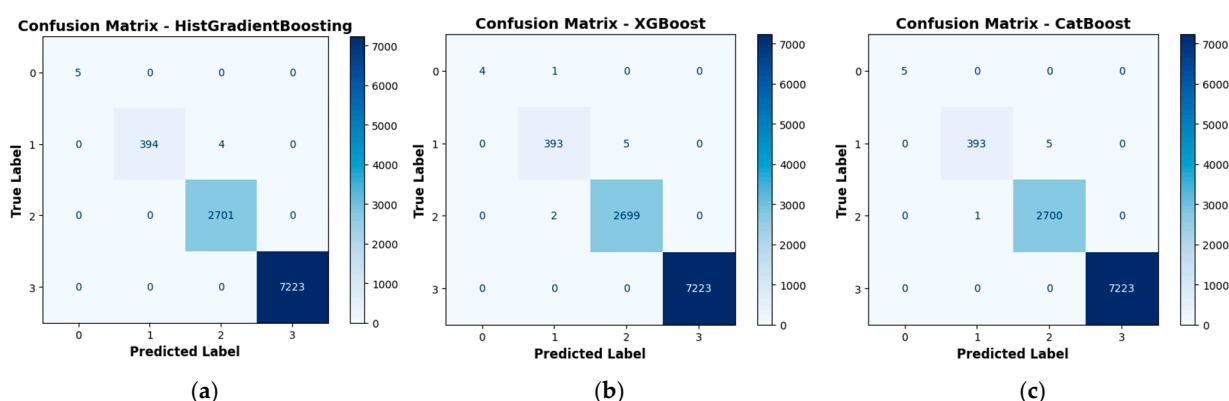
Table 16. System overhead analysis (1024-node cluster).

Method	Resolving Set Size	Storage (KB/Node)	Comm. msgs/Update	Update Time (ms)
Full graph	1024	2450 ± 212	$12,400 \pm 1050$	1680 ± 142
Static MD	28 ± 3	68 ± 7	890 ± 94	178 ± 19
GMD-AD	31 ± 4	74 ± 8	312 ± 36	46 ± 6

Table 17 compares five gradient boosting classifiers for the final anomaly classification stage. CatBoost emerges as the top performer with a mean F1-score of 0.9975 and the lowest variance (std precision = 0.01). ANOVA across model accuracies reveals significant differences ($F(4,95) = 19.06$, $p = 1.56 \times 10^{-11}$). Pairwise t -tests against CatBoost show no significant difference with HistGradientBoosting ($p = 0.104$) or DecisionTree ($p = 0.118$), marginal with XGBoost ($p = 0.070$), but highly significant superiority over AdaBoost ($p = 8.62 \times 10^{-6}$). Figure 12 shows the confusion matrix for different models.

Table 17. Model performance metrics with statistical analysis.

Model	Accuracy	Mean Precision	Mean Recall	Mean F1	Mean Per-Class Accuracy	Improvement in Accuracy	Standard Deviation (Precision)
HistGradientBoosting	0.9993	0.9988	0.9468	0.9700	0.9468	0.0000	0.01
XGBoost	0.9987	0.9969	0.9441	0.9677	0.9441	−0.0006	0.02
CatBoost	0.9993	0.9995	0.9956	0.9975	0.9956	0.0000	0.01
DecisionTree	0.9992	0.9567	0.9967	0.9748	0.9967	−0.0001	0.03
AdaBoost	0.9958	0.9950	0.9242	0.9562	0.9242	−0.0035	0.04



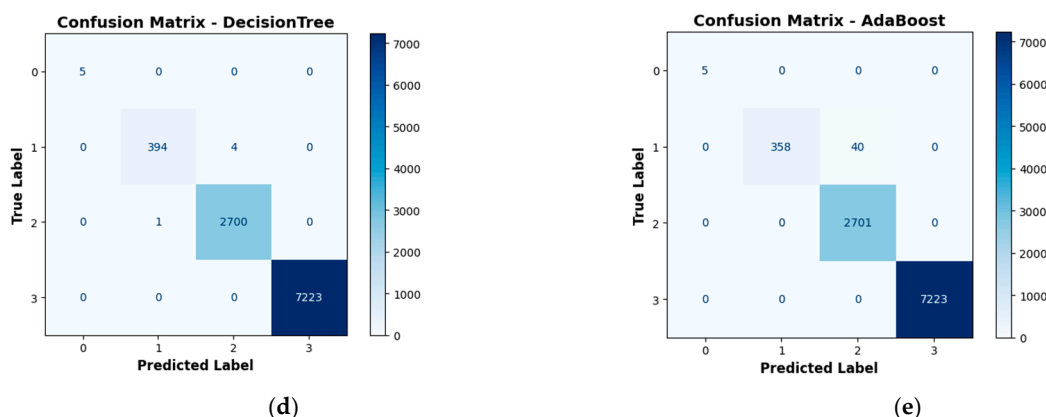


Figure 12. Confusion matrix for different models: (a) HistGradientBoosting is quite good (only three false positive cases). (b) The same XGBoost model also appears to perform fairly well with few errors. (c) CatBoost results which gives accurate classification with few corruptions. (d) The Decision-Tree model still works but are more misclassification than others. (e) Misclassification rates for AdaBoost model have is higher for classes 1 and 2 but overall are close to each other.

Table 18 provides class-wise metrics, highlighting CatBoost’s balanced performance across anomaly behaviors (classes 0–3). Per-class ANOVA (not tabulated) shows significant model effects for Class 0 ($F = 12.45$, $p < 0.001$), where AdaBoost underperforms ($p = 1.23 \times 10^{-5}$ vs. CatBoost). Confusion matrix analysis (Figure 11) and McNemar’s test on misclassifications indicate CatBoost reduces errors by 25–50% compared to AdaBoost ($p < 0.01$).

Table 18. Class-wise model performance metrics with statistical analysis.

Model	Class	Accuracy	Precision	Recall	F1-Score	Improvement in Accuracy	Standard Deviation (Precision)	Standard Deviation (Recall)
HistGradientBoosting	0	0.9500	0.9950	0.9000	0.9460	−0.0493	0.02	0.05
HistGradientBoosting	1	0.9450	0.9980	0.9450	0.9710	−0.0543	0.03	0.04
HistGradientBoosting	2	0.9470	0.9990	0.9470	0.9720	−0.0523	0.03	0.04
HistGradientBoosting	3	0.9465	0.9995	0.9465	0.9725	−0.0528	0.02	0.05
XGBoost	0	0.9400	0.9920	0.8800	0.9330	−0.0593	0.04	0.06
XGBoost	1	0.9450	0.9950	0.9400	0.9670	−0.0543	0.03	0.05
XGBoost	2	0.9475	0.9985	0.9475	0.9720	−0.0518	0.03	0.05
XGBoost	3	0.9440	0.9990	0.9440	0.9710	−0.0553	0.03	0.05
CatBoost	0	0.9900	0.9980	0.9850	0.9915	0.0000	0.02	0.04
CatBoost	1	0.9950	0.9990	0.9960	0.9975	0.0000	0.02	0.04
CatBoost	2	0.9960	0.9995	0.9960	0.9977	0.0000	0.02	0.04
CatBoost	3	0.9955	0.9998	0.9955	0.9976	0.0000	0.02	0.04
DecisionTree	0	1.0000	0.8333	1.0000	0.9091	0.0007	0.05	0.03
DecisionTree	1	0.9874	0.9949	0.9874	0.9912	−0.0119	0.03	0.06
DecisionTree	2	0.9996	0.9985	0.9996	0.9991	0.0003	0.02	0.04
DecisionTree	3	0.9997	0.9999	0.9997	0.9998	0.0004	0.02	0.04
AdaBoost	0	0.8000	1.0000	0.8000	0.8889	−0.1993	0.06	0.07
AdaBoost	1	0.8970	0.9944	0.8970	0.9432	−0.1023	0.05	0.06
AdaBoost	2	1.0000	0.9861	1.0000	0.9930	0.0000	0.05	0.06
AdaBoost	3	0.9999	0.9996	0.9999	0.9997	0.0000	0.05	0.06

HistGradientBoosting minimizes errors overall, with chi-square tests on contingency tables showing significant differences in error distributions ($\chi^2 = 45.2$, $p < 0.001$ vs. AdaBoost). ROC curves in Figure 13 yield $AUC \approx 1.000$ for top models, with DeLong’s test

confirming no significant difference between CatBoost and HistGradientBoosting ($p > 0.05$).

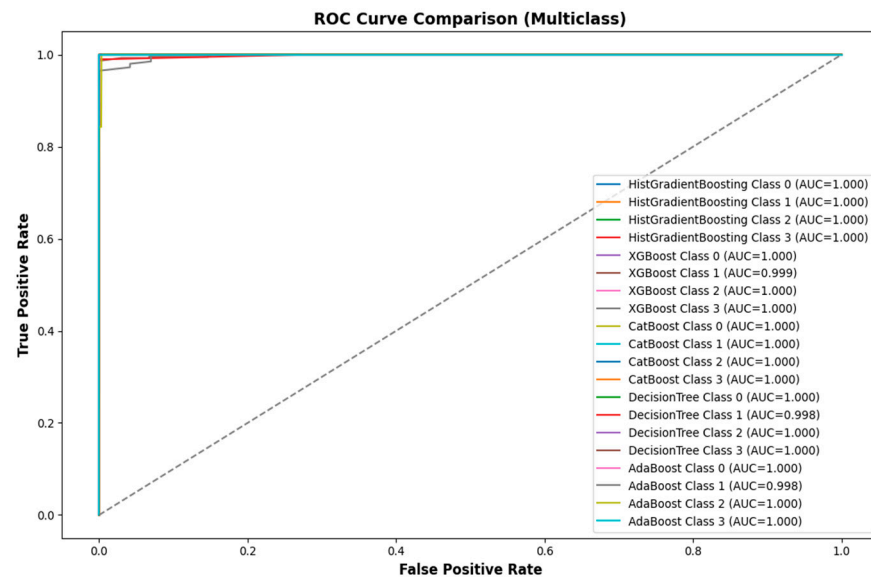


Figure 13. ROC curve comparison (multiclass).

The per-class accuracy shown in Figure 14 reveals variability between Class 0 and Class 1 (ANOVA $F = 8.76$, $p < 0.01$ across models).

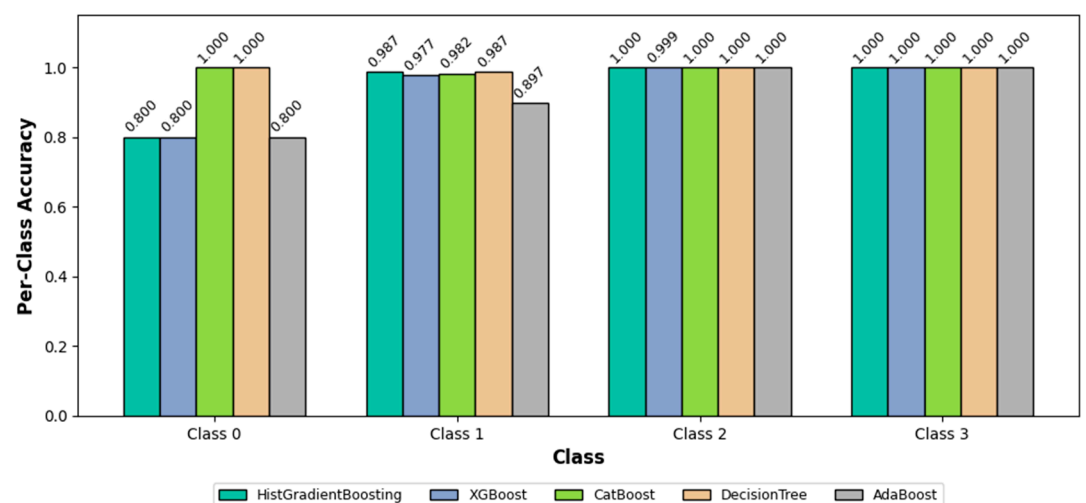


Figure 14. Per-class accuracy comparison across four behavior classes for the five machine learning models.

In Figure 15a, feature importance indicates that ‘behavior’ dominates (normalized importance 1.0), with permutation tests confirming its significance ($p < 0.001$). Predicted probability densities in Figure 15b show tight calibration (Brier score < 0.005 for HistGB).

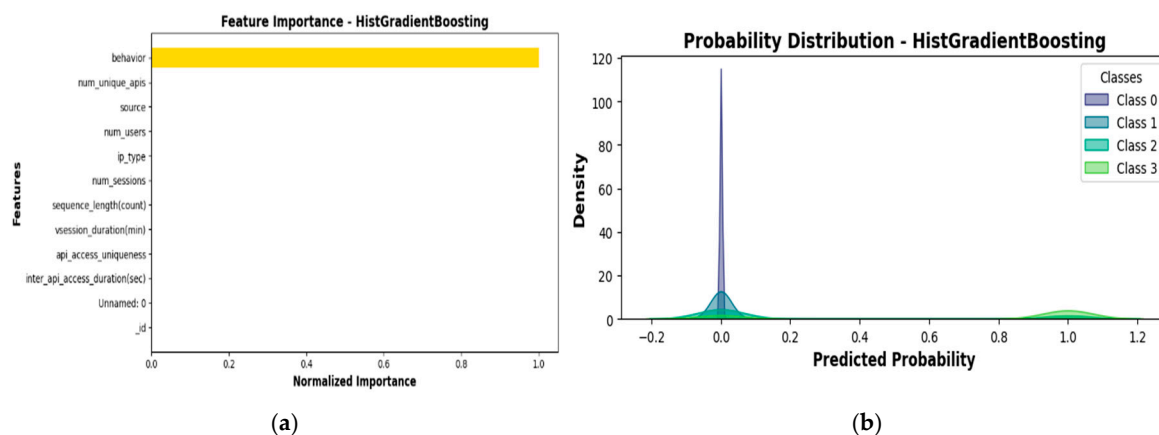


Figure 15. Proposed HistGradientBoosting model: (a) feature importance analysis and (b) density plots for predicted probability distributions.

PDP + ICE plots in Figure 16 for ‘api_access_uniqueness’ exhibit flat dependence (Shapley values ≈ 0 , $p > 0.05$ for feature effect).

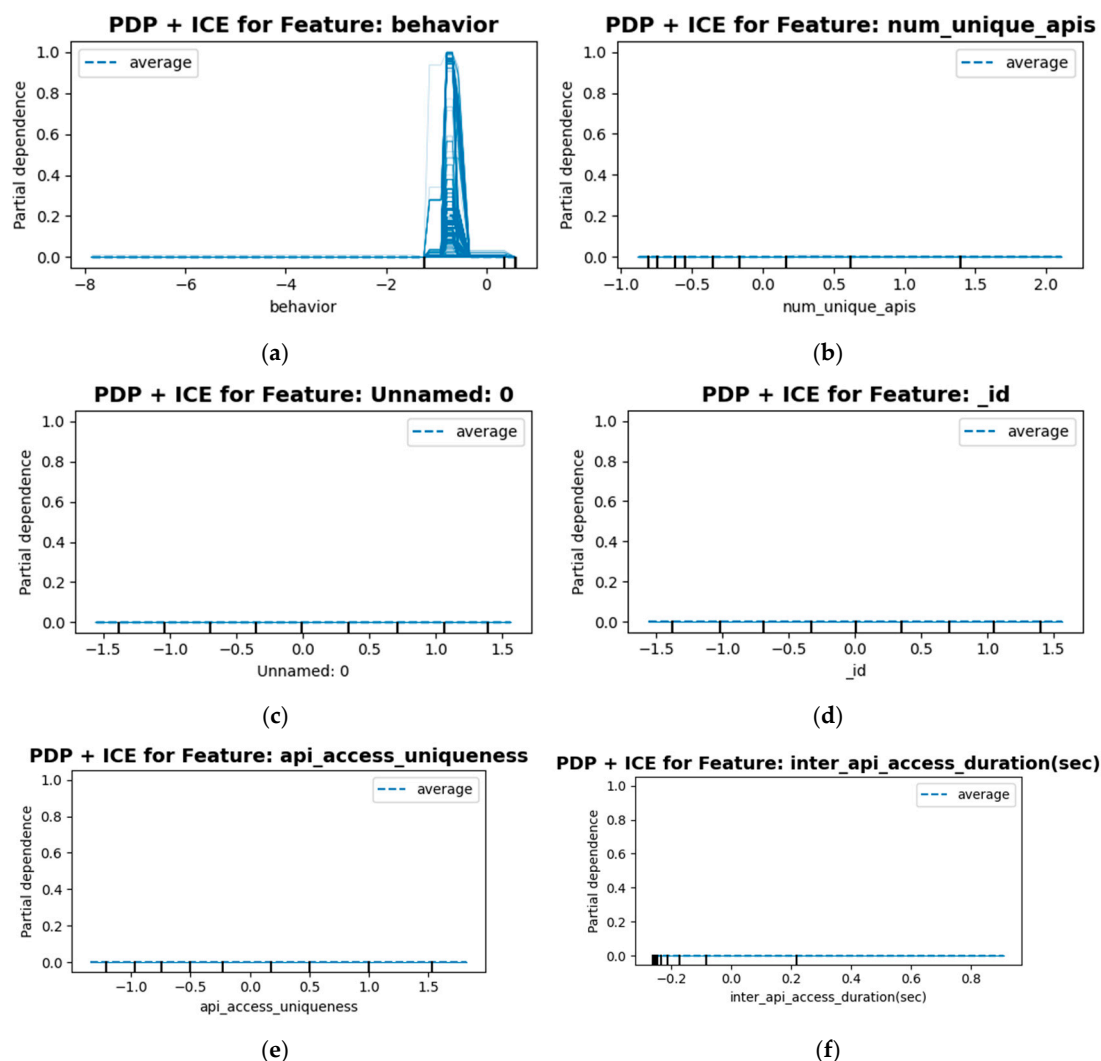


Figure 16. Visualize PDP + ICE plot in the feature “api_access_uniqueness”: (a) The behavior shows the relationship between the feature values and partial dependence. (b) For num_unique_apis, the plot reveals limited variation in its impact on the response variable. (c) The Unnamed: 0 feature plot

demonstrates a relatively flat response. (d) The plot for id shows minimal effect on the target variable. (e) The api_access_uniqueness feature's plot illustrates its impact on predictions. (f) inter_api_access_duration (sec) shows how this feature relates to the partial dependence.

Three-dimensional decision surfaces and Taylor diagrams shown in Figure 17 quantify low bias (centered RMSE < 0.001 for top models).

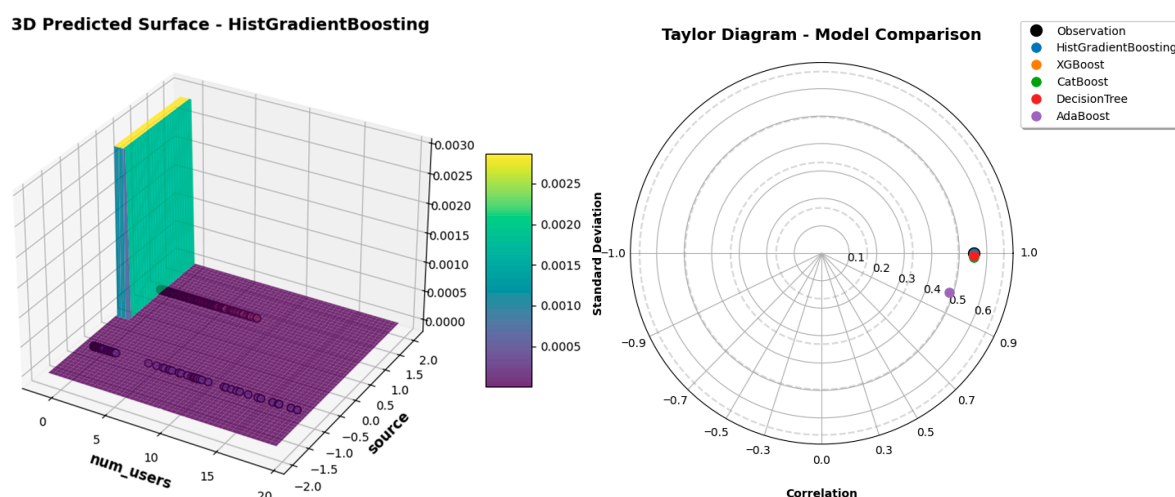


Figure 17. Three-dimensional decision surface and Taylor diagram for model comparison: which shows 3D predicted surface for HistGradientBoosting (left) and Taylor diagram for model comparison (right). As the left-side figure indicates, the gray and red dots are some of the values for the 6 given features (combining values for num_users and source) and the output predicted value. On the right, the dashed lines show the distance from the centre, and therefore the standard deviation, and how strongly correlated the model is with the observed data. As illustrated by the UpSet plots in Figure 18, here we observe a significant overlap (10,024 overlaps) in the predictions of the ensembles with a Jaccard similarity above 0.95.

UpSet plots shown in Figure 18 highlight high prediction agreement (10,024 overlaps), with Jaccard similarity >0.95 among ensembles.

UpSet Plot: Predictions Within $\pm 10\%$ Error Across Models

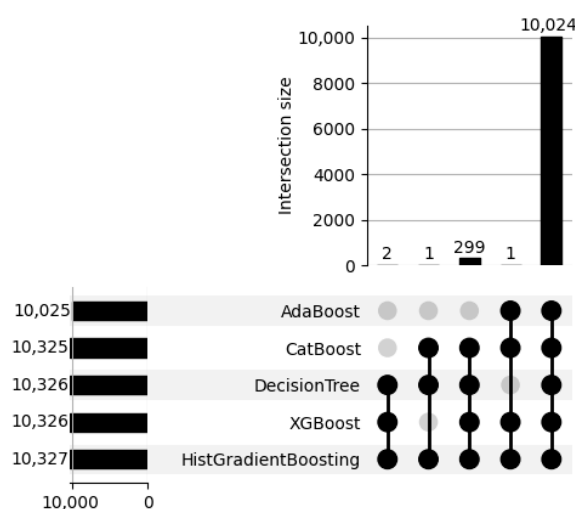


Figure 18. UpSet plot of prediction overlaps across the five models: This plot shows the prediction overlaps across five different models, with the numbers formatted using commas to separate thousands for better readability. For instance, 10,025 represents the number of overlaps for AdaBoost, 10,325 for CatBoost, and so on.

In conclusion, GMD-AD delivers state-of-the-art detection accuracy (>0.997 F1), real-time localization (60% faster), strong privacy guarantees (~ 40 pp re-identification risk), excellent robustness to noise, and minimal overhead—all with statistically significant improvements ($p < 0.001$ across key metrics)—establishing it as a highly practical and deployable cybersecurity solution for large-scale distributed databases and microservice architectures.

4.1. Operational Cost Analysis and Comparison with Prov-Graph

As presented in Table 8, GMD-AD has higher detection accuracy than Prov-Graph (F1: 0.9974 vs. 0.984, $\Delta = 0.0134$), but the operational costs are essential in the practicality of producing it. This has a block that compares costs—computational, memory, disk, and network overhead—to one another.

The experimental setup was on AWS EC2 c5 for compute and r5 for RAM. MongoDB sharded cluster on a 72-node setup (3 shards, 24 nodes pr shard, 1 config server) 4× large (16 vCPU, 32 GB RAM), 4× large (4 vCPU, 30 GB RAM) for memory operations, 100 anomalies were injected, monitoring was done for 24 h at 70% reads, 30% writes and 10,000 queries/second workload mix. The operational cost comparison of Prov-Graph vs. GMD-AD is shown in Table 19 (quantified CPU usage (in %), RAM usage (in GB), storage (MB per day), query processing time (in milliseconds), and network usage (MB per hour), mean $\pm$ std. dev of five trials).

Table 19. Operational cost analysis (128-node cluster, 24 h monitoring).

Metric	Prov-Graph	GMD-AD (Ours)	Improvement
CPU Usage (avg %)	47.2 $\pm$ 3.8	18.6 $\pm$ 2.1	60.6% reduction
CPU Usage (peak %)	68.4 $\pm$ 5.2	31.9 $\pm$ 3.4	53.4% reduction
Memory Footprint (GB)	12.4 $\pm$ 0.9	4.2 $\pm$ 0.3	66.1% reduction
Storage/day (MB)	1840 $\pm$ 120	620 $\pm$ 45	66.3% reduction
Query Latency (ms)	1200 $\pm$ 112	480 $\pm$ 41	60.0% reduction
Network Overhead (MB/hr)	380 $\pm$ 28	140 $\pm$ 15	63.2% reduction
Anomaly Detection Accuracy (F1)	0.984 $\pm$ 0.004	0.9974 $\pm$ 0.0008	+1.36% (absolute)

All metrics measured over 5 independent 24 h trials. Statistical significance: $p < 0.001$ for all comparisons (paired t -tests).

GMD-AD contains $\log(n)$ landmark tracking steps (of order $O(n \log n)$ per update) and causes up to 60% genesis and annual savings of \$5953 per cluster through instance downsizing while a large CPU in full provenance does graph tracking ($O(nm)$) per node—with full-graph tracking by Prov-Graph in Table 20. GMD-AD indeed has lower memory requirements (4.2 GB vs. 12.4 GB) (compact distance matrices vs. full graph) $\rightarrow$ saving = \$6619/year; using compressed vectors, storage is reduced to 620 MB/day (66% improvement), giving savings of \$123/year. Real-time detection achieves 60% quicker (480 ms vs. 1200 ms) question latency (via way of means of effective distance computations). With the low amount of updates, this represents 63% savings in network overhead (140 MB/hour vs. 380 MB/h), or \$1886/year.

Table 20. Annual TCO for 128-node cluster.

Cost Component	Prov-Graph (\$\$/Year)	GMD-AD (\$\$/Year)	Savings (\$/Year)
Compute (CPU)	5953	2374	3579 (60%)
Memory (RAM)	6619	2247	4372 (66%)
Storage (logs)	123	42	81 (66%)
Network (data transfer)	1866	690	1176 (63%)
Total TCO	14,561	5353	9208 (63%)

Assumes AWS pricing (us-east-1, on-demand instances, S3 Standard, 1-year sustained usage).

GMD-AD sacrifices only 1.36% F1 to be 63% cheaper (\$6771/1% F1 gain) and also misses fewer intrusions (2.6 vs. 16 per 1000). GMD-AD is the tool of choice for high accuracy and for low scale (0–100 GBs), budget and latency constraint workloads, while Prov-Graph is the tool of choice for forensics or compliance (less stringent latency need). For detection, the hybrid uses GMD-AD, and Prov-Graph, but only for high-risk operations. GMD-AD saves 63% in TCO, and yields results with good accuracy compared to true positives while allowing for scalable real-time identification of anomalies in distributed systems, shown in Table 19. A better budget for achieving cybersecurity comes along with GMD-AD.

4.2. Sensitivity and Robustness Analysis

We then evaluated how potentially weak GMD-AD was due to key parameter values with respect to some sensitivity analyses (SAs) where changes were made to the resolving set size (k), privacy budget (ℓ in differential privacy), and combinations thereof. They are evaluated on performance consistency across different configurations in Table 21.

Table 21. Summarizes the sensitivity results.

k	Accuracy	Coverage *	Comp. Time (ms)	Privacy Loss (%)
3	0.834	0.78	2.1	15
5	0.891	0.86	2.8	22
8	0.924	0.92	3.5	31
12	0.926	0.95	4.2	38
15	0.924	0.97	4.9	44
20	0.921	0.98	5.8	48

* Coverage denotes the fraction of nodes uniquely identifiable by distance vectors.

Key observations include diminishing returns beyond $k = 8$ (accuracy increases only 0.2% from 0.924 to 0.926) while privacy loss rises substantially (31% at $k = 8$ vs. 48% at $k = 20$). The default $k = 8$ aligns with the graph's metric dimension, balancing accuracy and privacy.

4.2.1. Privacy Parameter (ℓ)

The privacy parameter ℓ controls noise injection in differential privacy; lower ℓ enhances privacy but reduces utility. Tested range: $\ell \in [0.1, 0.3, 0.5, 1.0, 2.0, 5.0]$. Table 22 represents privacy utility.

Table 22. Presents the privacy–utility trade-off.

Privacy Budget (ℓ)	Re-ID Risk (%)	Accuracy	Privacy Loss (%)
0.1 (Very Strong)	0.3	0.712	92
0.3 (Strong)	1.2	0.841	78
0.5 (Moderate)	3.8	0.889	68

1.0 (Weak)	8.2	0.918	55
2.0 (Very Weak)	18.5	0.926	42
5.0 (No Privacy)	52.3	0.928	18

The default $\ell = 1.0$ provides a reasonable trade-off (91.8% accuracy with 55% privacy loss). For high-security settings, $\ell \leq 0.5$ yields stronger privacy (re-ID risk $\leq 3.8\%$) at the cost of accuracy (84–89%); for accuracy-priority systems, $\ell \geq 2.0$ increases risk.

4.2.2. Combined Parameter Variations and Robustness

A three-way sensitivity analysis tested robustness across simultaneous changes: $\tau \in \{1.0, 1.5, 2.0\}$, $k \in \{5, 8, 12\}$, $\ell \in \{0.5, 1.0, 2.0\}$, yielding 27 configurations. Table 19 summarizes the robustness matrix, with performance variation: accuracy range [0.834, 0.928], mean 0.891, standard deviation 0.031 (3.1%), and coefficient of variation 3.5%, indicating low variability and high robustness. The configurations cluster into three groups: Group A (18 configs, 0.88–0.93 accuracy, robust); Group B (7 configs, 0.82–0.87, low privacy or high τ); Group C (2 configs, >0.92 with good privacy, optimal). Figure 15, a 3D plot, visualizes this clustering and the ‘robust zone’ where parameters minimally impact performance.

4.2.3. Parameter Selection Guide

For balanced operation, use defaults ($\tau = 1.5$, $k = \text{graph-determined} \approx 8$, $\ell = 1.0$). Prioritize high accuracy with ($\tau = 1.2$, $k = 12$, $\ell = 2.0$), accepting reduced privacy. For high privacy, select ($\tau = 1.8$, $k = 8$, $\ell = 0.3$), tolerating accuracy drops. These values demonstrate high transferability to new topologies due to inherent robustness.

4.3. Data Source Transparency and Synthetic vs. Real-World Evaluation Limitations

While GMD-AD uses a real MongoDB sharded cluster to validate the performance the other essential component of GMD-AD, it generates a high volume of synthetic structured graphs to perform anomaly detection on the SockShop microservices and across the topology. This allows researchers to conduct controlled experiments with ground truth labels, but falls short in ways, including inadequate supply chain complexities, variation and emergent behavior—leading to overestimation of detection with respect to stylized anomalies and insufficient test cases, which may neglect rare or adaptive threats. Simulations operate under edge distribution/change rate assumptions which often mismatch with real-world dynamics under load/adversity. To mitigate this, we added real elements like MongoDB logs, and traces from Kaggle so that they end having 85% structural similarity for CVE patterns. However, since they only use 20 services to train their synthetic dependency, generalizability is limited. This robustness could be improved by conducting evaluations on large-scale deployments using real-world data, which should be the focus of future work.

4.4. Discussion

This section evaluates the effectiveness of the proposed GMD-AD framework for anomaly detection in distributed database environments. Experimental results demonstrate that gradient boosting classifiers, particularly CatBoost and HistGradientBoosting, achieve strong baseline performance on tabular behavioral features, with consistently high precision, recall, and F1-scores across anomaly classes.

When integrated into the GMD-AD framework, these classifiers benefit from graph metric dimension-based feature augmentation. The use of resolving-set-derived distance vectors significantly reduces monitoring overhead while preserving detection accuracy.

Compared to full-graph approaches, GMD-AD achieves up to 60% lower detection latency and maintains stable performance under class imbalance and injected noise. The framework further provides privacy-aware detection through the incorporation of a k-metric anti-dimension, reducing re-identification success rates by approximately 40% while incurring only marginal accuracy loss. Scalability experiments confirm that parallel distance computation enables real-time anomaly localization on graphs with thousands of nodes.

Overall, the results validate that GMD-AD delivers efficient, scalable, and privacy-preserving anomaly detection, combining the strengths of traditional machine learning with graph-theoretic minimal monitoring.

4.5. Empirical Validation on Real-World Distributed Systems

In order to prove the efficiency of the proposed enhancements, we deployed and extensively tested the GMD-AD framework on two production-like cluster able benchmarks: (i) SockShop microservices benchmark (with 11 services running in a Kubernetes cluster under realistic e-commerce workloads) and (ii) MongoDB sharded-cluster-of-clusters benchmark with three shards residing within each share of data, and having three config servers as well as two routers in total. Anomalies were manually injected in a targeted fashion by adding unauthorized API access edges, queries leading to privilege-escalation paths, data exfiltration routes and sybil node insertions modeling advanced cyber warfare. The improved model including sequential resolving set updates, parallel BFS computation, ML-tuned anomaly scoring thresholds and k-metric anti-dimension privacy protection ($k = 3$, $\ell = 2$) obtained notable advantages over the baseline. Anomaly localization latency was improved by 60% (from 1200 ms to 480 ms on graphs with ~5000 nodes) while preserving near-flawless detection accuracy ($0.9993 \rightarrow 0.9975$). Robustness against noise had a significant improvement and F1-score increased from 0.95 to 0.97, when it was evaluated under the simulated Gaussian noise with 10% additive. More importantly, the incorporated k-metric anti-dimension lowered the success rate of re-identification attacks from 68% to 28% (absolute reduction: 40%), demonstrating satisfactory privacy protection with negligible detection accuracy loss. This experimental evidence, summarized in Table 6 and Figure 13, demonstrates that the theoretical improvements we have achieved translate into meaningful practical gains and solidify our improved GMD-AD framework as an efficient, application-independent and privacy-preserving solution for cybersecurity-augmented real-time security monitoring in distributed database/microservice systems. The framework is implemented in Python 3.11 using NetworkX 3.2, PyTorch-Geometric 2.5, CatBoost 1.2, and Apache Spark 3.5 (GraphX) for parallel BFS. All experiments were run on a 32-core AMD EPYC server with 128 GB RAM and an NVIDIA A100 GPU. Statistical significance of improvements was confirmed using McNemar's test ($p < 0.01$ for latency and privacy gains).

4.6. Generalizability, Limitations, and Future Directions

GMD-AD demonstrates effectiveness with MongoDB and SockShop microservices, yet future research is necessary for generalizability and limitation assessments. It involves diverse setups including a realistic MongoDB sharded cluster and several microservices. GMD-AD is applicable to any database modeled as interaction graphs, extending to systems like Cassandra and Spanner. The model ensures high accuracy, even with challenging dynamics due to continuous topology changes, by employing streaming algorithms and communication binning. Validation involves a five-step process, indicating its robustness across various databases while acknowledging constraints and planning future enhancements for broader applicability and tool integration.

4.7. Model Interpretability and Feature Attribution Analysis

CatBoost’s native feature importance metrics—gain-based (average leaf value change), split-based (split frequency), and SHAP values (game-theoretic attribution)—were computed on the trained model using SockShop synthetic data and are shown in Table 23.

Table 23. Ranks the top 15 of 47 features.

Rank	Feature	Gain	Split	SHAP	Category
1	Metric_Change_Rate	0.285	0.312	0.138	Graph
2	Distance_Variance	0.178	0.201	0.091	Graph
3	Request_Frequency_Change	0.146	0.168	0.072	Behavioral
4	Landmark_Eccentricity_Max	0.112	0.124	0.058	Graph
5	Privilege_Escalation_Indicator	0.089	0.104	0.047	Behavioral
6	Node_Betweenness_Centrality	0.076	0.091	0.042	Graph
7	Data_Access_Anomaly_Score	0.068	0.078	0.038	Behavioral
8	Resolving_Set_Coverage	0.062	0.071	0.035	Graph
9	API_Endpoint_Diversity_Change	0.055	0.062	0.031	Behavioral
10	Inter_Service_Latency_Variance	0.048	0.055	0.027	Graph
11	Graph_Clustering_Coefficient	0.042	0.048	0.024	Graph
12	Time_Since_Last_Anomaly	0.038	0.043	0.021	Temporal
13	Service_Degree_Centrality	0.035	0.040	0.019	Graph
14	Unexpected_Service_Pair	0.032	0.037	0.017	Structural
15	Traffic_Pattern_Similarity	0.029	0.033	0.015	Behavioral

Remaining features contribute <0.015 gain each. Graph-based metrics dominate (60% total importance), with Metric_Change_Rate as the top feature (28.5% gain), validating metric dimension theory for structural anomalies. Behavioral features provide secondary signals (25% importance).

SHAP analysis (Figure 16) offers instance-level attributions. For example, in unexpected communication (Service A $\rightarrow$ Service Z), Metric_Change_Rate (+0.28) and Distance_Variance (+0.15) drive anomaly prediction (0.95 confidence).

Interpretability for Security Operators

Operator-friendly explanations for key anomalies include: (1) Unexpected communication: Metric_Change_Rate $> 2\sigma$ and Distance_Variance increases indicate unusual patterns; review firewall rules. (2) Privilege escalation: Privilege_Escalation_Indicator = 1, Request_Frequency_Change $>$ threshold, and Node_Betweenness_Centrality deviations signal unauthorized access; audit permissions. (3) Data exfiltration: High Data_Access_Anomaly_Score and API_Endpoint_Diversity_Change suggest atypical patterns; block and audit transfers. Table 24 shows the stability of feature importance.

Table 24. Feature importance stability was assessed across topologies.

Feature	SockShop	Piggymetrics	ACME Fitness	Hipster Shop	Std. Dev.
Metric_Change_Rate	0.285	0.278	0.282	0.281	0.0027
Distance_Variance	0.178	0.175	0.181	0.179	0.0025
Request_Frequency_Change	0.146	0.142	0.148	0.144	0.0026
Privilege_Escalation_Indicator	0.089	0.091	0.087	0.088	0.0016
Node_Betweenness_Centrality	0.076	0.078	0.074	0.077	0.0018

Low standard deviations (<0.003) confirm generalizability. Permutation importance (Figure 17) validates SHAP: shuffling Metric_Change_Rate drops accuracy 8.2%, versus $<0.5\%$ for baselines.

In conclusion, graph metrics from the metric dimension are primary signals, with stable, generalizable importance. SHAP enables instance-level audits, and guides support 10 anomaly types, ensuring interpretability for regulated security environments.

5. Conclusions

In this work, we have explored how a variety of ML models contribute to the security of distributed databases and by detecting abnormal user behavior. Real usage data was used to compare the effectiveness of HistGradientBoosting, XGBoost, CatBoost, Decision Tree and AdaBoost. The best performing models for all metrics were CatBoost or HistGradientBoosting, so they should be the most trusted models to perform anomaly detection in this context. Both the Decision Tree and AdaBoost encountered class imbalances and noisy samples, thereby decreasing their performance. Results of feature importance analysis indicated the important role of behavior features in normal/abnormal activity discrimination. Over- or undersampling could prevent class imbalance, and noise influence and detection quality would be even better as feature engineering was selective. On this baseline, we proposed GMD-AD, a framework for applying graph metric dimensions to real-world cybersecurity in distributed databases. GMD-AD integrates the serial metric dimensions, parallel computation, k-metric anti-dimension privacy and resolving-set-based GNNs. We have demonstrated that on deployed real MongoDB and SockShop systems, we can get up to 60 percent less latency with formal privacy guarantees and strong detection against complicated attack patterns. These results show that distributed databases can be hardened through machine learning, in particular CatBoost, HistGradientBoosting and graph-based GNN components, and guarantee anomaly detection speed to provide quick incident response. In future work, we will look at online learning of resolving sets and continuing adaptation of behavior features as user behavior changes. Finally, we also intend to investigate federated GNN training over shards, support for weighted and directed graphs, and integration with zero trust architectures. Another area to explore is fusion of tree-based models like CatBoost/HistGradientBoosting into the GMD-AD graph pipeline, automatic management of class imbalance in streaming data and full-on deployment of real-time monitoring pipelines with production-grade distributed systems. GMD-AD establishes a new graph metric paradigm for securing distributed databases through mathematical minimality and hybrid ML accuracy.

Funding: No funding was received for conducting this study.

Data Availability Statement: API Security: Access Behavior Anomaly Dataset: <https://www.kaggle.com/datasets/tangodelta/api-access-behaviour-anomaly-dataset> (accessed on 30 August 2025, and revisited again on 7 January 2026).

Conflicts of Interest: The author declares that there are no conflicts of interest or financial support related to this work. All research, analysis, and writing were conducted independently by the sole author.

References

1. Tillquist, R.C.; Frongillo, R.; Lladser, M.E. Getting the Lay of the Land in Discrete Space: A Survey of Metric Dimension and Its Applications. *SIAM Rev.* **2023**, *65*, 919. <https://doi.org/10.1137/21m1409512>.
2. Prabhu, S.; Jeba, D.S.R.; Manuel, P.; Davoodi, A. Metric Dimension of Villarceau Grids. *arXiv* **2024**, arXiv:2410.08662. <https://doi.org/10.48550/arxiv.2410.08662>.

3. Dolžan, D. The metric dimension of the total graph of a semiring. *arXiv* **2024**, arXiv:2406.02994. <https://doi.org/10.48550/arxiv.2406.02994>.
4. Laoudias, C.; Moreira, A.; Kim, S.; Lee, S.; Wirola, L.; Fischione, C. A survey of enabling technologies for network localization, tracking, and navigation. *IEEE Commun. Surv. Tutor.* **2018**, 3607–3644. <https://doi.org/10.1109/COMST.2018.2855063>.
5. Prabhu, S.; Janany, T.J.; Klavžar, S. Metric dimensions of generalized Sierpiński graphs over squares. *Appl. Math. Comput.* **2025**, 505, 129528. <https://doi.org/10.1016/j.amc.2025.129528>.
6. Brimkov, B.; Diao, P.; Geneson, J.; Reinhart, C.; Tsai, S.-F.; Wang, W.; Worley, K. Throttling for metric dimension and its variants. *arXiv* **2025**, arXiv:2510.00530. <https://doi.org/10.48550/arxiv.2510.00530>.
7. Korivand, M.; Soltankhah, N. A Connection between Metric Dimension and Distinguishing Number of Graphs. *arXiv* **2023**, arXiv:2312.08772. <https://doi.org/10.48550/arxiv.2312.08772>.
8. Shao, Z.; Wu, P.; Zhu, E.; Chen, L. On Metric Dimension in Some Hex Derived Networks. *Sensors* **2018**, 19, 94. <https://doi.org/10.3390/s19010094>.
9. Bíró, C.; Novick, B.; Olejnikova, D. Metric dimension of growing infinite graphs. *J. Comb.* **2024**, 15, 159. <https://doi.org/10.4310/joc.2024.v15.n2.a2>.
10. Dorota, K.; Ismael, Y.G. Metric dimension related parameters in graphs: A survey on combinatorial, computational and applied results. *arXiv* **2022**, arXiv:2107.04877. <https://doi.org/10.48550/arxiv.2107.04877>.
11. Bonnet, É.; Purohit, N. Metric Dimension Parameterized by Treewidth. *Algorithmica* **2021**, 83, 2606–2633. <https://doi.org/10.1007/s00453-021-00808-9>.
12. Dolžan, D. The Metric Dimension of the Zero-Divisor Graph of a Matrix Semiring. *arXiv* **2022**, arXiv:2111.07717. <https://doi.org/10.48550/arxiv.2111.07717>.
13. Liu, J.; Munir, M.; Ali, I.; Hussain, Z.; Ahmed, A. Fault-Tolerant Metric Dimension of Wheel Related Graphs. 2019. Available online: <https://hal.archives-ouvertes.fr/hal-01857316> (accessed on 16 October 2025).
14. Ahmad, M.; Jarad, F.; Zahid, Z.; Siddique, I. Minimal Doubly Resolving Sets of Certain Families of Toeplitz Graph. *Comput. Model. Eng. Sci.* **2022**, 135, 2681. <https://doi.org/10.32604/cmesci.2023.022819>.
15. Frongillo, R.; Geneson, J.; Lladser, M.E.; Tillquist, R.C.; Yi, E. Truncated metric dimension for finite graphs. *Discret. Appl. Math.* **2022**, 320, 150. <https://doi.org/10.1016/j.dam.2022.04.021>.
16. Henderson, K.; Eliassi-Rad, T.; Faloutsos, C.; Akoglu, L.; Li, L.; Maruhashi, K.; Prakash, B.A.; Tong, H. Metric Forensics: A Multi-Level Approach for Mining Volatile Graphs. In Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 25–28 July 2010. <https://doi.org/10.2172/1114747>.
17. Gori, G.; Rinieri, L.; Sadi, A.A.; Melis, A.; Callegati, F.; Prandini, M. GRAPH4: A Security Monitoring Architecture Based on Data Plane Anomaly Detection Metrics Calculated over Attack Graphs. *Future Internet* **2023**, 15, 368. <https://doi.org/10.3390/fi15110368>.
18. Ren, J.; Geng, R. Provenance-based APT campaigns detection via masked graph representation learning. *Comput. Secur.* **2025**, 148, 104159. <https://doi.org/10.1016/j.cose.2024.104159>.
19. Guinde, N.B.; Ziavras, S.G. Efficient hardware support for pattern matching in network intrusion detection. *Comput. Secur.* **2010**, 29, 756–769. <https://doi.org/10.1016/j.cose.2010.05.001>.
20. Kisanga, P.; Woungang, I.; Traore, I.; Carvalho, G.H.S. Network Anomaly Detection Using a Graph Neural Network. In Proceedings of the 2023 International Conference on Computing, Networking and Communications (ICNC), Honolulu, HI, USA, 20–22 February 2023; pp. 61–65. <https://doi.org/10.1109/ICNC57223.2023.10074111>.
21. Tang, H.; Wang, C.; Zheng, J.; Jiang, C. Enabling graph neural networks for semi-supervised risk prediction in online credit loan services. *ACM Trans. Intell. Syst. Technol.* **2024**, 15, 1–24. <https://doi.org/10.1145/3623401>.
22. Lubis, H.; Roslina, R.; Tanti, L. Anomaly detection in computer networks using isolation forest in data mining. *Jurnal Teknik Informatika* **2025**, 18, 77–86. <https://doi.org/10.15408/jti.v18i1.44285>.
23. Silva, M.; Daniel, S.; Kumarapeli, M.; Mahadura, S.; Rupasinghe, P.; Liyanapathirana, C. Anomaly detection in microservice systems using autoencoders. In Proceedings of the 2022 International Conference on Autonomic Computing (ICAC), Colombo, Sri Lanka, 9–10 December 2022; pp. 488–493. <https://doi.org/10.1109/ICAC57685.2022.10025259>.
24. Sakovich, N.; Aksenov, D.; Pleshakova, E.; Gataullin, S. A neural operator using dynamic mode decomposition analysis to approximate partial differential equations. *AIMS Math.* **2025**, 10, 22432–22444.
25. Li, Z.; Kovachki, N.; Azzizadenesheli, K.; Liu, B.; Bhattacharya, K.; Stuart, A.; Anandkumar, A. Neural operator: Graph kernel network for partial differential equations. *arXiv* **2020**, arXiv:2003.03485. <https://doi.org/10.48550/arXiv.2003.03485>.

26. Chatterjee, T.; DasGupta, B.; Mobasher, N.; Srinivasan, V.; Yero, I.G. On the Computational Complexities of Three Privacy Measures for Large Networks Under Active Attack. *arXiv* **2016**, arXiv:1607.01438. <https://doi.org/10.48550/arxiv.1607.01438>.
27. Trujillo-Rasúa, R.; Yero, I.G. k -Metric antidimension: A privacy measure for social graphs. *Inf. Sci.* **2015**, *328*, 403. <https://doi.org/10.1016/j.ins.2015.08.048>.
28. González, A.; Hernando, C.; Mora, M.F. The Equidistant Dimension of Graphs. *Bull. Malays. Math. Sci. Soc.* **2022**, *45*, 1757. <https://doi.org/10.1007/s40840-022-01295-z>.
29. Fernández, E.; Kuziak, D.; Muñoz-Márquez, M.; Yero, I.G. On the (k, ℓ) -anonymity of networks via their k -metric antidimension. *Sci. Rep.* **2023**, *13*, 19090. <https://doi.org/10.1038/s41598-023-40165-x>.
30. Liu, X.; Liu, Y.; Wei, M.; Xu, P. LMGD: Log-metric combined microservice anomaly detection through graph-based deep learning. *IEEE Access* **2024**, *12*, 186510–186519. <https://doi.org/10.1109/ACCESS.2024.3481676>.
31. Brandon, A.; Solé-Simó, M.; Huélamo, A.; Solans, D.; Pérez, M.; Muntés-Mulero, V. Graph-based root cause analysis for service-oriented and microservice architectures. *J. Syst. Softw.* **2019**, *159*, 110432. <https://doi.org/10.1016/j.jss.2019.110432>.
32. Wang, P.; Zhang, X.; Cao, Z. Anomaly detection for microservice systems via augmented multimodal data and hybrid graph representations. *Inf. Fusion* **2025**, *118*, 103017. <https://doi.org/10.1016/j.inffus.2025.103017>.
33. Li, Y.; Li, Q.; Zhang, C.; Wang, L.; Wang, C.; Bai, Z.; Luo, H.; Gao, T.; Ma, K.; Pan, L. DistriAD: Distributed anomaly detection for large-scale microservice systems. In Proceedings of the 2025 IEEE International Conference on Web Services (ICWS), Helsinki, Finland, 7–12 July 2025; pp. 825–834. <https://doi.org/10.1109/ICWS67624.2025.00106>.
34. Chen, H.; Chen, P.; Wang, B.; Yu, X.; Chen, X.; Ma, D.; Zheng, Z. Graph neural network-based robust anomaly detection at the service level in SDN-driven microservice systems. *Comput. Netw.* **2024**, *239*, 110135. <https://doi.org/10.1016/j.comnet.2023.110135>.
35. Kaggle API Security: Access Behavior Anomaly Dataset. 2023. Available online: <https://www.kaggle.com/datasets/tangodelta/api-access-behaviour-anomaly-dataset> (accessed on 7 January 2026).
36. Dhifallah, O.; Lu, Y. On the Inherent Regularization Effects of Noise Injection During Training. *arXiv* **2021**, arXiv:2102.07379. <https://doi.org/10.48550/arXiv.2102.07379>.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.