ISSN 2320 - 2602
Volume 14 No. 9, September 2025
International Journal of Advances in Computer Science and Technology
Available Online at http://www.warse.org/IJACST/static/pdf/file/ijacst021492025.pdf
https://doi.org/10.30534/ijacst/2025/021492025

Machine Learning-Based Methodology and System Architecture for Anomaly Detection in Software Products

Milena Lazarova[1], Adelina Aleksieva-Petrova[2], Antoniya Tasheva[3]
[1]Technical University of Sofia, Bulgaria, milaz@tu-sofia.bg
[2]Technical University of Sofia, Bulgaria, aaleksieva@tu-sofia.bg
[3]Technical University of Sofia, Bulgaria, atasheva@tu-sofia.bg

Received Date : July 28, 2025 Accepted Date : August 29, 2025 Published Date : September 07, 2025

ABSTRACT

Anomaly detection has become a critical area of research in fields such as cybersecurity, system monitoring, and software engineering. Modern approaches utilize machine learning techniques to identify anomalies by learning patterns of expected behavior. This paper presents a technical characterization of anomalies in software systems and proposes a data-driven approach to anomaly detection. By using machine learning techniques, the proposed methodology aims to improve the adaptability, scalability, and accuracy of detection mechanisms in modern software products.

Key words : anomaly detection, machine learning, software products, system architecture.

1. INTRODUCTION

The reliability and stability of software products are paramount in modern digital ecosystems. As software systems grow in complexity, ensuring their reliability during runtime becomes increasingly challenging. Modern software applications, whether desktop, web, or cloud-based, operate in dynamic environments where unexpected behaviors referred to as anomalies can arise from bugs, misconfigurations, resource leaks, or unanticipated usage patterns. The increasing complexity of software products, combined with dynamic user demands and diverse deployment environments, has made the identification of anomalies an essential aspect of software maintenance and quality assurance. As applications scale in complexity and interconnectivity, unforeseen anomalies such as performance degradation, functional errors, and unusual usage patterns become increasingly common. Anomalies, which manifest as deviations from expected software behavior, can lead to performance bottlenecks, degraded user experience, or even system failures if left undetected. Early detection and accurate analysis of these anomalies are crucial to ensuring software quality, maintaining performance, minimizing downtime, improving and maintaining user trust and experience.

Traditional rule-based monitoring approaches often rely on static thresholds or manual rule definitions which lack adaptability and generate high false-positive rates. Although useful for well-known issues the traditional anomaly detection methods often fail to capture nuanced and evolving anomalies related to the complexity of modern software behavior. This necessitates the development of advanced anomaly detection methods that leverage machine learning (ML) techniques utilized to automatically learn patterns of normal software behavior and accurately detect anomalies in real-time, with minimal human intervention.

The paper focuses on the development of a technical characterization of anomalies in software systems and presents a data-driven approach to anomaly detection in software products leveraging ML to achieve more adaptive, scalable, and accurate detection systems.

2. RELATED WORKS

Anomaly detection has been widely studied in various disciplines, including cybersecurity, system monitoring, and software engineering. Classical approaches such as threshold-based alerts and log file scanning are common but limited in terms of scalability and adaptability. More recent approaches include machine learning models to automatically identify deviations from expected behavior. Techniques range from simple clustering to complex deep learning architectures such as autoencoders. For example, MidLog is presented as an automated method for anomaly detection in log files based on a multi-head GRU architecture inspired by the multi-head Transformer mechanism [1]. Each GRU learns local patterns of sequence in the system log files, and their combined analysis allows for accurate anomaly detection. MidLog offers greater flexibility and scalability, achieving better accuracy

The research reported is funded by the project with contract №: KP-06-N57/4 from 16.11.2021, funded by the Bulgarian National Science Fund.

than baseline methods in experiments with public log file datasets.

Some researchers address the growing need for advanced cybersecurity tools due to emerging technologies like IoT and pervasive computing, which introduce new intrusion threats [2]. To support cybersecurity efforts, the authors propose a novel data preprocessing model based on a distributed computing architecture designed for handling large-scale datasets such as UGR'16. The study also explores the use of machine learning techniques to enhance the efficiency and responsiveness of intrusion detection systems. Experimental results show that decision tree algorithms perform better than multilayer perceptron neural networks when applied to large datasets in this context. However, most existing developments focus primarily on network-level anomalies or system-level resource monitoring, with relatively less emphasis on application-level anomalies in software products. Furthermore, the challenge remains in building interpretable, real-time, low-latency detection mechanisms suitable for deployment in software ecosystems.

An example of network-level anomalies is the study [3], which addresses the need for simple and effective methods for anomaly detection in large-scale network environments, where traditional approaches often lack depth and fail to utilize techniques such as unsupervised neural networks. As an alternative, the study uses self-organizing maps and extends the analysis beyond network traffic, including data from various information systems and using in-memory databases for faster processing. Data from application log files was analyzed and the anomaly detection method achieved a 96% success rate in validation tests. The proposed approach reduces the need for manual pre-qualification, reduces the burden on IT and security monitoring, and helps prevent potential attacks and security issues in advance.

Several anomaly detection techniques have been proposed in software engineering literature, ranging from static code analysis to dynamic monitoring techniques. Rule-based and signature-based methods rely heavily on domain expertise and predefined thresholds, limiting their ability to generalize to new or evolving anomalies. For example, the study [4] introduces a rule-based method using First-Order Logic to detect anomalies in software product lines, with a focus on false-optional features and incorrect cardinality. While issues such as dead features and redundancy have been widely studied, false-optional features and incorrect cardinality have received less attention, despite their significant impact on software configuration and validity. The authors propose a new classification for wrong cardinality and define all known cases of these anomalies in the domain engineering process. Experimental results confirm the scalability and effectiveness of the proposed approach.

Early approaches to anomaly detection in software focused on log scanning, rule-based alerts, and static performance thresholds. While simple to implement, these methods often fail in complex applications where behavior fluctuates based on user interactions, deployment environments, or business logic. Recent research in the software engineering domain highlights the effectiveness of machine learning techniques especially unsupervised and semi-supervised learning for automatically detecting anomalous application behavior. Models such as autoencoders, clustering algorithms, and tree-based methods have shown promise, particularly for detecting performance degradation, software regressions, and usage anomalies. Nevertheless, challenges remain in applying these models efficiently at runtime with low latency and high accuracy.

More recent advancements focus on machine learning models, including clustering algorithms, neural networks, and probabilistic models, to identify patterns indicative of anomalies. For example, some researchers introduce GRAND, a neural network model that combines a variational autoencoder and a generative adversarial network to identify anomalies in execution traces. [5] They focus on detecting software runtime anomalies using internal execution traces rather than external performance metrics, allowing the detection of both performance and functional errors. The method was tested on data from the Cassandra database system, comprising over 5,000-time series with millions of data points.A deep learning-based method for software anomaly detection is presented that uses convolutional neural networks to extract runtime data features and predict anomalies [6]. Experimental results demonstrate that the approach outperforms traditional methods in terms of accuracy, although it has limitations, including prolonged testing times.

Other study [7] presents a framework for automated anomaly detection and application change analysis, which integrates a regression-based transaction model and an application performance signature to detect significant changes in application behavior. It offers a non-intrusive and efficient solution using readily available monitoring data, making it suitable for enterprise environments.

The Coniferest package [8] is an open-source Python tool for anomaly detection that aligns closely with modern machine learning practices. It supports multiple algorithms, including Isolation Forest, Active Anomaly Discovery, and Pineforest. It utilizes Cython for performance-critical operations, enabling fast and scalable parallel processing, and features a user-friendly interface. Coniferest supports model serialization in ONNX format, making it easy to integrate into automated machine learning pipelines. However, application-level anomaly detection, especially within production software environments, remains an open challenge

due to data scarcity, the dynamic nature of applications, and the need for real-time analysis.

The DevOps toolchain generates large amounts of data that are often overlooked; yet, analyzing this data can provide valuable insights into a project's status and evolution [9]. Metrics such as the number of lines of code added since the last release or the number of failures detected in the staging environment can help predict potential risks in upcoming releases. To prevent issues in production, an anomaly detection system can analyze the staging environment by comparing current and previous releases using predefined metrics, with human operators handling false positives and negatives. This paper presents a proof-of-concept implementation that demonstrates the feasibility of this approach for selected functionalities.

Some researchers argue that equivalent mutants, traditionally viewed as a drawback in mutation analysis, can actually aid in detecting static anomalies in software artifacts [10]. The authors propose a technique that combines mutation, equivalence checking, and quality evaluation to identify deficiencies such as lack of clarity, unnecessary elements, or redundancy. They demonstrate that this approach applies to various artifacts, such as source code, Boolean expressions, feature models, and dependency graphs, where anomalies often reflect issues like non-minimality or poor readability. Although the method is not the fastest due to the cost of equivalence checking, experiments show it can detect anomalies as effectively as other techniques.

3. TECHNICAL CHARACTERIZATION OF ANOMALIES IN SOFTWARE SYSTEMS

An anomaly in software systems refers to any deviation from normal behavior that may indicate a defect, inefficiency, or abnormal state. We define anomalies along the following dimensions:

- *Performance Anomalies*: Unexpected spikes or drops in response time, CPU usage, memory consumption, or I/O throughput.
- *Behavioral Anomalies*: Deviations in user interactions, API call sequences, or internal application events.
- *Functional Anomalies*: Execution of unintended code paths, logical inconsistencies, or unusual error rates.

For machine learning, these anomalies are detected through structured features such as application-level metrics (response times, error rates), event logs (log message frequencies, log patterns), and user behavior sequences (clickstreams, session flows).

To effectively capture these anomalies, a structured feature taxonomy is proposed including resource usage metrics, application-level indicators, user interaction data, and system log patterns (Table 1).

The taxonomy guides the selection of relevant data features for model training and anomaly detection. The technical characterization guides the design of anomaly detection systems by defining what constitutes "normal" and the metrics to monitor deviations.

Table 1: Feature Taxonomy for Anomaly Detection

Feature Type	Example Metrics
Resource Usage	CPU, Memory, Disk I/O, Network I/O
Application Metrics	Response Time, Error Rate, API Latency
Event Sequences	Function Call Traces, API Call Graphs
User Interaction	Clickstreams, Session Durations, Page Flows
System Logs	Log Frequency, Log Pattern Deviations

4. MACHINE LEARNING-BASED ANOMALY DETECTION METHODOLOG

The proposed methodology consists of four stages: data collection, preprocessing, anomaly modeling, and evaluation. The suggested high-level system architecture for data-driven anomaly detection pipeline in software products is shown in Figure 1. The system comprises seven main layers: (1) data collection layer; (2) data preprocessing & feature extraction layer; (3) machine learning model layer; (4) model explainability (5) anomaly scoring & decision logic; (6) monitoring, alerting, & visualization layer; (7) feedback and model retraining.

4.1 Data Collection Layer

The Data Collection Layer serves as the foundational component of the anomaly detection pipeline, responsible for continuously capturing, aggregating, and forwarding operational data from software applications. Its primary goal is to ensure comprehensive and timely visibility into the behavior of software systems, enabling downstream analysis and anomaly detection processes to function effectively. This layer operates in both real-time and batch modes, depending on the operational requirements. In real-time mode, data is streamed with minimal delay to facilitate immediate anomaly detection, while in batch mode, historical data is periodically ingested for retrospective analysis or model training.

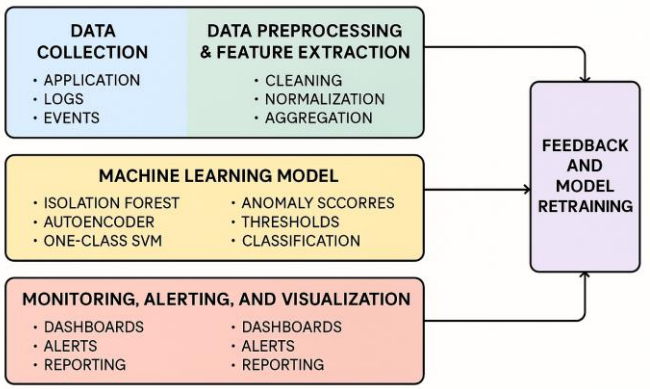

Figure 1: High-level system architecture for data-driven anomaly detection pipeline in software products

The key data collection layer components are as follows:

- *Application Monitoring Agents*: lightweight agents or SDKs embedded in applications to collect telemetry data without significantly impacting application performance.
- *Application Performance Monitoring Tools*: tools capture high-level application metrics such as OpenTelemetry, Prometheus, or commercial APMs (e.g., Dynatrace, Datadog).
- *Log Collection Tools*: solutions that gather structured and unstructured logs standardizing them into a centralized logging system.

Data is collected from various sources such as application logs (structured (JSON, XML) and unstructured (plain text) logs generated by application components capturing events, errors, and execution flows), performance counters (key metrics such as response time, request rate, latency, throughput, CPU/memory utilization, and error rates), event streams (time-ordered sequences of application events including user interaction events (clicks, session actions), API request traces, and transaction flows) and optional system-level signals (metrics such as resource utilization and system health indicators, if relevant to application behavior).

The tools and technologies that can be used involve logging agents (Fluentd, Logstash), metrics collectors (Prometheus, OpenTelemetry SDKs), event streaming platforms (Apache Kafka or Redis Streams for reliable, high-throughput data streaming, supporting low-latency anomaly detection).

4.2 Data Preprocessing & Feature Extraction Layer

The data preprocessing and feature extraction layer serves as a crucial intermediary stage in the anomaly detection pipeline. Its primary objective is to convert the raw, often noisy and unstructured operational data collected from software applications into clean, structured, and high-quality feature representations suitable for effective machine learning analysis. This layer ensures that downstream machine learning models receive well-prepared inputs that accurately reflect the application's behavioral patterns. It is aimed to prepare raw data by cleaning, structuring, and transforming them into meaningful features suitable for machine learning models raw.

The main purpose and objectives of data preprocessing and feature extraction layer are as follows:

- clean and filter the raw data to eliminate noise, inconsistencies, and irrelevant information;
- structure unstructured data formats (especially logs) into organized datasets;
- engineer meaningful statistical, temporal, and behavioral features that capture both short-term anomalies and long-term patterns;
- normalize and scale features to ensure consistency and comparability across data streams, thereby improving machine learning model convergence and stability.

The core layer components include:

- *Data cleaning*: detect and remove duplicate records that may arise from repeated log transmissions or retries; handle missing values using appropriate imputation techniques (e.g., forward-fill, interpolation) or by discarding corrupted records when necessary; filter out non-informative or irrelevant events, such as health-check pings or debug-level logs that do not contribute to anomaly characterization.
- *Data parsing*: convert unstructured logs into structured formats by using regex patterns or log templates to extract key fields (e.g., timestamps, log levels, error codes), apply Natural Language Processing (NLP) techniques like keyword extraction or clustering (e.g., TF-IDF, topic modeling) for complex log messages; transform raw event streams into organized, timestamped event sequences suitable for sequence-based modeling (e.g., recurrent neural networks or sequence embeddings).
- *Feature extraction*: generate aggregated statistical features (mean, median, variance, minimum, maximum values within fixed or dynamic time windows), construct sequence features including event frequency counts and encodings, n-grams encoding of event or API call sequences to capture behavioral patterns and session-based features (e.g., number of actions per user session, duration of sessions), derive behavioral ratio features like errors per request, retry rates, failure ratios, or latency-to-throughput ratios that are sensitive indicators of system health, extract trend features such as rolling averages, exponentially weighted moving averages and anomaly likelihood scores based on recent behavior.
- *Data normalization*: apply normalization techniques to scale numerical features into a common range or standardize them to zero mean and unit variance, apply encoding strategies for categorical features such as one-hot encoding or ordinal encoding, depending on model requirements, manage time-alignment of features to handle asynchronous data sources ensuring temporal consistency across the feature set.

The tools used at the data preprocessing and feature extraction layer include Python scripts (pandas, NumPy), Apache Spark (for large-scale processing), or in-stream processing (Apache Flink).

4.3 Machine Learning Model Layer

The machine learning model layer frames the core ML inference engine for anomaly detection. It is the central analytical component of the anomaly detection system responsible for analyzing preprocessed data and generating anomaly predictions. This layer utilizes a variety of machine learning algorithms to model the normal behavioral patterns of software applications and detect deviations indicative of anomalies. It is designed to handle the diverse nature of software application data, which can be high-dimensional, time-dependent, and partially labeled or entirely unlabeled.

The primary objectives of the layer in the system architecture is to model the complex dynamics of software application behavior, to detect unusual patterns or deviations that signal potential software anomalies and to provide real-time or near-real-time inference capabilities to support proactive issue detection.

The main categories of ML models that can be used to predict or score anomalies in the software application behavior include:

- *Unsupervised learning*: used when labeled anomaly data is scarce or unavailable which is common in real-world software systems. The ML algorithms utilized include:

– Isolation Forest: a tree-based ensemble method effective for high-dimensional datasets that isolates outliers based on feature splitting mechanisms producing anomaly scores based on how easily a data point can be isolated;

– Autoencoders: neural network architectures trained to reconstruct normal data patterns; higher reconstruction errors typically indicate anomalous behavior. Suitable for complex, non-linear feature spaces and temporal data sequences (e.g., LSTM Autoencoders);

– Clustering Models: algorithms like DBSCAN or k-means that identify clusters of "normal" behavior, with points outside these clusters flagged as anomalies. Useful for systems where behavioral modes naturally cluster.

- *Semi-Supervised learning*: applied when normal data is abundant but anomalous data is rare or undefined. One-Class Support Vector Machines is a promising approach to learn the boundary of normal data distribution in the feature space and classify any point falling outside the learned boundary as anomalous data and identifies outliers. One-class SVM is effective for compact feature spaces with stable normal behavior patterns.

- *Supervised learning*: utilized when historical labeled datasets are available, typically after substantial data collection and manual labeling efforts. Gradient Boosting Machines models such as XGBoost or LightGBM efficiently classify data into normal and anomalous classes capable of capturing non-linear dependencies and handling heterogeneous feature types. When rich labeled datasets are available deep learning classifiers can be utilized to classify different types of anomalies (e.g., performance vs. functional anomalies), especially when raw logs or sequences are used as input.

Depending on the modeling approach and system requirements, the output from the ML layer can be:

- raw anomaly scores: continuous scores representing the likelihood or severity of an anomaly, useful for flexible thresholding strategies;
- binary anomaly labels: simple "normal" vs. "anomalous" classification outputs, suitable for triggering immediate alerts;
- multi-class anomaly classification: labels that distinguish between different anomaly categories (e.g., performance degradation, functional failure, behavioral drift), providing more context for issue diagnosis;
- confidence intervals: optional uncertainty estimates accompanying predictions, especially useful for critical applications where decision risk needs quantification.

Model serving options include microservice model API, inference server or embedded model inference for low-latency environments). In the microservice-based model API strategy models are deployed as RESTful APIs behind scalable services (Docker, Kubernetes) that ensures flexible and language-agnostic integration with existing application monitoring tools. Inference servers strategy uses dedicated ML serving frameworks for high-throughput low-latency inference at scale and supports advanced features like batching, model versioning, and hardware acceleration (GPU/TPU inference). Embedded model inference strategy is based on utilization of lightweight models embedded directly into application processes or edge environments for ultra-low-latency predictions without external service calls. Embedded model inference is suitable for mission-critical, high-frequency anomaly detection tasks within microservices.

The design considerations for the ML layer include model selection criteria based on data availability (labeled/unlabeled), inference latency requirements, and interpretability needs, model versioning and lifecycle management for tracking and deploying model versions systematically and design for horizontal scaling to handle growing application telemetry volumes.

4.4 Model Explainability

Anomaly detection explainability is crucial in software applications because it bridges the gap between automated detection and human understanding, fostering trust, transparency, and actionable insights. Without explainability, anomaly detection systems often operate as opaque black boxes, making it difficult for developers and operators to comprehend why specific behaviors are flagged as anomalous.

By providing clear, interpretable reasons—such as highlighting which features or patterns contributed most to an anomaly—teams can more efficiently diagnose issues, reduce false positives, and improve system reliability. Explainability also enhances accountability in critical environments by enabling organizations to audit decisions and ensure that detection mechanisms align with operational goals and compliance requirements. Explainability transforms anomaly detection from a purely technical tool into a practical decision-support system that accelerates root cause analysis and promotes faster incident resolution.

Incorporating explainability outputs alongside model predictions for improved transparency and developer trust using:

- *SHAP (SHapley Additive exPlanations) Integration*: provides game-theory-based explanations that attribute how much each feature contributed to a particular prediction (in this case, the anomaly score). Use cases include explanation why a specific software request or session was flagged as anomalous as well as advising developers to identify which features (e.g., error rate, API latency, log frequency) had the strongest impact on the anomaly decision.
- *LIME (Local* Interpretable *Model-agnostic Explanations) Integration*: works by perturbing input features and observing changes in predictions, providing local approximations for individual anomalies. Possible use cases are quick explanation of individual anomaly cases, especially useful for deep learning models like autoencoders where feature interactions are opaque and on-demand explanation service that allow developers or operators to request LIME explanations via API. Dedicated LIME microservice takes in feature vectors and returns an interpretable feature importance visualization. When integrated with developer dashboards clicking on an anomaly alert a LIME explanation is fetched and displayed. LIME explanations can be used during model validation to help understand false positives or negatives.

4.5 Anomaly Scoring & Decision Logic

The anomaly scoring and decision logic layer serves as the interpretive and decision-making component of the anomaly detection system. It transforms the raw anomaly scores or labels generated by machine learning models into actionable decisions, such as triggering alerts or escalating anomalies for further analysis. This layer ensures that the system balances sensitivity (detecting true anomalies) and specificity (avoiding false positives), adapting to evolving software application behavior through flexible, rule-driven logic. Its main purpose and objectives are to interpret and post-process the outputs of machine learning models, to apply business-appropriate logic to distinguish between benign anomalies and actionable incidents and to manage alerting sensitivity dynamically, reducing alert fatigue while maintaining timely anomaly detection as well as optionally to incorporate domain knowledge through rule augmentation to handle edge cases that statistical models may overlook.

The core functional components and the decision logic techniques include anomaly scoring aggregation, thresholding techniques, rule augmentation and anomaly prioritization and labeling.

Anomaly scoring aggregation aggregates model outputs (e.g., probability scores, reconstruction errors, distance metrics) into unified anomaly scores and supports both direct usage of single-model scores and combined scoring from multiple models to create a robust anomaly detection signal. In addition can include smoothing techniques (e.g., exponential moving averages) to stabilize noisy predictions in high-frequency environments.

Thresholding techniques that can be utilized are as follows:

- *Static thresholding*: uses fixed pre-defined threshold values to classify anomalies. It is suitable for environments with well-understood, stable behavior patterns and is simple and easy to audit with low operational complexity. The main disadvantage of the single threshold approach is it is prone to false positives or missed anomalies in dynamic environments.
- *Dynamic thresholding*: adjusts thresholds in real-time or over rolling time windows accounting for fluctuations in system behavior (e.g., time-of-day patterns, seasonal load changes). The main techniques include rolling mean and standard deviation thresholds, percentile-based adaptive thresholds (e.g., top 5% of anomaly scores over the past hour) and exponentially weighted moving average thresholds. Compared to static thresholding these techniques are more resilient to natural system variability and reduces false positives but requires careful tuning to avoid masking genuine anomalies.
- *Multi-model ensemble scoring*: combines outputs from multiple ML models (e.g., unsupervised and supervised models) to improve detection robustness. The ensemble techniques include weighted average scoring, majority voting on binary anomaly labels and anomaly score stacking with meta-learners. The multi-model ensemble scoring balances strengths and weaknesses of different algorithms and improves overall system reliability but is more computationally intensive and requires careful calibration of ensemble strategies.

The additional rule augmentation can also be utilized as optional business logic validation layer. Rule augmentation implements customizable business logic rules on top of ML outputs to filter, prioritize, or contextualize anomalies. Example rules that can be utilized suppress anomalies during maintenance windows, escalate anomalies affecting critical transaction paths or only trigger alerts if anomalies persist across multiple time windows (debounce logic). The rule augmentation enables incorporation of domain-specific knowledge, making the system more relevant to operational

teams and allows human-in-the-loop controls where engineers can modify or override decision logic without model retraining.

The primary output of the anomaly scoring and decision logic layer are discrete anomaly events, annotated with metadata (e.g., timestamp, anomaly type, score, affected application component) as well as alerting mechanisms and event streaming emission of enriched anomaly events to real-time event buses for further consumption by monitoring dashboards or incident response pipelines and logging captured anomalies to a feedback store enabling human validation and future model retraining.

4.6 Monitoring, Alerting and Visualization Layer

The monitoring, alerting, and visualization layer is the critical interface between the anomaly detection system and human operators. Its primary function is to convert the analytical outputs – anomaly detections, scores, and classifications, into actionable insights through intuitive dashboards, real-time alerts, and periodic reports. This layer enhances operational visibility, enables rapid response to emerging anomalies, and supports long-term trend analysis of software application behavior. The purpose of this layer is to provide real-time observability of anomaly detection outputs through interactive dashboards, to deliver proactive alerts to relevant teams for immediate anomaly resolution, to offer historical insights and anomaly trends to support root cause analysis and capacity planning and to integrate seamlessly into existing operational workflows and observability platforms to minimize friction and maximize operational efficiency.

The core functional layer components include:

- *Dashboard tools*:

– real-time anomaly dashboards: interactive visual dashboards (e.g., Grafana, Kibana) to monitor anomaly scores, labels, and incident trends in near real-time, dynamic visualizations such as time-series charts, heatmaps, and correlation matrices to visualize anomalies alongside application metrics (latency, throughput, error rates) and drill-down capabilities to explore anomalies at different levels: system-wide, application-specific, or even component/module-specific;

– historical anomaly analytics: persistent storage of anomaly events enables historical querying and visualization, trend charts and anomaly count histograms to analyze daily, weekly, or monthly behavior patterns and visual correlation between anomaly occurrence and operational incidents (e.g., deployments, infrastructure changes).

- *Alerting tools*:

– real-time notifications: configurable alert triggers based on anomaly flags, severity levels, or aggregated anomaly scores and integration with established alerting pipelines such as Prometheus Alertmanager for metric-based alerts, PagerDuty for incident escalation management, Slack or Microsoft Teams for developer and operational team chat notifications, e-mail notifications for broader team communications or escalation paths;

– custom alert policies: threshold-based alerts (static or dynamic), frequency-based alerting (e.g., only alert if N anomalies in T minutes) and conditional alerting (e.g., escalate if critical anomaly detected during business hours);

– alert suppression logic: alert deduplication, silencing, or maintenance window exclusions to reduce alert fatigue and irrelevant noise.

- *Reporting tools*:

– scheduled summary reports: automated daily or weekly reports summarizing anomaly detection activity that include number of anomalies, types of anomalies, top affected services or components, and severity breakdowns;

– incident retrospective support: integration of anomaly summaries with post-incident review processes and timeline views overlaying anomalies with system events (e.g., deployments, incidents);

capacity and risk reporting: monthly or quarterly anomaly trend reports to assist in capacity planning, identifying application hotspots, or monitoring regression risks after code changes.

4.7 Feedback and Model Retraining

The feedback and model retraining loop plays a critical role in maintaining and improving the accuracy, relevance, and robustness of the anomaly detection system over time. It establishes a continuous learning process, allowing the detection models to adapt to evolving application behaviors, changing workloads, software updates, and operational patterns. This layer ensures the anomaly detection system remains effective in the face of concept drift, seasonal behavior changes, and system upgrades. Its main purpose and objectives is to ensure continuous improvement of model performance through operational feedback, to reduce false positives and false negatives by incorporating human insights into model updates, to detect and respond to data distribution shifts (concept drift) before performance degradation affects downstream systems and to automate the retraining and deployment process for machine learning models, ensuring agility and scalability.

The main feedback mechanisms comprise:

- *Human-in-the-loop feedback*: manual confirmation of anomalies through:

– interactive feedback interfaces: integration of UI components within dashboards (e.g., Grafana annotations, custom web apps) where operators can label anomalies as “True Anomaly”, “False Positive”, or “Needs Investigation”;

– feedback logging: collected human feedback is stored in a structured feedback repository (e.g., PostgreSQL, Elasticsearch) including metadata such as timestamp, user ID, anomaly score, and resolution status;

– feedback-driven data augmentation: validated anomalies and false positives are appended to training datasets, enhancing both supervised and semi-supervised models.
– closed-loop improvement: facilitates root cause analysis and highlights blind spots in model logic by promoting transparent feedback cycles.

- *Auto-retraining pipelines*:

– active learning integration: employs active learning strategies to selectively sample data points with high model uncertainty or borderline anomaly scores for human labeling, prioritizing the most informative data.
– automated data pipelines: end-to-end pipelines that collect labeled data, update datasets, and retrain models in scheduled cycles (e.g., weekly, monthly), leveraging orchestration tools like Apache Airflow, Kubeflow Pipelines, or Prefect;
– CI/CD integration: CI/CD pipelines (e.g., Jenkins, GitLab CI, GitHub Actions) that automatically trigger retraining jobs, conduct model validation (e.g., cross-validation, A/B testing), and deploy validated models to production environments;
– model validation gates: retrained models must pass predefined performance thresholds (e.g., reduced false positive rate, improved F1-score) before deployment.

- *Drift detection mechanisms*:

– statistical drift detectors: techniques that measure shifts in input data distribution over time;
– concept drift detectors: model performance monitoring through monitoring residuals, prediction confidence drop-offs, or error distribution shifts to identify degradation in prediction quality;
– automatic drift triggers: when significant data or concept drift is detected, the system flags the model as "outdated," triggering automatic retraining pipelines or notifying ML operations teams for manual intervention;
– visualization of drift: drift monitoring dashboards showing changes in feature distributions, model accuracy over time, and anomaly volume shifts.

The operational workflow of the feedback and model retraining loop is as follows:

(a) anomalies are detected and re displayed on dashboards;
(b) human operator reviews anomalies and labels anomalies with feedback;
(c) feedback repository collects and logs this information;
(d) drift detection runs continuously and detects when data distribution or model performance degrades;
(e) retraining pipeline is automatically initiated to update training data with new labeled samples, retrain and validate model on augmented datasets and deploy updated model using CI/CD workflows;
(f) new model serves predictions continuously monitored by the scoring and decision logic layer.

5. DISCUSSION

In proposed high-level system architecture for data-driven anomaly detection pipeline in software products data flows are collected from various application sources through lightweight agents or APIs to the data collection layer. From there, data is normalized and tagged with application version or environment metadata, and transmitted to centralized pipelines. Real-time data streams are directed to a message broker (e.g. Kafka) or time-series database (e.g. InfluxDB) for immediate consumption. Batch data is stored in a data lake or cold storage for historical model training, drift detection, or forensic analysis.

In the context of the ML model layer, datasets represent structured collections of real-world information that help models learn underlying patterns [2]. Each entry in a dataset reflects a specific instance, defined by a set of variables, which together form the input space for the model. The complexity and size of these datasets can vary significantly depending on the domain, directly influencing the depth and accuracy of the resulting models. The Data Collection Layer ensures that all downstream components—preprocessing, machine learning inference, explainability, and alerting—receive consistent, accurate, and timely data. A robust and scalable data collection architecture minimizes data loss, supports multi-environment observability (e.g., production, staging), and lays the groundwork for high-accuracy anomaly detection by capturing rich operational signals from the application.

The success of machine learning systems fundamentally depends on the quality of the data pipeline – specifically, how data is collected, cleaned, and structured [11]. The discussion underlines the need for greater attention to data preprocessing as a foundational aspect of any ML solution. So, we propose to include Data Preprocessing & Feature Extraction Layer, which transforms diverse raw operational data into robust, information-rich feature sets that improves the accuracy and robustness of anomaly detection models, reduce noise and false-positive rates by ensuring that models focus on relevant, high-quality signals and enables the detection of both known and unknown (novel) anomalies by capturing complex behavioral patterns.

By systematically applying preprocessing and feature extraction, the pipeline ensures that machine learning algorithms operate on data representations that closely reflect the operational state of the software applications, thereby maximizing the effectiveness of anomaly detection. The machine learning model layer transforms preprocessed application data into actionable anomaly insights through advanced ML algorithms. It ensures rapid, reliable, and explainable anomaly detection, serving as the analytical engine that enables proactive system health monitoring, root cause analysis, and automated operational responses.

Anomaly scoring & decision logic layer acts as the decision gateway, translating probabilistic or continuous anomaly signals into clear, operationally meaningful alerts. It provides the flexibility to adapt the system to changing business contexts, ensures actionable precision, and supports continuous system improvement via adaptive thresholds and rule tuning. By balancing statistical detection with practical business logic, the anomaly scoring and decision logic layer minimizes false alarms, reduces mean-time-to-detect and enhances the overall trustworthiness of the anomaly detection framework.

The monitoring, alerting, and visualization layer ensures that anomalies detected by the system do not remain hidden within technical logs or backend systems. It surfaces critical insights to human operators in a timely, clear, and actionable manner, fostering faster detection-to-response cycles, reduced Mean Time to Detection (MTTD) and Mean Time to Resolution (MTTR), increased transparency and accountability through accessible historical records and reports and continuous feedback loops to improve anomaly detection efficacy based on user feedback and observed outcomes. By providing intuitive visibility and actionable alerts, the monitoring, alerting, and visualization layer transforms raw anomaly data into operational intelligence that directly improves software reliability and user experience.

The main benefits of the feedback and retraining loop are improved model adaptability to changing application environments and user behaviors, reduced operational burden through automation of retraining and deployment, enhanced model robustness, capturing rare or emerging anomaly patterns via active learning, transparent system improvement, with a clear audit trail of feedback, retraining, and performance shifts and reduced alert fatigue through lower false positive rates and more accurate anomaly detection. This layer transforms the anomaly detection system from a static, one-time deployment into a living, learning system that evolves with the software application lifecycle. It embeds human expertise into the machine learning lifecycle and ensures the system remains reliable, trustworthy, and aligned with operational realities.

6. CONCLUSION

The paper presents a system architecture for effective data-driven anomaly detection in software products. The suggested architecture creates a full lifecycle for application anomaly detection from real-time data capture to ML-based detection, automated alerts, and continuous model improvement, making software systems more resilient, adaptive, and transparent. The proposed methodology is adaptable, interpretable, and suitable for real-time deployment in diverse software environments. The system architecture is designed to be modular, scalable, and adaptable for various types of software applications (web, desktop, microservices).

Future work will focus on integrating the suggested high-level system architecture within DevOps and CI/CD pipelines for early anomaly detection and extending it for microservices and distributed architectures, utilizing adaptive learning methods to handle software evolution and dynamic user behavior and integration with explainable AI (XAI) techniques to improve interpretability and trust in anomaly detection decisions.

ACKNOWLEDGEMENT

The research reported is funded by the project with contract №: KP-06-N57/4 from 16.11.2021, funded by the Bulgarian National Science Fund.

REFERENCES

1. W. Yuan, S. Ying, X. Duan, H. Cheng, Y. Zhao, and J. Shang. **MidLog: An automated log anomaly detection method based on multi-head GRU**, *Journal of Systems and Software*, vol. 226, 2025.
2. X. Larriva-Novo, M. Vega-Barbas, V. A. Villagra, D. Rivera, M. Alvarez-Campana, and J. Berrocal. **Efficient distributed preprocessing model for machine learning-based anomaly detection over large-scale cybersecurity datasets**, *Applied Sciences*, vol. 10, no. 10, 2020.
3. F. Sönmez, M. Zontul, O. Kaynar, and H. Tutar. **Anomaly detection using data mining methods in IT systems: a decision support application**, *Sakarya University Journal of Science*, vol. 22, no. 4, pp. 1109-1123, 2018.
4. A. O. Elfaki, S. L. Fong, P. Vijayaprasad, M. G. M. Johar, and M. S. Fadhil. **Research Article Using a Rule-based Method for Detecting Anomalies in Software Product Line**, *Research Journal of Applied Sciences, Engineering and Technology*, vol. 7, no. 2, pp. 275-281, 2014.
5. S. Kong, J. Ai, M. Lu, and Y. Gong. **GRAND: GAN-based software runtime anomaly detection method using trace information**, *Neural networks*, vol. 169, pp. 365-377, 2024.
6. G. Long, K. Yu, S. Yang, X. Zhou, X. Shen, X., and N. Lu. **Software anomaly detection technology based on deep learning**, *Procedia Computer Science*, vol. 259, pp. 1123-1129, 2025.
7. L. Cherkasova, K. Ozonat, N. Mi, J. Symons, J., and E. Smirni. **Automated anomaly detection and performance modeling of enterprise applications**, *ACM Transactions on Computer Systems (TOCS)*, vol. 27, no. 3, pp. 1-32, 2009.
8. M. V. Kornilov, et al. **Coniferest: a complete active anomaly detection framework**, *Astronomy and Computing*, vol. 52, 2025.
9. A. Capizzi, S. Distefano, L. J. Araújo, M. Mazzara, M., Ahmad, and E. Bobrov. **Anomaly detection in devops toolchain**, in *Software Engineering Aspects of Continuous Development and New Paradigms of*

Software Production and Deployment, JM. Bruel, M. Mazzara, B. Meyer, Eds. Lecture Notes in Computer Science, vol. 12055, Springer, Cham: Springer International Publishing, 2019, pp. 37-51.

10. P. Arcaini, A. Gargantini, E. Riccobene, and P. Vavassori. **A novel use of equivalent mutants for static anomaly detection in software artifacts**, *Information and Software Technology*, vol. 81, pp. 52-64, 2017.
11. O. S. Ndibe. **Integrating Machine Learning with Digital Forensics to Enhance Anomaly Detection and Mitigation Strategies**, *International Journal of Advance Research Publication and Reviews*, vol. 2, no. 5, pp 365-388, 2025.