Adaptive issue detection in holistic optimization of distributed data streaming systems

Victoria Vysotska[1, †], Iryna Kyrychenko[2, †] and Vadym Demchuk[2, †, *]

[1] *Lviv Polytechnic National University, Stepan Bandera street 12, Lviv, 79013, Ukraine*
[2] *Kharkiv National University of Radio Electronics, Nauky Ave. 14, Kharkiv, 61166, Ukraine*

Abstract
While existing Holistic Adaptive Optimization Techniques (HAOT) in distributed data streaming systems excel at parameter tuning, they are insufficient for identifying latent vulnerabilities that can lead to future pipeline failures. To address this gap, this paper introduces a novel framework for Adaptive Issue Detection integrated within the HAOT architecture. The core of this framework is a novel Competency Module that enhances pipeline intelligence through two synergistic components: a Machine Learning (ML) Model for Issue Detection and an interactive Control Center.
The ML model employs a range of algorithms, including anomaly detection and reinforcement learning, to incrementally analyze historical and real-time pipeline data. It proactively identifies issues such as the Small Files Problem, Data Skew, Resource Exhaustion, Throughput Degradation, and Latency Spikes.
Complementing the ML model is the Control Center, a web-based interface that translates the model's complex analytics into clear, data-driven recommendations and risk assessments. Through intuitive charts and alerts, it provides operators with a centralized view of pipeline health, identified risks, and clear, data-driven recommendations. This human-in-the-loop system not only facilitates informed decision-making but also allows user feedback to guide the optimization process. By creating a symbiotic feedback loop between automated ML discovery and human operational expertise, our framework provides a robust, adaptive solution for enhancing the performance, efficiency, and reliability of distributed data streaming systems.

Keywords
Distributed Data Streaming Systems, Early Issue Detection, Performance Tuning, Parameter Tuning, Machine Learning, Realtime Configuration Adaptation, Stream Processing, Automatic Parameter Tuning, System Optimization, Infrastructure Cost Savings, Data Pipeline Management, Evaluation Methods, Performance Metrics, Workflow Optimization, Data Management Techniques, Pipeline Efficiency, Process Improvement, Shift-Left Architecture

1. Introduction

The history of data processing architecture is marked by a series of advancements, each designed to overcome the shortcomings of the previous generation. Initially, the Extract-Transform-Load (ETL) framework was the standard. This process (Fig. 1) involved pulling raw data from on-premises, transforming it using constrained storage and computational power, and finally depositing the refined data into a centralized data warehouse. While a functional solution for its time, this traditional ETL method had notable weaknesses, including limited processing power and scalability, which made it challenging to manage and analyze growing volumes of historical data efficiently [1].

The second generation of data architecture, centered on the Extract-Load-Transform (ELT) model, was a direct response to earlier limitations. ELT prioritizes speed by first loading raw data into a scalable cloud Data Lake before transforming it with powerful, parallel computing resources. This approach, often using a medallion framework, delivers high performance and supports advanced analytics for downstream applications (as depicted in the ETL vs. ELT diagram in Fig. 2) [2].

MoDaST 2025: Modern Data Science Technologies Doctoral Consortium, June, 15, 2025, Lviv, Ukraine
* Corresponding author.
† These authors contributed equally.
Victoria.A.Vysotska@lpnu.ua (V. Vysotska); iryna.kyrychenko@nure.ua (I. Kyrychenko); vadym.demchuk@nure.ua (V. Demchuk);
0000-0001-6417-3689 (V. Vysotska); 0000-0002-7686-6439 (I. Kyrychenko); 0000-0003-3700-2344 (V. Demchuk);

However, this ELT model is not without its flaws. Its batch-based nature can create data delays and inconsistencies. A more significant issue is operational inefficiency, where different teams (e.g., AI, Marketing) build redundant data pipelines for their specific needs. This practice, often undocumented, leads to wasted resources, increased costs, and maintenance difficulties. The addition of Reverse ETL processes further exacerbates these problems by creating additional copies of data and increasing processing overhead [3].

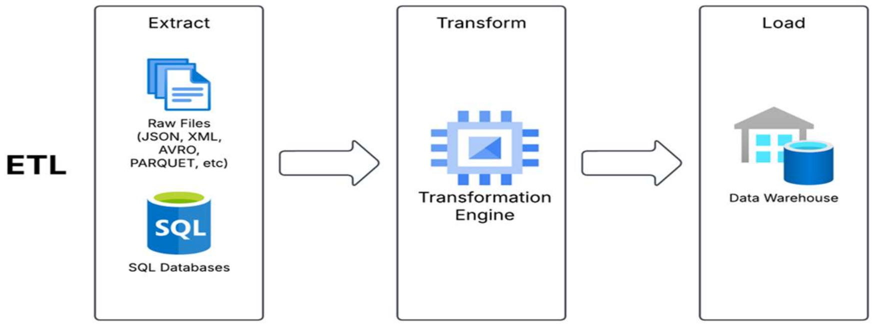

Figure 1: An architecture of the ETL flow

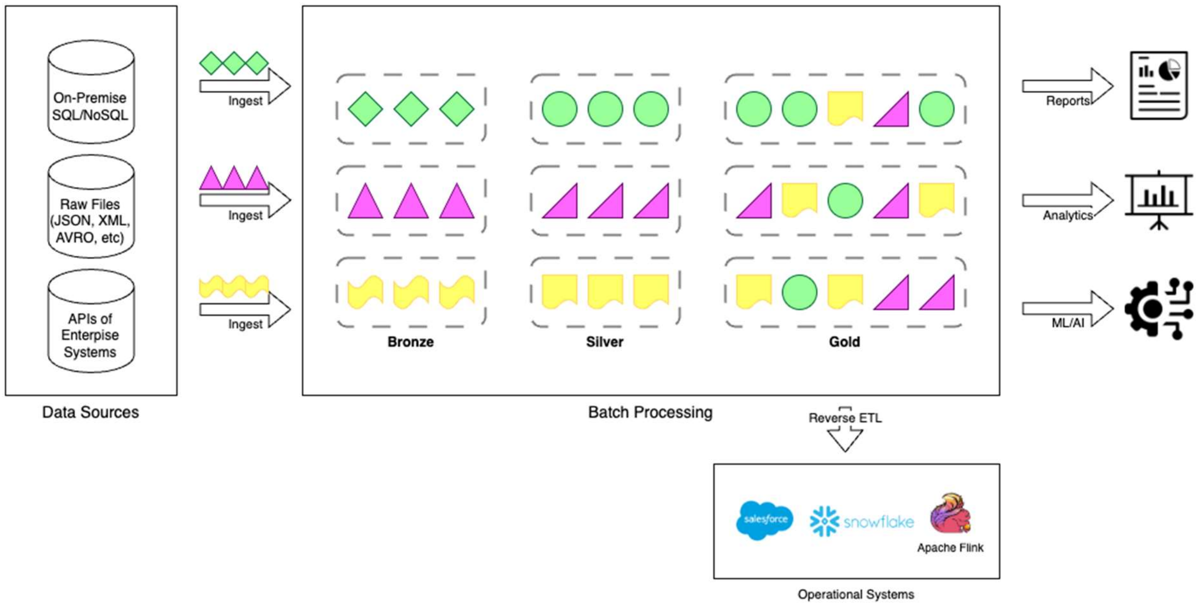

Figure 2: An architecture of ELT flow

In today's competitive business world, data is a critical asset, and organizations need immediate access to reliable, contextualized data to foster innovation and thoughtful decision-making. The "Shift-Left Architecture" has been introduced as a solution to meet these demands.

Inspired by "Shift-Left Testing" in software development, this architecture moves data processing and governance closer to the origin of the data. It emphasizes processing data in real-time as it is generated, utilizing tools such as Apache Kafka and Apache Flink. This approach focuses on creating reliable "data products" at the source for systems like Snowflake or Databricks. Following a "build once, use many times" principle, this method ensures data is prepared efficiently for widespread reuse, which reduces redundancy and boosts data quality throughout the company [4].

As illustrated in Figure 3, the Shift-Left Architecture redesigns data workflows by performing tasks like cleaning and enrichment early. This ensures that all downstream systems, from analytics platforms to operational microservices, receive high-quality, structured data, thereby minimizing repetitive work and reducing the time required to gain valuable insights [5].

One of the most significant aspects of the Shift-Left approach is its strong alignment with the principles of a data mesh, which it facilitates by creating real-time data products. This architecture utilizes a combination of technologies, including Apache Kafka, Apache Flink, and Apache Iceberg, to unify both operational (transactional) and historical (analytical) data processing. The result is a consistent and reliable stream of high-quality data that can be processed instantly or loaded into advanced analytics platforms, such as Snowflake, Databricks, or Google BigQuery. This creates a solid, cohesive foundation for an organization's artificial intelligence and data analytics initiatives [4, 5].

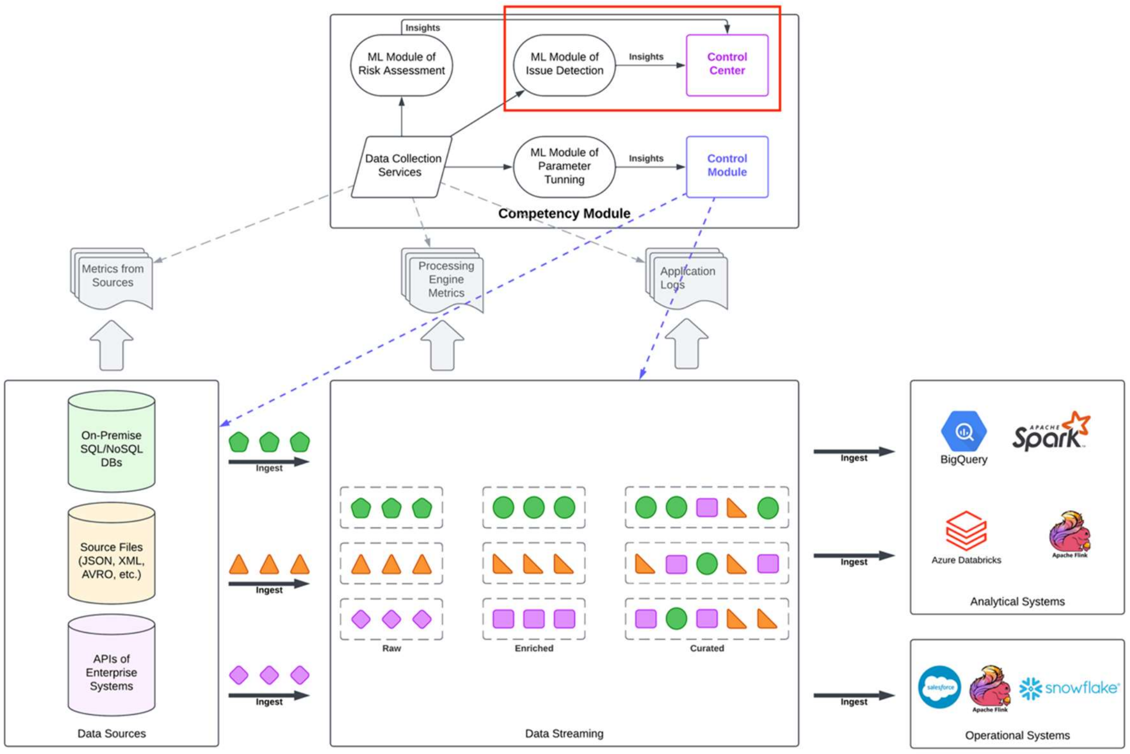

Figure 3: Shift-Left Architecture with HAOT Applied

Recent research on the Shift-Left Architecture has revealed the importance of integrating early-stage anomaly detection and risk mitigation protocols. This approach will focus on identifying and resolving data integrity issues — including quality, schema, and security defects — at the source, thereby preventing their downstream escalation and minimizing costly operational rework. A specialized Competency Module was suggested within the architecture to facilitate this, providing data engineers with the necessary tools and expertise for proactive issue resolution. The early issue detection significantly enhances the overall resilience and operational effectiveness of the data architecture [6].

In this article, we design and evaluate a framework for adaptive issue detection within streaming systems.

2. Related Works

2.1. Limitations of Traditional ETL and ELT Paradigms

A central challenge in data engineering is efficiently and reliably processing vast, real-time data streams. Foundational methodologies such as Extract-Transform-Load (ETL) and Extract-Load-Transform (ELT) have been extensively analyzed. However, their architectural limitations present significant obstacles for industries that require low-latency and high-fidelity data. This review examines the evolution of these paradigms, focusing on their inherent issues and the subsequent attempts to resolve them.

Initially, the ETL paradigm became the standard for data warehousing. However, as Nishanth Reddy Mandala notes in his research, “ETL in Data Lakes vs. Data Warehouses” [7], its design suffers from inherent issues. The required pre-load transformation step causes significant delays, postponing data availability. Additionally, its rigid, schema-on-write approach reduces flexibility. It also results in high costs for maintenance and scaling, making it less suitable for the evolving needs of Big Data environments.

To counteract ETL's limitations, the ELT approach was introduced, which delays transformation until after the data is loaded — a core concept of the Lakehouse paradigm described by Gupta and Yip in their book, “Delta Lake - Deep Dive” [8]. While this resolves the initial latency issue, the multi-hop architecture of ELT creates its own significant challenges. Data freshness is impacted by delayed updates, as longer pipelines slow refresh cycles. Development becomes less efficient, with increased time-to-market and duplicated processing tasks across different business units [9]. This redundancy also raises costs, as computationally intensive transformations are repeated multiple times. Most importantly, this fragmented approach often leads to inconsistent data across operational and analytical systems, preventing a unified, real-time view.

2.2. The Emergence of the "Shift-Left" Architecture

In response to these ongoing challenges, new paradigms have been introduced, such as the one described in the article “Shift Left: Headless Data Architecture” [10] and the unified approach explored in “An Overview of Shift Left Architecture” [11]. This "Shift-Left" approach fundamentally alters the data pipeline by moving processing closer to the data source. Its primary goal is to address the issues inherent in traditional models by enhancing processing efficiency, facilitating real-time data handling, and consolidating workloads. A key benefit is the potential for improved data quality, as it enables early detection and correction of issues, preventing their spread downstream.

2.3. The Critical Challenge of "Bad Data" in Streaming Architectures

However, as Adam Bellemare critically observes in his work, “Shift Left. Unifying Operations and Analytics With Data Products,” this architecture's design introduces a major vulnerability: the detection and management of "bad data" [12]. The reliance on immutable, append-only data streams means that corrupted data, once introduced, cannot be easily changed or removed. Unlike traditional ETL/ELT pipelines that can "stop the world" to make corrections, a faulty record in a streaming architecture can cascade downstream, leading to severe and potentially irreversible business impacts. Bellemare highlights key limitations, including the high cost of data reconciliation and the fact that downstream consumers bear the burden of ensuring data consistency without owning the source system.

While Bellemare proposes reactive strategies—such as robust validation ("Prevention"), corrective notifications ("Issue Correction Events"), and stream recreation ("Rewind, Rebuild, and Retry") — these methods mainly deal with errors after they happen. This creates a significant gap: the absence of a proactive approach to foresee and prevent issues before they impact performance or lead to data loss. The Shift-Left architecture remains very vulnerable to early errors caused by incorrect data schemas, flawed processing configurations, or sudden workload changes [13].

To create an effective monitoring solution, it is essential to first gain a deep and detailed understanding of the problem domain. Distributed data streaming systems, while powerful, present a unique set of operational challenges that distinguish them from traditional monolithic or batch-based systems.

2.4. Analysis Issues in Distributed Data Streaming Systems

The subsequent studies, “Demeter: Resource-Efficient Distributed Stream Processing under Dynamic Loads with Multi-Configuration Optimization” [14] and “Towards Fine-Grained Scalability for Stateful Stream Processing Systems” [15], explored key challenges in streaming, including resource exhaustion, throughput decline, and latency spikes. They observed that uncontrolled resource use —

driven by data surges or stateful operations, such as windowed aggregations — can cause backpressure, node failures, or reduced service quality. Excessive uploads can deplete compute and storage resources, resulting in increased costs or service outages. Solutions such as horizontal scaling and dynamic configuration (e.g., Demeter) help optimize resource use during fluctuating loads.

The studies also addressed throughput issues, which occur when high data volumes overwhelm network bandwidth and hardware, resulting in congestion, packet loss, and processing delays. For instance, overloaded Kafka shards or Flink operators increase acknowledgment times, delaying real-time dashboards. To counter these problems, techniques such as compression (Avro/Protobuf), load balancing, and adaptive bitrate encoding are recommended to sustain throughput during traffic surges.

Lastly, structural bottlenecks — such as synchronous replication in geo-distributed systems or network congestion during peak periods — can cause delays exceeding 50 seconds in live streaming. Mitigation strategies include edge caching, asynchronous replication, and traffic-shaping protocols.

2.5. Analysis of Performance Bottleneck

The work "A Deep Dive into the Latest Performance Improvements of Stateful Pipelines in Apache Spark Structured Streaming" [16] examines performance bottlenecks in streaming data pipelines. A bottleneck is a component or resource that limits overall throughput or increases a system's latency, acting as a choke point in the data flow. The key performance metrics, latency (the processing delay) and bandwidth (the processing rate), are directly affected by the following constraints:

- Network bottlenecks occur when there is insufficient bandwidth to handle traffic, high latency caused by physical distance or congestion, or an inefficient network layout with excessive hops. For example, a video streaming service attempting to serve many users without a geographically distributed network will inevitably face buffering and low-quality streams, as its network becomes the primary bottleneck [17].
- A computational bottleneck occurs when a server's processor is consistently overloaded, resulting in a backlog of pending tasks and increased processing latency. In streaming systems, typical causes encompass algorithmic approaches that are computationally demanding (such as complex event processing), suboptimal code execution, or the overhead associated with data serialization and deserialization across diverse formats. An illustrative example is an application dedicated to real-time image processing, which saturates the CPU and consequently hampers overall system performance [17].
- Insufficient Random Access Memory (RAM) can significantly impair system performance. When an application's memory demands exceed the available physical RAM, the operating system employs memory page swapping to disk, a process that is substantially slower than direct memory access and can drastically reduce system efficiency. In the context of streaming analytics, this problem frequently occurs with stateful operations, such as maintaining extensive time-windowed aggregations (e.g., a 24-hour running average of user activity), which may result in out-of-memory errors if not properly managed [18].
- The speed at which data can be written to and read from persistent storage is a key performance factor for systems like Apache Kafka, which depend on continuously writing event logs to disk. Slow disk operations, often associated with traditional Hard Disk Drives (HDDs), can become a major bottleneck, causing backpressure to spread through the entire pipeline. Moving to faster storage options, such as Solid-State Drives (SSDs) or NVMe SSDs, is a common and effective way to address I/O-bound workloads [19].
- Distributed systems rarely operate in isolation. Their performance is often limited by the downstream services they rely on, such as databases or third-party APIs. A slow database query, inefficient indexing, or API rate limiting can become the main bottleneck for the entire streaming application, regardless of how well-optimized the streaming components are [18].

2.6. Classical Approaches of Issue Detection and Their Limitations

The current forms of anomaly detection rely on statistical and rule-based methods.

The research article “Developing Big Data anomaly dynamic and static detection algorithms: AnomalyDSD spark package” [20] utilizes statistical methods. These techniques model normal behavior with statistical distributions and flag data points that fall into low-probability areas. Common approaches include using the Z-score or Grubbs' test to spot points that deviate significantly from the mean, or applying moving averages to smooth data and detect deviations from a trend. While these methods are easy to implement and highly interpretable, they also have notable limitations. They usually assume the data follows a specific distribution (e.g., Gaussian), which is rarely true for complex system telemetry. Additionally, their dependence on fixed, manually set thresholds makes them fragile; they can't adjust to non-stationary data where the definition of "normal" changes, resulting in a high rate of false positives and alert fatigue.

The next research, “A comprehensive survey of anomaly detection techniques for high dimensional big data” [21], discusses rule-based systems. These systems employ a set of predefined rules and thresholds, often derived from domain expertise, to identify anomalies. For example, a rule might say, "alert if CPU utilization > 95% for 5 minutes." They excel at identifying known and well-understood failure modes. However, they need constant manual updates, are inflexible, and cannot detect new, unforeseen anomalies or "black swan" events, which are common in complex systems.

2.7. Identified Research Gap and Proposed Contribution

The analysis of existing literature reveals that while the Shift-Left architecture effectively addresses the latency and redundancy issues of ETL/ELT, it introduces a critical sensitivity to data quality that is not fully resolved by current, reactive strategies. Therefore, the next chapter proposes an extension to the Shift-Left architecture: a novel module designed to proactively identify and mitigate these potential issues much closer to the source, enhancing the resilience and reliability of real-time data pipelines.

3. Methodology of Adaptive Issue Detection

Addressing the identified gaps requires moving beyond a collection of disparate tools and toward a cohesive, intelligent monitoring platform. This section outlines the conceptual architecture for such a system, detailing its core principles, internal components, and the end-to-end data flow from ingestion to insight.

3.1. High-Level System Overview and Core Principles

The mission of the proposed module is to provide a unified, intelligent monitoring layer that transcends individual system components, offering a holistic and real-time view of the distributed system's health. Four core principles guide its design:

- Adaptive: The system must not be static. It must learn the unique behavioral patterns of the target system and automatically evolve its understanding as those patterns change over time, effectively handling concept drift.
- Real-Time: For detection to be actionable, it must occur with minimal latency. The architecture must be optimized for high-throughput, low-latency stream processing.
- Interpretable: The system must not be a "black box." To build operator trust and facilitate rapid remediation, it must provide clear, understandable explanations for why an anomaly was flagged. This principle mandates the integration of Explainable AI (XAI).
- Human-in-the-Loop (HITL): The system must be designed as a collaborative partnership between machines and humans. It must incorporate human expertise to validate anomalies, correct misclassifications, and guide model retraining, creating a virtuous feedback cycle that continuously improves performance.

This architecture represents a shift from a traditional monitoring tool to a comprehensive monitoring platform, where components for data transport, stream processing, ML inference, and human interaction are tightly integrated.

3.2. The Architectural Diagram: Internal Components and Data Flow

The proposed architecture comprises several key components that work in tandem. A high-level diagram would show the following flow (See Fig. 4):

- Data Sources (e.g., Streaming Platforms, Node Exporters) generate raw telemetry. The system must ingest a wide variety of data to build a holistic view. This includes low-level system metrics (CPU, memory, disk I/O, network statistics) scraped by agents like Prometheus; application logs aggregated by systems like the ELK Stack (Elasticsearch, Logstash, Kibana); and high-level, framework-specific metrics extracted directly from the streaming platforms' APIs (e.g., Kafka consumer group lag, Flink's backpressure indicators, Spark's micro-batch processing durations).
- A Telemetry Collector/Agent gathers this data.
- The data is ingested into a central Data Ingestion Bus (Apache Kafka), which serves as a durable and scalable buffer. Using Kafka provides critical architectural benefits: it decouples the data producers (the monitored systems) from the consumers (the anomaly detection logic), provides a durable buffer to handle temporary outages or processing delays, and allows multiple downstream systems (e.g., both the real-time detector and an archival system) to consume the same data stream.
- A preprocessing and feature engineering service (Kafka Streams) consumes data from the bus, cleans it, and creates features for the model. Its responsibility is to prepare the data for the ML model in real-time. This involves tasks such as parsing different log formats, normalizing metrics to a common scale (e.g., 0 to 1), imputing missing values, and performing feature engineering to create more informative signals for the model, such as calculating moving averages or rates of change.
- The processed data is sent to the ML Core, which performs real-time inference using a deployed model. The ML Core contains a Model Store for versioned models and an Anomaly Scorer.
- Detected anomalies are enriched and stored in a fast-access Anomaly Datastore (Elasticsearch).
- The Control Center (a UI/API layer) queries the Anomaly Datastore to populate dashboards and alerts for human operators.
- Operators interact with the Control Center, providing labels and feedback on anomalies, which are stored in a Feedback Database.
- This feedback triggers an MLOps Retraining Pipeline, which pulls data from the original telemetry stream and the Feedback Database to train, evaluate, and deploy a new, improved model version back into the ML Core's Model Store.

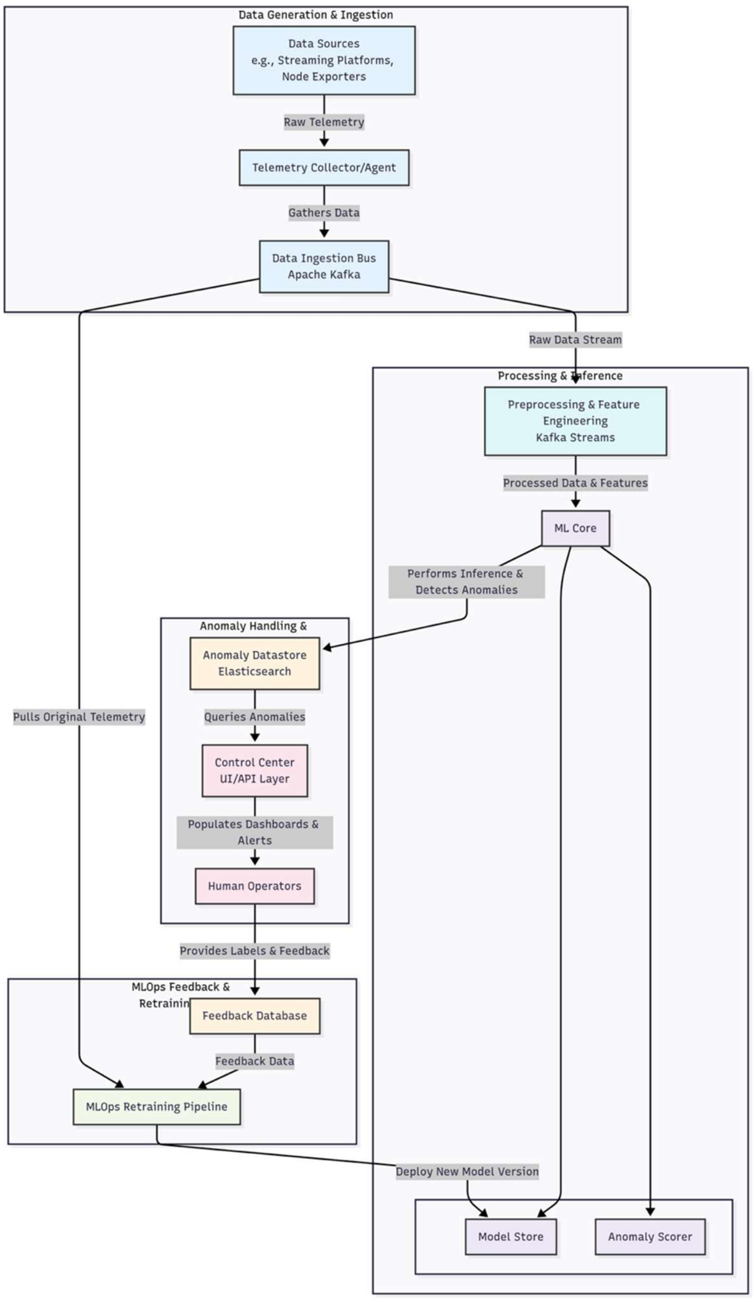

Figure 4: An architecture of the novel Issue Detection Service

This design explicitly treats the feedback loop not as an afterthought, but as a first-class architectural component — a hallmark of a mature MLOps practice.

The data collected from the distributed system — comprising metrics such as CPU utilization, memory usage, network throughput, application error rates, and consumer lag from every node and service — form a high-dimensional, multivariate time series. The term multivariate signifies that we are not just looking at one metric over time, but at hundreds or thousands of interdependent metrics simultaneously. The fundamental task of the ML model is to learn the complex, non-linear correlations and temporal dependencies within this high-dimensional data stream, thereby building a robust mathematical representation of normal system behavior. An anomaly is then defined as a significant deviation from this learned normal model.

3.3. A Comparative Analysis of Deep Learning Architectures

Given the complexity and high dimensionality of the data, deep learning models are the most suitable candidates. Several architectures are prominent in the field of time-series anomaly detection.

- Recurrent Neural Networks (RNNs) and LSTM Autoencoders. Long Short-Term Memory (LSTM) networks, a type of RNN, are specifically designed to handle sequential data. Their internal memory cells enable them to retain information over long periods, making them effective at learning temporal patterns. In the context of anomaly detection, they are often employed within an autoencoder architecture. An autoencoder consists of an encoder, which compresses the input time-series data into a low-dimensional latent representation, and a decoder, which attempts to reconstruct the original time series from this compressed form. The model is trained only on normal data. The idea is that the model becomes very good at reconstructing normal patterns. When it encounters an anomalous pattern, it has not seen before, it will struggle to reconstruct it accurately, resulting in a high reconstruction error. This error is then used as the anomaly score. LSTMs have been proven and are effective for capturing temporal dependencies, including seasonality and trends. While LSTMs outperform simple RNNs, they can still struggle to capture very long-range dependencies and complex interactions between different variables in high-dimensional series [22, 23].
- Transformer. Originally designed for natural language processing, the Transformer architecture has achieved notable success in time-series analysis. Its main innovation is the self-attention mechanism, which enables the model to evaluate the importance of all other data points in the sequence when processing a single point. This allows it to capture long-range dependencies and, importantly, the inter-series dependencies (such as the relationship between CPU usage on node A and network latency on node B) more effectively than RNNs. The Anomaly Transformer is a specialized version that explicitly models both "series-association" (temporal patterns) and "prior-association" (expected correlations) to enhance detection accuracy. While it is the state-of-the-art performance in capturing complex, long-range, and multivariate dependencies, the transformers are notoriously data-hungry and computationally expensive to train. Their application in time-series anomaly detection is a more recent and still-evolving area of research [24, 25].
- Other Architectures. The field is rapidly advancing. Other promising architectures include Generative Adversarial Networks (GANs), which use a generator/discriminator pair to learn the distribution of normal data, and very recent developments like Kolmogorov-Arnold Networks (KAN-AD), which aim to model data using smooth functions, potentially offering higher efficiency and better resilience to minor fluctuations in normal data [26].

3.4. Proposed Model: A Hybrid, Federated, and Continual Learning Approach

Synthesizing these requirements leads to a proposal for a hybrid and highly adaptive model architecture.

- Core Architecture: A Transformer-based autoencoder is selected as the core model. This choice is motivated by the need to effectively capture both the long-range temporal dependencies within individual metric streams and the complex, cross-metric correlations that are characteristic of distributed system failures.
- Training Paradigm: The system will employ a Federated Continual Learning approach. This paradigm is perfectly suited to the distributed and dynamic nature of the problem. On individual clusters or nodes, lightweight local models are trained on local telemetry data. This preserves data privacy and locality. Federated Aggregation models periodically send their parameter updates to a central server. The central server uses a continual learning strategy (e.g., federated averaging combined with regularization techniques to prevent catastrophic forgetting) to intelligently aggregate these updates into a robust global model. This global model captures a holistic understanding of the entire system's health.
- Drift Handling: The central server is responsible for explicit concept drift detection by monitoring the global data distribution and the performance of the aggregated model. When significant drift is confirmed (ideally with human validation), it can orchestrate an accelerated retraining cycle across the federation, ensuring the entire system adapts to the

new normal. This approach creates a scalable, privacy-preserving, and continuously adapting intelligence core.

The following table provides a comparative analysis of the leading ML architectures, justifying the proposed selection.

Table 1
Comparative Analysis of ML Architectures for Time-Series Anomaly Detection

Model Architecture	Core Mechanism	Strengths	Weaknesses	Suitability for Streaming Telemetry
LSTM Autoencoder	Reconstruction Error via Recurrent Layers	Proven effectiveness; excellent for capturing clear temporal patterns and seasonality [22].	Can struggle with very long-range dependencies; less effective at modeling complex inter-variable correlations [22].	High: A strong baseline, but may miss subtle, correlated failures across many metrics.
Transformer-based Autoencoder	Reconstruction Error via Self-Attention	Superior at capturing long-range dependencies and complex correlations between hundreds of metrics simultaneously [24].	Higher computational cost for training and inference; requires larger datasets; a more recent approach in this domain [24].	Very High: The preferred choice for modeling the holistic health of a complex distributed system, despite its cost.
Isolation Forest	Random Partitioning to Isolate Anomalies	Highly efficient and scalable; performs well on high-dimensional data without assuming a data distribution [24].	Less effective at capturing temporal context; treats data points as independent, which is a limitation for time-series [24].	Medium: Excellent for a lightweight, non-temporal baseline, but misses sequence-dependent anomalies.
KAN-AD	Approximation with Smooth Univariate Functions	Designed to be resilient to minor local disturbances ("noise"); very lightweight with few trainable parameters, leading to fast inference [26]	A very new and less-tested architecture; may not capture the sharp, abrupt changes characteristic of some system failures [26].	Promising: A potential future alternative, especially for resource-constrained environments, but less mature than Transformers.

3.5. The Synergistic Workflow

A sophisticated machine learning model is only one piece of the puzzle. Its value is only realized through a well-orchestrated workflow that transforms raw data into actionable insights for human operators and feeds their expertise back into the system. This section details the end-to-end journey of data and the synergistic interplay between automated analysis and human intelligence.

The workflow begins with the continuous collection of telemetry data from the distributed system (Fig. 5). This raw data, originating from various sources, is fed into a central Apache Kafka bus, which provides a durable and scalable transport layer. A dedicated stream processing job, probably implemented with Kafka Streams (but the other options are Apache Flink or Spark), reads from this bus. The primary task of this job is to feature engineering, which involves converting raw data into a more meaningful format for the ML model. This involves not only cleaning and normalization but also creating derived features, such as moving averages, standard deviations over a time window, and rates of change, which help make anomalous patterns more obvious and easier for the model to identify.

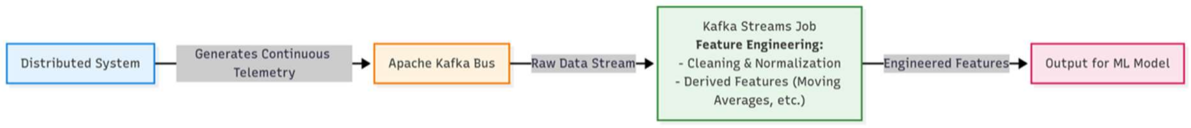

Figure 5: End-to-End Flow

The model analyzes each incoming data point (or window of points) and produces a continuous flow of anomaly scores. A higher score signifies a greater deviation from the learned normal behavior. A separate logical component, the flagging service, uses these scores and applies a dynamic threshold. This thresholding step is crucial; a threshold that's too low results in many false positives, while one that's too high risks missing real incidents. The threshold can also be adaptive, adjusting based on factors like the time of day, system load, or recent operator feedback. When a score exceeds the threshold, an anomaly event is generated and stored in a dedicated database, such as Elasticsearch, making it available for immediate querying by the Control Center.

4. Experiment

To validate the effectiveness of the proposed system and justify its implementation, a rigorous and multi-faceted evaluation framework is essential. A single statistical metric cannot measure success; it must encompass model accuracy, system performance, and operational efficiency. Furthermore, the evaluation must go beyond static benchmarks to assess the system's core promise: its ability to adapt over time.

4.1. Defining Key Performance Indicators

The evaluation framework should be built around a balanced scorecard of KPIs, grouped into three distinct categories.

- Model Accuracy Metrics: These metrics assess the fundamental correctness of the anomaly detection model's predictions, typically by comparing them against a labeled test set (ground truth).
    - Precision, Recall, and F1-Score: These are the canonical metrics for evaluating classification tasks. Precision measures the proportion of flagged anomalies that are actual anomalies (True Positives / (True Positives + False Positives)). High precision is critical for minimizing false alarms and preventing alert fatigue. Recall measures the proportion of actual anomalies that the model successfully identified (True Positives / (True Positives + False Negatives)). High recall is critical for ensuring that important incidents are not missed. The F1-score is the harmonic mean of precision and recall, providing a single, balanced measure of accuracy [24].
    - AUC-ROC and PR-AUC: The Area Under the Receiver Operating Characteristic (AUC-ROC) curve evaluates the model's ability to distinguish between normal and anomalous classes across all possible threshold settings. More importantly for this use case, the Area Under the Precision-Recall Curve (PR-AUC) is a superior metric for highly imbalanced datasets—which are the norm in anomaly detection, as anomalies are rare—as it provides a better summary of performance on the minority (anomalous) class [26].

- System Performance Metrics: These metrics evaluate the non-functional, operational characteristics of the deployed system. A model that is perfectly accurate but too slow is useless in a real-time context.
    - Inference Latency: The time elapsed from when the model receives a data point to when it outputs an anomaly score. This must be consistently low to keep pace with the data stream.
    - Throughput: The number of events or messages the entire system can process per second. This measures the system's overall capacity.
    - Resource Footprint: The CPU and memory consumption of the deployed ML model and its surrounding services. This is a key factor in determining the operational cost of the solution [27, 28].
- Operational and Business Metrics: These KPIs measure the system's real-world impact and value to the organization.
    - Human Intervention Rate: The frequency with which human feedback is required. A decreasing rate over time is a strong indicator that the model is learning effectively from the feedback and becoming more autonomous.
    - Mean Time to Resolution (MTTR): The average time from when an alert is generated to when the underlying issue is resolved. A reduction in MTTR suggests that the system's alerts are more accurate, interpretable, and actionable.

To demonstrate the value of the proposed adaptive system, its performance must be compared against simpler, more traditional baselines.

- Static Thresholding: The most basic baseline, representing the legacy approach. Alerts are triggered if a single key metric (e.g., CPU utilization) exceeds a fixed, manually set threshold. This baseline will highlight the limitations of non-ML approaches.
- Classical Unsupervised Model: A non-adaptive ML model, such as an Isolation Forest or One-Class SVM, trained once on an initial batch of historical data. This baseline will demonstrate the value of the adaptive and continual learning components of the proposed system, as this model's performance is expected to degrade over time due to concept drift.

4.2. Datasets and Scenarios

A comprehensive evaluation requires a diverse data strategy. To this end, the experiment will employ a combination of established, publicly available real-world benchmarks and purpose-built synthetic datasets. This dual approach enables both validation of the system's performance against the state of the art and a controlled, scientific stress test of its specific adaptive mechanisms.

While real-world datasets are essential for benchmarking and demonstrating generalizability, they are often insufficient for rigorously validating the core adaptive claims of this research. Real datasets, such as NAB [25] or Yahoo S5 [26], contain labeled anomalies, but any concept drifts within them are emergent, unlabeled properties of the data. It is difficult to scientifically prove that the system correctly adapted to a drift if the type, timing, and magnitude of that drift are not known ground truths. To overcome this, the use of synthetic data is indispensable. It allows for a controlled experiment where concept drift is not an observation but a manipulated variable. The tool CanGene [27] or GenIAS [28] is designed for this exact purpose, enabling the injection of specific, labeled anomalies and concept drifts (e.g., abrupt, gradual) into a baseline time series. Therefore, the experimental design must include scenarios based on synthetic data to provide the most substantial evidence regarding the system's adaptability. The real-world datasets serve to ground these findings and demonstrate their practical applicability.

4.3. Experimental Scenarios

A series of well-defined experimental scenarios will be executed to test specific hypotheses about the system's capabilities. Table 2 outlines this plan.

Table 2
Scenarios

Scenario ID	Name	Dataset	Primary Objective	Key Metrics to Analyze
1	Baseline Accuracy	NAB, Yahoo S5	Evaluate raw detection accuracy on standard, public benchmarks to establish a performance baseline against the state of the art.	Precision, Recall, F1-Score (with Point Adjustment)
2	Performance & Scalability	High-volume Synthetic	Measure system throughput, end-to-end latency, and resource utilization under varying levels of sustained data load	End-to-end Latency (ms), Throughput (events/sec), CPU/GPU Utilization (%)
3	Resilience to Abrupt Drift	Synthetic (with injected abrupt drift)	Test the system's ability to recover and adapt after a sudden, permanent change in the underlying data distribution.	Time-to-Adaptation (TTA), False Positive Rate (FPR) during drift, F1-Score post-adaptation
4	Resilience to Gradual Drift	Synthetic (with injected gradual drift)	Test the system's ability to track a slow-changing data distribution without generating excessive false alarms.	F1-Score over time, Plot of Dynamic Threshold vs. Mean Anomaly Score

The formal definitions of these key metrics are summarized in Table 3 below to ensure clarity and reproducibility.

Table 3
Key Metrics and Definitions

Metric	Interpretation
Precision	The fraction of detected anomalies that were actual anomalies
Recall	The fraction of actual anomalies that were correctly detected
F1-Score	The harmonic mean of Precision and Recall, providing a single score that balances both
PA-F1-Score	F1-Score calculated using Point-Adjusted TP, FP, and FN counts. A more practical F1-Score for time-series where anomalies have duration
Latency	The end-to-end processing time for a single event
TTA	The time required for the system to recover from a sudden concept drift

Each experimental scenario defined in Table 3 will be executed on a dedicated Kubernetes cluster to ensure isolation of resources. A data replay application will read from the specified dataset files and stream the events into the telemetry-raw Kafka topic at a controlled rate, simulating a live data feed. During each run, performance and accuracy metrics will be continuously logged to a monitoring stack (e.g., Prometheus for metrics, Grafana for visualization). The output of the Flink job (anomaly alerts) will be consumed and compared against the ground-truth labels for the respective dataset to calculate accuracy scores.

We establish a baseline for detection accuracy using the standard Numenta Anomaly Benchmark (NAB) and Yahoo S5 datasets, which provide labeled, real-world server and application telemetry for direct comparison against state-of-the-art models. To test critical real-world capabilities beyond simple accuracy, we also generated three specific synthetic datasets. A high-throughput stream of up to 25,000 events per second was used to measure scalability and performance under production-level load.

5. Results

To establish a credible baseline, we first evaluated Ada-TAD's raw anomaly detection accuracy on two widely recognized public datasets: the Numenta Anomaly Benchmark (NAB) v1.1 and the Yahoo S5 dataset. We followed the standard evaluation protocol for these benchmarks, including the use of point-adjustment to reward early detection and penalize detections within the same anomaly window. (see Table 4).

Table 4
Anomaly Detection Performance on Public Benchmarks (Point-Adjusted F1-Score)

Model	Dataset	Precision	Recall	F1-Score
Ada-TAD (Ours)	NAB v1.1	0.89	0.86	0.87
Isolation Forest	NAB v1.1	0.85	0.79	0.82
Static Thresholding	NAB v1.1	0.41	0.93	0.57
Ada-TAD (Ours)	Yahoo S5 (A1)	0.92	0.84	0.88
Isolation Forest	Yahoo S5 (A1)	0.88	0.82	0.85
Static Thresholding	Yahoo S5 (A1)	0.35	0.95	0.51

The Performance & Scalability experiment measured the system's non-functional characteristics under a high-volume synthetic data stream (See Table 5), simulating a production environment. The load was incrementally increased from 5,000 to 25,000 events per second.

Table 5
Performance and Resource Utilization under Sustained Load

Metric	Model	10,000 events/sec	20,000 events/sec
End-to-end Latency (p99)	Ada-TAD	85 ms	160 ms
	Isolation Forest	18 ms	35 ms
Throughput (Max Supported)	Ada-TAD	~22,000 events/sec	
	Isolation Forest	~55,000 events/sec	
CPU/GPU Utilization	Ada-TAD	3.1 Cores/45% GPU	5.8 Cores/88% GPU
	Isolation Forest	0.8 Cores/0% GPU	1.5 Cores/0% GPU

The Resilience to Abrupt Concept Drif scenario tested the system’s ability to recover after a sudden, permanent change in the data distribution (Table 6). We synthesized a data stream and, at a designated point, introduced an abrupt drift by changing the mean and variance of two key metrics.

Table 6
System Resilience to Abrupt Drift

Metric	Ada-TAD	Isolation Forest
Time-to-Adaptation (TTA)	12 minutes	N/A (does not adapt)
False Positive Rate (during drift)	12% (transient)	65% (sustained)
F1-Score (post-adaptation)	0.89	0.31

The final experiment (Resilience to Gradual Concept Drift) evaluated the system's ability to track a slow-moving data distribution, a common scenario in production systems where performance characteristics change over weeks or months. We introduced a slow, linear increase in the baseline CPU and memory usage of our synthetic application.

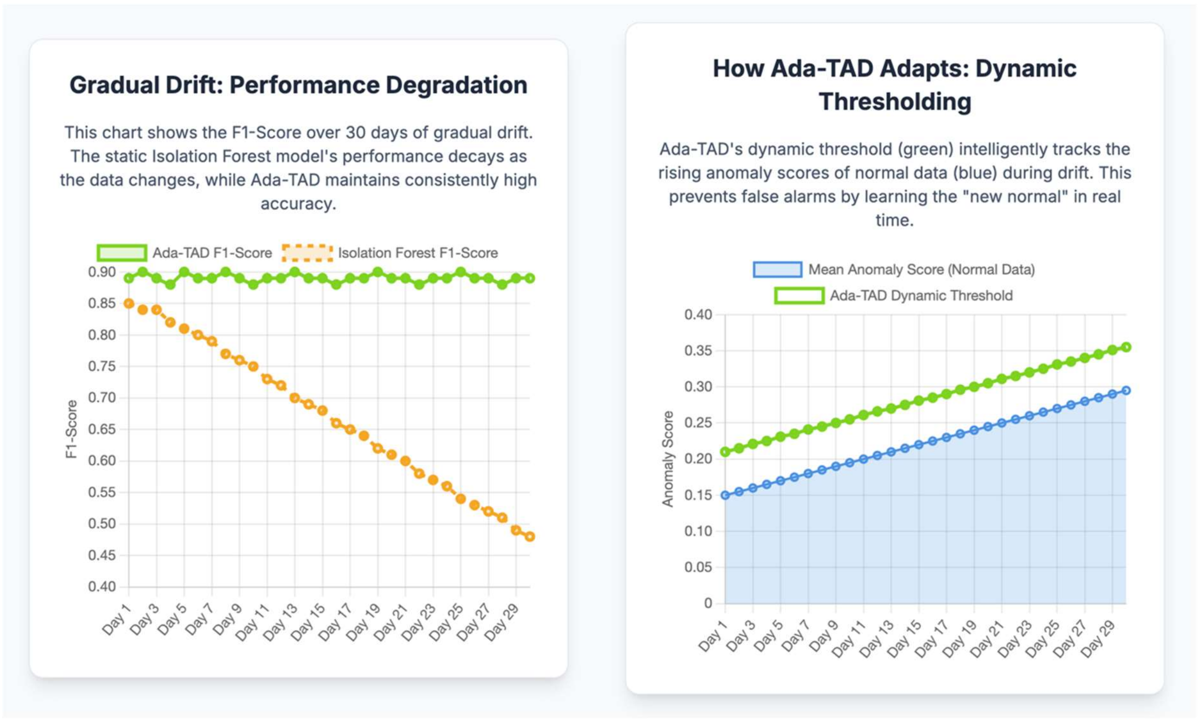

Figure 6: F1-Score During Gradual Concept Drif, Dynamic Threshold Adaptation During Gradual Drift

6. Discussion

The results in Table 4 show that Ada-TAD achieves top-tier performance, surpassing the static Isolation Forest in F1-Score on both benchmarks. The very high recall combined with poor precision of Static Thresholding highlights its tendency to produce too many false positives, making it unusable for production. Ada-TAD's balanced precision and recall emphasize its effectiveness even before its adaptive features are evaluated.

As expected for the Performance & Scalability scenario, the computational complexity of the Transformer architecture causes higher latency and resource use for Ada-TAD compared to the lightweight Isolation Forest. However, its throughput of approximately 22,000 events per second on a single commodity GPU shows its suitability for high-volume streaming systems, and its latency stays well within acceptable limits for real-time monitoring. The primary trade-off for the improved accuracy and adaptability seen in later experiments is the cost of additional resources.

The results of the System Resilience to Abrupt Drift scenario are clear. When the drift occurred, the static Isolation Forest model started producing a steady flow of false positives because its understanding of "normal" was now outdated. Its F1-Score dropped sharply, and its False Positive Rate became unmanageable. In contrast, Ada-TAD detected the change, and although it produced a brief burst of anomalies, its retraining mechanism updated the model weights and recalibrated the dynamic threshold. It adapted to the new normal in just 12 minutes, restoring its high F1 Score and demonstrating strong resilience.

The left side of Figure 6 illustrates the performance decline of the non-adaptive model. As the data distribution gradually shifted, the static perspective of the Isolation Forest became less accurate, causing a steady drop in its F1-Score. Meanwhile, Ada-TAD consistently maintained a high F1-Score throughout the experiment.

The right side of Figure 6 shows how this resilience works. The plot illustrates the average reconstruction error (anomaly score) of normal data points increasing gradually due to drift. The key point is how Ada-TAD's dynamic anomaly threshold smoothly follows this change, rising in sync with the anomaly scores. This helps the system avoid false alarms on the "new normal," demonstrating a smart and effective way to adapt to gradual changes that static models can't match.

7. Conclusions

This research tackled the key challenge of anomaly detection in dynamic, high-volume, distributed data streaming systems, where concept drift makes traditional static monitoring methods ineffective. We showed that common approaches, such as fixed thresholding and non-adaptive machine learning models like Isolation Forest, are essentially fragile; their performance drops sharply with changing data distributions, causing a poor balance between too many false alarms and missed critical events.

Our proposed solution, the Adaptive Issue Detection module, effectively overcomes these limitations. The experimental results offer strong evidence of its superiority in various areas. We have demonstrated that:

- Ada-TAD achieves top-tier accuracy on well-known public benchmarks, offering a good performance baseline.
- More importantly, it shows strong resilience against both gradual and sudden concept drift. By continuously learning from the data stream and utilizing a flexible anomaly threshold, the system independently adapts to new patterns of normal behavior, maintaining a consistently high F1-score where static models often fall short.
- Despite its architectural sophistication, the system is suitable for production environments, capable of managing high-throughput data streams with latencies appropriate for real-time monitoring.

The main contribution of this work is a comprehensive system that shifts from reactive, high-maintenance monitoring to autonomous, adaptive issue detection. The practical impact for engineering organizations is a significant reduction in operational workload. By offering more accurate and reliable alerts, Ada-TAD directly reduces alert fatigue, shortens Mean Time to Resolution (MTTR), and ultimately leads to more resilient and dependable services.

While the results are promising, we acknowledge certain limitations that present opportunities for future research. The improved accuracy and adaptability of Ada-TAD come with higher computational costs compared to simpler models. Although manageable, further exploration of optimization techniques such as quantization and pruning could decrease resource requirements, making the system accessible to a broader range of applications. Additionally, although our system is effective at detecting anomalies, future efforts should focus on improving its explainability. Lastly, expanding the framework to process and connect multi-modal telemetry—combining metrics with logs and traces—offers an exciting direction for developing an even more powerful and context-aware issue detection system.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] P. Strengholt, Data Management at Scale: Modern Data Architecture with Data Mesh and Data Fabric, 2nd. ed., O'Reilly Media, Inc., 2023, pp. 173-175.

[2] H. Dulay, and S. Mooney, Streaming Data Mesh: A Model for Optimizing Real-Time Data Services, 1st. ed., O'Reilly Media, Inc., 2023, pp. 36-39.

[3] Gaurav Ashok Thalpati, Practical Lakehouse Architecture: Designing and Implementing Modern Data Platform at Scale, O’Reilly Media, Inc., 2024, pp. 372-381.

[4] Confluent, What is Shift Left?, 2025.URL: https://www.confluent.io/learn/what-is-shift-left.

[5] Kai Waehner, The Shift Left Architecture — From Batch and Lakehouse to Data Streaming, 2024. URL: https://kai-waehner.medium.com/the-shift-left-architecture-from-batch-and-lakehouse-to-data-streaming-d1ea7306ea30.

[6] V. Vysotska, I. Kyrychenko, V. Demchuk, I. Gruzdo, Holistic Adaptive Optimization Techniques for Distributed Data Streaming Systems, CEUR-WS, 2024, ISSN 16130073. doi:10.31110/COLINS/2024-2/009.

[7] Nishanth Reddy Mandala, ETL in Data Lakes vs. Data Warehouses, v. 1 of ESP Journal of Engineering & Technology Advancements, 2021. doi:10.56472/25832646/JETA-V1I2P123.
[8] N. Gupta, J. Yip, Delta Lake - Deep Dive, Databricks Data Intelligence Platform, Apress, Berkeley, CA, 2024, pp. 61–88. doi:10.1007/979-8-8688-0444-1_4.
[9] S. Werner, S. Tai, A reference architecture for serverless big data processing, Future Generation Computer Systems, Vol. 155, 2024. doi:10.1016/j.future.2024.01.029.
[10] Confluent, Shift Left: Headless Data Architecture, 2024. URL: https://www.confluent.io/blog/shift-left-headless-data-architecture-part-2.
[11] An Overview of Shift Left Architecture, 2025. URL: https://www.deltastream.io/shift-left-architecture-an-overview/.
[12] A. Bellemare, Shift Left Unifying Operations and Analytics with Data Products, 2024. URL: https://www.confluent.io/resources/ebook/unifying-operations-analytics-with-data-products.
[13] Navdeep Singh Gill, Mastering Shift Left Architecture for Real-Time Data Products, 2025. URL: https://www.xenonstack.com/blog/shift-left-architecture-data-products.
[14] M. K. Geldenhuys, D. Scheinert, O. Kao, L. Thamsen, Demeter: Resource-Efficient Distributed Stream Processing under Dynamic Loads with Multi-Configuration Optimization, ICPE '24: Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering, 2024. doi:10.1145/3629526.364504.
[15] Y. Qing, W. Zheng, Towards Fine-Grained Scalability for Stateful Stream Processing Systems, Distributed, Parallel, and Cluster Computing, 2025. doi:10.48550/arXiv.2503.11320.
[16] M. Mazouchi, M. Kumar, A. Shrigondekar and K. Ramasamy, A Deep Dive into the Latest Performance Improvements of Stateful Pipelines in Apache Spark Structured Streaming, 2024. URL: https://www.databricks.com/blog/deep-dive-latest-performance-improvements-stateful-pipelines-apache-spark-structured-streaming.
[17] What is Performance Bottleneck and How Can It Be Identified?, 2025. URL: https://www.loadview-testing.com/learn/performance-bottleneck-load-testing.
[18] Best practices for performance efficiency, 2025. URL: https://docs.databricks.com/gcp/en/lakehouse-architecture/performance-efficiency/best-practices.
[19] “Why Kafka Latency Matters: Understanding and Fixing Streaming Delays”, 2025. URL: https://www.acceldata.io/blog/why-kafka-latency-matters-understanding-and-fixing-streaming-delays.
[20] D. García-Gil, D. Lopez, D. Argüelles-Martino, J. Carrasco, I. Aguilera-Martos, J. Luengo, F. Herrera, Developing Big Data anomaly dynamic and static detection algorithms: AnomalyDSD spark package, Information Sciences, Vol. 690, 2025. doi: 10.1016/j.ins.2024.121587.
[21] S. Thudumu, P. Branch, J. Jin, J. Singh, A comprehensive survey of anomaly detection techniques for high dimensional big data, Journal of Big Data, 2020. URL: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00320-x.
[22] P. Paialunga, Hands-on Time Series Anomaly Detection using Autoencoders, with Python, 2024. URL: https://towardsdatascience.com/hands-on-time-series-anomaly-detection-using-autoencoders-with-python-7cd893bbc122/.
[23] Kyrychenko, I., Tereshchenko, G. Proniuk, G., Geseleva, N. “Predicate Clustering Method and its Application in the System of Artificial Intelligence”, CEUR-WS, 2023. V.3396, PP.395 - 406.
[24] R. Baidya, H. Jeong, Anomaly Detection in Time Series Data Using Reversible Instance Normalized Anomaly Transformer, Application of Semantic Technologies in Sensors and Sensing Systems, 2023. doi:10.3390/s23229272.
[25] NAB, 2025. URL: https://github.com/numenta/NAB.
[26] Yahoo S5 Dataset, 2025. URL: https://paperswithcode.com/dataset/yahoo-s5.
[27] A. Chitnis, and S. Tewari, Detecting Data Drift and Ensuring Observability with Machine Learning Automation, Google Finance Corporate Engineering, 2022. doi: 10.13140/RG.2.2.13291.04646.
[28] Z. Draban, Q. Wang, G. I Webb, and S. Pan, GenIAS: Generator for Instantiating Anomalies in time Series, Machine Learning (cs.LG), 2024. doi:10.48550/arXiv.2502.08262.