



Real-Time Anomaly Detection in Cybersecurity Using Deep Learning and Stream Analytics

¹Venkata Subbareddy Pallamreddy and ²Dr. Sunil Kumar Verma

¹Research Scholar, Department of Computer Science & Engineering, Monad University, Hapur, Uttar Pradesh, India

²Professor, Department of Computer Science & Engineering, Monad University, Hapur, Uttar Pradesh, India

DOI: <https://doi.org/10.5281/zenodo.23609628>

Corresponding Author: Venkata Subbareddy Pallamreddy

Abstract

The rapid growth of cloud computing, connected devices, remote work, and software-defined infrastructure has expanded the scale and complexity of cyber threats. Conventional signature-based security controls are effective against known attack patterns but may struggle when attacks are novel, modified, low-and-slow, or hidden within legitimate activity. Anomaly detection offers a complementary approach by identifying deviations from learned patterns of normal system or network behaviour. In operational environments, however, labelled attack data are often limited, imbalanced, costly to obtain, and quickly outdated. Semi-supervised machine learning is therefore a promising strategy: it learns primarily from abundant unlabelled observations while using a smaller labelled set, or an assumption that most training data represent normal behaviour, to distinguish suspicious activity. This paper presents a research framework for real-time cybersecurity anomaly detection using semi-supervised machine learning. It reviews major anomaly-detection paradigms, discusses candidate models including one-class support vector machines, isolation-based methods, autoencoders, and teacher-student approaches, and proposes a streaming architecture that integrates feature extraction, online scoring, threshold calibration, alert prioritisation, and analyst feedback. Evaluation is framed around detection quality, false-alarm burden, latency, throughput, and robustness to concept drift. The paper also addresses data leakage, class imbalance, adversarial adaptation, privacy, explainability, and deployment governance. Rather than claiming unperformed experiments, it provides a reproducible evaluation plan and hypotheses that can be tested on public intrusion-detection datasets and, where possible, operational telemetry. The central argument is that practical performance depends not only on model choice but also on data quality, calibration, stream processing, and a disciplined human-in-the-loop response process.

Keywords: Cybersecurity, anomaly detection, semi-supervised learning, intrusion detection, streaming analytics, concept drift, autoencoder, false-positive reduction

Introduction

Modern cybersecurity monitoring requires continuous analysis of events from networks, endpoints, identity systems, cloud services, applications, and security devices. The diversity and velocity of these streams make it difficult to depend only on manual investigation or static signatures. Real-time anomaly detection attempts to recognize deviations while events are being generated. Deep learning is relevant because complex attacks may be represented through nonlinear relationships and temporal sequences that are difficult to encode as simple rules.

This research paper focuses on a distinct perspective: temporal modelling, layered detection, ensemble scoring, and scalable stream analytics. The emphasis is on how a detection architecture can remain useful when individual

events appear legitimate but their sequence becomes unusual.

Temporal Nature of Attacks

Many attacks develop over a sequence of actions. An isolated successful login may be normal, and a single connection may also be normal, but an unusual combination of authentication, resource access, privilege change, and outbound communication may indicate a developing incident.

Temporal anomaly detection includes previous observations, rolling statistics, event frequency, inter-arrival times, session duration, and changes from established baselines. This context can improve discrimination between isolated unusual events and coordinated behaviour.

The challenge is balancing context and speed. Longer observation windows provide more information but can increase detection delay. Short windows improve responsiveness but may miss slow or distributed patterns.

Layered Detection

A layered architecture can combine fast statistical screening with deeper analysis. The first layer evaluates every event using inexpensive indicators. A second layer applies more computationally demanding deep learning to selected windows. A third layer correlates related anomalies and assigns risk.

This approach can reduce computational cost because complex models do not have to process every low-risk event. It can also provide a useful explanation path: a preliminary deviation triggered deeper analysis, which then found additional evidence.

The architecture must nevertheless be calibrated carefully. Combining several model scores without normalization can create unstable decisions, and additional layers increase maintenance requirements.

Significance

Real-time detection has social and organizational significance because cyber incidents can disrupt services and compromise information. Faster recognition can support containment and reduce the period during which an attacker can operate.

Scalable analytics can also help security teams handle event volumes that would otherwise exceed human capacity. The objective is not full automation of security decisions but intelligent prioritization of human attention.

The research therefore considers technical accuracy together with latency, throughput, alert volume, explainability, and governance.

Review of Literature

Anomaly detection has evolved from statistical profiling to machine learning and deep learning. Statistical approaches provide understandable baselines, while machine learning can discover more complex patterns. Deep learning increases representational capacity but introduces computational and interpretive challenges.

Supervised learning is effective when labels are complete and representative. In cybersecurity, this assumption is often unrealistic because new attacks appear continuously and many events remain unlabeled.

Unsupervised and semi-supervised methods address label scarcity, while temporal models add contextual information. The combination is promising for real-time environments but requires careful system design.

Temporal Deep Learning

Recurrent neural networks can represent event sequences through an internal state. LSTM and GRU architectures were developed to preserve useful information over longer sequences. In cybersecurity, such models can represent authentication, process, or connection histories.

Temporal convolutional networks provide another approach using convolution over time windows. Attention-based models can capture relationships among events without relying solely on sequential recurrence.

Architecture choice should reflect the data rate and operational constraints. A complex model that cannot process events within the required latency window may not be suitable despite strong offline accuracy.

Autoencoders and Reconstruction

Autoencoders learn to reconstruct input observations from a compressed representation. When trained primarily on normal behaviour, reconstruction error can act as an anomaly score.

Reconstruction-based methods are useful when malicious examples are scarce. They can detect deviations without requiring every attack type to be represented during training. However, an autoencoder may also reconstruct unusual but legitimate events poorly. Contextual information, threshold calibration, and additional detection layers are therefore important.

Ensemble Detection

Ensemble detection combines signals from multiple models or analytical methods. A security pipeline can combine statistical deviation, reconstruction error, isolation-based scoring, and supervised probabilities.

Ensembles may improve robustness because different detectors fail in different ways. A statistical detector may identify volume anomalies while a temporal model identifies unusual sequences.

The limitation is complexity. Scores must be calibrated, models must be versioned, and failure of one component should not silently invalidate the complete system.

Stream Analytics

Streaming systems process events continuously through ingestion, transformation, aggregation, model inference, and alert generation. They must handle bursts, duplicate messages, missing fields, and events that arrive out of order. Windowing determines the context available to the detector. Sliding windows preserve recent history, tumbling windows simplify aggregation, and session windows can follow user or connection activity.

Scalability depends on feature computation as well as model inference. Rolling statistics and frequency features can become expensive at high event rates, making distributed processing or approximation useful.

Explainability and Human Factors

Security analysts need actionable alerts rather than unexplained anomaly scores. Explanations can identify contributing features, related events, baseline comparisons, or representative normal behaviour.

Alert fatigue is a major operational concern. Excessive low-value notifications can cause important alerts to receive less attention. Correlation and prioritization can reduce this problem.

Analyst feedback is valuable for model improvement, but feedback should be validated before automated learning uses it. Otherwise, an incorrect analyst label could influence future detections.

Research Gap

Many studies demonstrate temporal or deep learning models on benchmark datasets but provide limited evidence about

sustained high-volume operation. Production environments require additional evaluation of throughput, queueing, model maintenance, and alert workload.

A second gap is integration. Fast screening, temporal analysis, explainability, and incident correlation are often treated as separate problems. A layered architecture provides a way to connect them.

The research gap addressed here is therefore the need for a practical framework that evaluates deep anomaly detection as a complete streaming system rather than only as a classification model.

Objectives

1. To examine the effectiveness of deep learning and temporal modelling for identifying anomalous patterns in continuously generated cybersecurity data.
2. To evaluate a layered detection strategy using detection quality, false-positive control, processing latency, throughput, and alert prioritization measures.
3. To identify practical requirements and research gaps for scalable, explainable, and continuously monitored real-time cybersecurity anomaly detection.

Research Methodology

The research uses a quantitative experimental framework. The workflow includes data ingestion, normalization, time-window construction, feature extraction, deep representation learning, ensemble scoring, alert correlation, and evaluation. Possible inputs include network flows, DNS activity, authentication events, endpoint processes, and application logs. Each source is mapped to a common event representation containing time, context, event type, and quantitative attributes.

Preprocessing includes timestamp normalization, missing-value handling, categorical encoding, numerical scaling, duplicate removal, and temporal feature construction.

Model Architecture

The proposed architecture contains three layers. Layer one performs rapid screening with lightweight statistical indicators. Layer two applies a temporal deep model to selected windows. Layer three correlates related events and produces an operational risk score.

The deep model can combine an encoder-decoder representation with a temporal component. Reconstruction or prediction error provides an anomaly signal. The model can learn from normal sequences while a validation set determines the operational threshold.

Ensemble scoring combines normalized outputs from the statistical detector and deep model. Optional supervised information can refine the score where reliable labels are available.

Research Methodology - Data Splitting

Data are divided chronologically into training, validation, and test periods. Training develops representations, validation supports threshold and architecture selection, and testing provides the final estimate of performance.

Chronological separation reduces temporal leakage. Random splitting can place nearly identical adjacent events in training and testing, creating an unrealistic evaluation.

Results should be reported overall and, where labels permit,

by attack category and time period. This helps identify performance degradation and distribution changes.

Streaming Implementation

A practical implementation can use an event queue or stream broker to transport telemetry into a processing layer. Feature extraction is performed on incoming records or windows, after which the model service returns anomaly scores.

Back-pressure is important. When event arrival exceeds processing capacity, queues grow and detection becomes delayed. Lightweight screening, prioritization, horizontal scaling, and efficient inference can reduce this risk.

Monitoring should cover event loss, queue depth, feature quality, model latency, score distribution, alert rate, and system availability.

Robustness and Governance

Attackers may attempt to evade detection by producing behaviour close to the normal baseline or by generating large volumes of distracting events. Robustness testing should therefore include distribution shifts and controlled perturbations.

Model governance includes version control, approved deployment procedures, audit logs, documented thresholds, and rollback mechanisms. A production model should not change silently.

Explainability records can include important features, related events, baseline comparisons, and model version. These records support investigation and reproducibility.

Limitation of Research work (Research Gap)

Public datasets may not reflect the precise traffic distribution of a modern cloud or enterprise environment. Results should therefore be interpreted according to the characteristics of the selected dataset.

Temporal models depend on accurate event ordering and sufficient sequence context. Missing or delayed events can weaken sequence representations.

Deep learning may require substantial computational resources. Model compression, optimized inference, or distributed deployment may be required for high event rates. The principal gap is limited integration of temporal deep learning, stream processing, explainability, and operational alert management within a single evaluation framework.

Results and Discussion

The layered approach is expected to improve operational efficiency by applying computationally inexpensive screening before deeper analysis. High-risk windows receive additional modelling, while routine observations require less processing.

Temporal context should be most useful when anomalies develop through a sequence. The important question is whether sequence information improves recall without creating excessive false positives or latency.

Results should distinguish event-level detection from incident-level performance because several event alerts can correspond to one underlying security incident.

Evaluation Framework

Actual numerical values should come from the executed

experiment or selected dataset. The table below defines the recommended reporting structure and avoids unsupported numerical claims.

Interpretation should focus on trade-offs. An increase in recall is not automatically beneficial if alert volume or processing delay increases substantially.

Operational measures should be reported together with classification measures so that the detector can be judged as a complete system.

Table 1: Operational measures should be reported

Dimension	Measure	Purpose
Detection	Precision / Recall	Alert quality and coverage
Balance	F1 / PR-AUC	Rare-event performance
Speed	Latency	Real-time suitability
Scale	Throughput	Processing capacity
Operations	Alert rate	Analyst workload

Temporal Behaviour

Temporal modelling can identify sequences in which individual actions are not obviously suspicious. Examples include unusual authentication sequences, unexpected resource access after privilege changes, and abnormal connection chains.

The benefit depends on the quality of the time window. Short windows provide faster decisions, while longer windows provide more context. The best setting must be established empirically.

Time-based evaluation should also consider changes in the underlying environment. A detector that performs well during one period may degrade after application or user-behaviour changes.

False Positives and Alert Reduction

Legitimate rare events can appear anomalous. Context such as asset criticality, user role, historical behaviour, and maintenance schedules can improve alert prioritization.

Correlation can combine multiple low-confidence anomalies into a higher-confidence incident. Conversely, repeated identical events can be grouped to reduce redundant notifications.

Alert reduction should not be treated as an unconditional objective. Excessive suppression can hide evidence of a real attack. The goal is to reduce low-value duplication while preserving investigative context.

Operational Interpretation

A real-time detector should be evaluated end to end. Ingestion, feature calculation, model inference, correlation, and delivery all contribute to final latency.

System monitoring should record queue depth, processing time, event loss, model response time, and availability. Traceability from source event to alert is important for debugging and incident investigation.

The results therefore support a broader view of anomaly detection: deep learning is most useful when it is integrated into reliable data engineering, monitoring, and human-centered alert management.

Conclusions

Real-time cybersecurity anomaly detection can benefit from temporal deep learning and layered stream analytics. Deep

models can represent complex behavioural relationships while lightweight screening controls computational cost.

A layered architecture can support fast detection, contextual analysis, event correlation, and risk-based prioritization. This is particularly useful when individual events appear normal but their sequence is unusual.

Future research should evaluate diverse time-separated datasets, efficient online adaptation, explainability, and the impact of detection on analyst workload and response time.

References

1. Chandola V, Banerjee A, Kumar V. Anomaly detection: a survey. *ACM Computing Surveys*. 2009;41(3):1-58.
2. Ahmed M, Mahmood AN, Hu J. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*. 2016;60:19-31.
3. Buczak AL, Guven E. A survey of data mining and machine learning methods for cybersecurity intrusion detection. *IEEE Communications Surveys & Tutorials*. 2016;18(2):1153-1176.
4. Shone N, Ngoc TN, Phai VD, Shi Q. A deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*. 2018;2(1):41-50.
5. Mirsky Y, Doitsman T, Elovici Y, Shabtai A. Kitsune: an ensemble of autoencoders for online network intrusion detection. In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. c2018.
6. Hochreiter S, Schmidhuber J. Long short-term memory. *Neural Computation*. 1997;9(8):1735-1780.
7. Vaswani A, *et al*. Attention is all you need. In: *Advances in Neural Information Processing Systems*. c2017.
8. Ring M, Wunderlich S, Grödl D, Landes D, Hotho A. A survey of network-based intrusion detection data sets. *Computers & Security*. 2019;86:147-167.
9. Moustafa N, Slay J. UNSW-NB15: a comprehensive data set for network intrusion detection systems. 2015.
10. Sharafaldin I, Lashkari AH, Ghorbani AA. Toward generating a new intrusion detection dataset and intrusion traffic characterization. 2018.

Creative Commons (CC) License

This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY 4.0) license. This license permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.