Neural Computing and Applications
https://doi.org/10.1007/s00521-024-09863-z

ORIGINAL ARTICLE

Network security AIOps for online stream data monitoring

Giang Nguyen[1,2] · Stefan Dlugolinsky[2] · Viet Tran[2] · Álvaro López García[3]

Received: 13 November 2023 / Accepted: 12 April 2024

Abstract
In cybersecurity, live production data for predictive analysis pose a significant challenge due to the inherently secure nature of the domain. Although there are publicly available, synthesized, and artificially generated datasets, authentic scenarios are rarely encountered. For anomaly-based detection, the dynamic definition of thresholds has gained importance and attention in detecting abnormalities and preventing malicious activities. Unlike conventional threshold-based methods, deep learning data modeling provides a more nuanced perspective on network monitoring. This enables security systems to continually refine and adapt to the evolving situation in streaming data online, which is also our goal. Furthermore, our work in this paper contributes significantly to AIOps research, particularly through the deployment of our intelligent module that cooperates within a monitoring system in production. Our work addresses a crucial gap in the security research landscape toward more practical and effective secure strategies.

Keywords AIOps · Network security · Neural networks · Online stream data · Unsupervised learning

Abbreviations

ACL	Access control lists
AED	Autoencoder
AI	Artificial intelligence
AIOps	AI for IT operations
API	Application programming interface
ARIMA	Auto-regressive integrated moving average
BLAST	The best last method
CNN	Convolutional neural networks
DBM	Deep Boltzmann machines
DBN	Deep belief networks
DL	Deep learning
DNN	Deep neural networks
DT	Decision tree
DWM	Dynamic weighted majority
ELK	Elasticsearch-Logstash-Kibana
ELM	Extreme learning machines
GRU	Gated recurrent unit
HTTP	Hypertext transfer protocol
ICMP	Internet control message protocol
IDS	Intrusion detection systems
IPS	Intrusion prevention systems
IT	Information technology
JSON	JavaScript object notation
kNN	K-nearest neighbor
LB	Leveraged bagging
LR	Logistic regression
LSTM	Long short-term memory
MAE	Mean absolute error
MAPE	Mean absolute percentage error
ML	Machine learning
MLP	Multilayer perceptron
MSE	Mean squared error
NAB	Numenta anomaly benchmark
NB	Naïve Bayes
NBA	Network behavior analysis
NN	Neural networks
OS	Operating system
PCA	Principal component analysis

✉ Giang Nguyen
giang.nguyen@stuba.sk

Stefan Dlugolinsky
stefan.dlugolinsky@savba.sk

Viet Tran
viet.tran@savba.sk

Álvaro López García
aloga@ifca.unican.es

[1] Faculty of Informatics and Information Technologies, Slovak University of Technology, Ilkovičova 2, 84216 Bratislava, Slovakia

[2] Institute of Informatics, Slovak Academy of Sciences, Dúbravská cesta 9, 84507 Bratislava, Slovakia

[3] Instituto de Física de Cantabria (IFCA), CSIC-UC, Avda. los Castros s/n, 39005 Santander, Cantabria, Spain

Published online: 11 May 2024

PL	Paired learner
RBM	Restricted Boltzmann machines
REST	Representational state transfer
RF	Random forest
RMSE	Root mean square error
RNN	Recurrent neural network
SABLE	Simple adaptive batch learning ensemble
SMAPE	Symmetric mean absolute percentage error
SOC	Security operations center
SVM	Support vector machine
TCP	Transmission control protocol
UDP	User datagram protocol
VAED	Variational Autoencoders
XML	Extensible markup language
YAML	YAML Ain't markup language

1 Introduction

An essential component of an organization's information technology (IT) infrastructure is the Security Operations Center (SOC), which provides a centralized unit that employs people, processes and technology to continuously monitor and manage the organization's security assessment. One of its key functions is to identify, investigate, and resolve cyber threats and attacks. This is achieved through active monitoring, which involves a continuous scan of the network for abnormalities or suspicious activities. The SOC collects and analyzes data from various sources, such as security devices, logs, and other data sources, to detect and respond to potential security incidents. By providing continuous monitoring and response capabilities, SOC helps organizations maintain their security posture and protect against cyber threats.

In network monitoring, situational awareness refers to best-practice engineering approaches used to continuously monitor IT processes and operations to improve service provision in the form of automatic action recommendation and decision support [1]. It is a challenge to deal with large volumes of data online and in real time. Many of such data are not labeled [2] or are not significant for detection purposes. Another challenge presents large quantities of alerts whose evaluation consumes unnecessary time of security analysts and thus makes the notification of real alerts more difficult. Various tools such as firewalls, virtual private networks, intrusion detection systems (IDSs), and intrusion prevention systems (IPSs) are used to ensure network security. Early detection, alerting, and rapid response are crucial components of a proactive security approach, helping prevent potential problems from impacting the network. Other challenges [3] in detecting network anomalies are a dynamic and fast changing environment, the diversity of resources and tools, and the security nature of the domain that prevents the sharing of logs among SOCs.

Artificial intelligence (AI), specifically its subfield neural networks (NNs) and deep learning (DL) in unsupervised way, aids in this process. This involves processing telemetry data from a monitored network, representing them as time sequences, and training an intelligent model to predict the most probable future states (called *baseline*) based on previous observations [4, 5]. In typical network operation, alarms are triggered when a monitored value exceeds or falls below predefined thresholds. However, setting these thresholds accurately can be challenging over time [6, 7]. If the thresholds are too narrow, a large number of alarms can be triggered, exceeding the analysis capacity of network managers. If the thresholds are too wide, real anomalies can be overlooked, leaving the network vulnerable to potential threats. Then, dynamically defining thresholds are becoming increasingly important. Such a nuanced soft baseline enables a complex system overview and helps to indicate possible anomalies, allowing security systems to continuously refine and adapt to changing situations over time and online. However, real data in real production are hard to obtain due to the security nature of the domain. There are a number of public, synthesized, and artificially generated datasets, but real scenarios are rarely seen.

The motivation of this work is to deploy an AI module for proactive network monitoring in real time to improve security protection. In this field, the implementation aspect of AI deployment has not yet been thoroughly investigated. By cooperating with IDS/IPS, the intelligent AI module is designed to monitor the infrastructure network in production and identify potential threats. To achieve this, the latest trend in multihorizon and multivariate time-series prediction is used, along with DL techniques in an unsupervised way, to simultaneously model multiple monitoring channels. This approach is expected to provide a more comprehensive and accurate view of the network's state, enabling proactive measures to be taken to improve network security.

The work presented in this paper makes several key contributions to the field of AI for IT operations (AIOps) research, specifically in the context of online data stream monitoring, including:

- The development of a DL-based multihorizon forecast model that simultaneously models multiple monitoring channels in an unsupervised manner, enabling a comprehensive view of network activities.

- The deployment of an AIOps module containing the DL-based trained model for proactive network security monitoring that cooperates in real-time with IDS in production and allows a dynamic nuanced baseline to improve the effectiveness of threat detection.
- The automated development for the entire software stack using declarative composition, which helps to simplify AIOps deployment and management over time.

These contributions demonstrate the potential for AI and advanced modeling techniques presented in this work to improve network security and improve the efficiency of IT operations.

The paper is further structured as follows: Section 2 surveys the related work. Section 3 provides the main idea about the cooperation architecture in Sect. 3.1, the learning development phase in Sect. 3.2, the online model deployment in Sect. 3.3, and the online anomaly detection in Sect. 3.4. Section 4 goes through the deployment of the architecture in Sect. 4.1, online data processing in Sect. 4.2, the quality forecast in Sect. 4.3, and anomaly detection results in Sect. 4.4. Section 5 concludes the main points of the work and its future extension.

2 Related work

2.1 Network behavior analysis

Network anomaly and detection, real-time monitoring, behavior baseline, analysis dashboards, network security measures, and access to threat intelligence are essential components of the network security strategy. By implementing these measures, organizations ensure that their networks are secure and protected against cyber threats. There are several types of IDS, including network IDS, host IDS, wireless IDS, and their combination. IDS only alerts against network threats and vulnerabilities, and more intervention is needed to act on these alerts. More than IDS, IPS alerts and blocks network threats and vulnerabilities based on a signature approach. Automated thread response is based on user-predefined thresholds.

Most of the current IDS/IPS are capable to respond in real time or near real time to network activities. This is achieved primarily by reactive solutions, typically as a set of rules. These rules are used to identify known patterns of malicious behavior, such as known attack signatures, and to trigger an alert or take an action to block activity in real time [8]. However, the effectiveness of these reactive solutions is limited by the system's ability to recognize previously unknown or novel attack patterns. To address this limitation, the use of machine learning (ML) algorithms is merged to learn from historical data and proactively detect and prevent potential threats [9].

In this context, network behavior analysis (NBA) [10] is a technique used in cybersecurity to monitor and analyze network traffic for suspicious behavior (Table 1). Unlike misuse-based detection methods that rely on known attack patterns, NBA is designed to detect anomalous behavior that may indicate a new or unknown threat. NBA involves analyzing network traffic in real time to identify patterns of behavior that deviates from normal usage. This approach can help detect a wide range of attacks, including malware infections, data exfiltration, and insider threats. Examples of cybersecurity platforms and tools [11, 12] that can incorporate NBA techniques are IDS/IPS software such as ZEEK, Snort, Nessus, Suricata, Zabbix, OSSEC, Flow-Mon, and Rapid7.

2.2 Predictive analysis in cybersecurity

A data-driven approach and intelligent data modeling extract information from large volumes of data for informed decision making, using advanced analytics to identify patterns, predict trends, and respond to threats and opportunities [13]. There are two main types of network analysis:

- *Misuse-based detection*, also called the knowledge-based or signature-based approach, aims to detect known attacks by using the signatures of attacks. Its strong side is its high accuracy and it can identify types of attacks. However, it is a reactive solution and requires a data background of attacks. There is a lack of real labeled datasets of known patterns due to the diversity of variety of IDS/IPS working in real production.
- *Anomaly-based detection* considers an anomaly as a deviation from a known behavior (other words: profile, baseline, threshold), representing the normal or expected behaviors derived from monitoring regular activities over a period of time [14]. It is a proactive solution, predicting in advance and in time, for any type of attack and does not require a data background of attacks. However, the approach may provide low accuracy in dynamic environments where all events dynamically change.

In the context of misuse-based detection, the works [15–17] present a range of techniques for data preprocessing, data filtering, feature learning, and representation, which are presented across multiple topics, including semi-supervised anomaly detection and insider detection of network traffic [18]. The authors compare and select from a variety of ML methods such as logistic regression (LR), Naïve Bayes (NB), k-nearest neighbor (kNN), support

Table 1 Cybersecurity tools and platforms that can incorporate NBA techniques

Tool	Production IDS/IPS software
ZEEK	Open-source IDS that provides real-time analysis of network traffic
Snort	Open-source IPS that uses signature-based detection to identify known threats
Nessus	Vulnerability scanner that identifies weaknesses in monitored systems and networks
Suricata	Open-source IPS that uses signature-based and behavior-based detection techniques
Zabbix	Real-time alerts and notifications of security events
OSSEC	Host-based IPS that monitors system logs and other data sources for signs of an intrusion
FlowMon	Network monitoring and analysis tool that provides real-time visibility into network traffic
Rapid7	Includes IoT tools for vulnerability management, incident detection and response, and threat intelligence

vector machine (SVM), decision tree (DT), random forest (RF), extreme learning machines (ELMs), and multilayer perceptron (MLP). These works utilize various techniques to calculate anomaly scores, such as local outlier factor, one-class SVM, isolation forest, histogram-based outlier score, or distance distribution-based anomaly detection. To calculate the distances between feature spaces, dimensionality reduction techniques, such as principal component analysis (PCA) or random projection, are often used.

Anomaly-based detection is also a widely used technique to identify situations in data that deviate from expected behavior. It finds applications in various domains, including fraud detection in credit card, insurance, traffic flows, intrusion detection in the cybersecurity industry, and fault detection in industrial analytics [19]. In time series, the objective is to recognize temporal patterns in past data and to use these results for forecasting. Kalman filtering method was one of the models capable of resolving regression concerns and minimizing variance to achieve optimal results. After that, the auto-regressive integrated moving average (ARIMA) model is a well-known and standard framework for predicting short-term data flow. Numerous modifications to the ARIMA model were implemented, and the results ensured improved performance [20]. Unsupervised real-time anomaly detection for streaming data was adequately investigated in [21] to evaluate the performance of hierarchical temporal memory networks over the Numenta Anomaly Benchmark (NAB) corpus that contains a single metric with timestamps.

In works [22, 23], the authors present dynamic monitoring characteristics and propose the use of incremental DL to handle concept drift in simulated evolving data stream scenarios. The comparison result between traditional approaches such as ARIMA versus DL showed that DL outperformed traditional approaches, providing better results. We can list DL methods from references such as NN, deep neural networks (DNNs), convolutional neural networks (CNNs), recurrent neural network (RNN), long short-term memory (LSTM) with incremental learning [24], gated recurrent unit (GRU), autoencoder (AED), and variational autoencoders (VAEDs), as well as their combinations. More examples are restricted Boltzmann machines (RBMs), deep Boltzmann machines (DBMs), deep belief networks (DBNs) [25, 26], and recently transformer [27, 28] for multiple metrics with timestamp problem as sequences.

These works provide valuable information on the strengths and weaknesses of different approaches, helping researchers and practitioners makes informed decisions about the methods to apply to specific use cases and identify inherent obstacles to applying ML/DL in practical domains [30]. The authors report a significant interest in validating the effectiveness of their network intrusion detection approach [31] to simulate a wide range of scenarios that mirror real-world conditions as closely as possible. Using various techniques such as time series, graph network, DL, and their combination [32], the authors aim to develop a robust and reliable system to detect anomalies in network traffic flows.

DL has become increasingly popular in anomaly detection with multihorizon time-series forecasting (Table 2), showing significant performance improvements compared to traditional time-series methods [33, 34]. Multihorizon forecasting is a technique in time-series data analysis [35]. Unlike one-step-ahead forecasting, which predicts the value of a variable only one time period ahead, multihorizon forecasting allows the estimation of future trends over multiple time periods [36]. This capability is especially valuable for situational awareness and decision support, as it enables optimization of actions in multiple steps in advance.

This surge in the use of DL has led to various experiments aimed at developing intelligent modules that address intrusion detection in a complex way. In this problem, patterns that deviate from a baseline, estimated only from normal traffic, are indicated as anomalous. Here is also a discussion of technical challenges of applications of deep ensemble models [37, 38] with the challenges that arise during the data management, model learning, model verification, and model deployment stages, as well as

Table 2 Deep learning models for multihorizon multichannel modeling

DL model	Specific description and usage in our work for data modeling
MLP	MLP works well in many cases with proper hyperparameter setting such as various regularization and the appropriate activation function (Table 7). MLP is used as a baseline for the performance of the models in our work
CNN	CNN is well-known for image processing. CNN is great for extracting features from images and has been shown to be very effective in finding patterns that are difficult to detect with traditional methods. Recent studies that applied CNN to time-series problems show promising results
LSTM	The most well-known RNN type is LSTM. It has a hidden state that is recurrently updated on the basis of previous time steps. LSTM modeling results are not presented in our work because it acts similarly to GRU performance but with a longer runtime
GRU	GRU belongs to the RNN architecture group. It is similar to, but lighter in architecture complexity with performance comparable to that of LSTM
CNN-GRU	In our work, CNN-GRU combines the CNN block (two subsequent Conv1D layers are placed at the beginning) to *extract features* and the GRU block (two stacked GRU layers are used) to learn the representation of the sequence in the data. The settings are in Table 7
s2s-GRU	Sequence-to-sequence GRU or also called GRU-encoder-decoder contains a GRU layer that acts as *encoder*, which processes the input sequence and returns its own internal state. Then, the *RepeatVector* layer is used to repeat the input for the number of multihorizons of times. The other GRU layer acts as *decoder* and learns to generate the output conditioned on the input sequence
AEC	Autoencoder (AEC) compresses a multidimensional sequence into a single vector that represents this information. The final layer is a dense layer that produces a sequence similar to our input. AEC is not presented in our work because it acts similarly to CNN based on preliminary testing
Transformer	The architecture adopt the self-attention mechanism [29], which provides context for any position in the input sequence. In our work, transformer has a simpler form in comparison with Natural Language Processing (NLP) ones. Our *encoder* block contains of *MultiHeadAttention* layer and the *feed forward* part. The model contains four stacked encoder blocks and the head, which consists of dense layers with *sigmoid* activation function for time-series forecasting

considerations that affect the whole deployment pipeline in production [39].

2.3 AI for IT operations

AIOps is the term used to describe the integration of AI and ML algorithms into IT operations to improve the efficiency and reliability of IT services [40]. It represents a significant step forward in the field of IT operations, enabling organizations to take advantage of the power of AI to improve their IT operations and provide better services.

In this context and for cybersecurity intrusion detection research, several artificially generated, public, and synthesis datasets [17, 25, 41] are widely recognized as valuable resources. The most well-known are presented in Table 3. These datasets provide a valuable resource for researchers and practitioners to test and evaluate new approaches to cybersecurity challenges, allowing the development of more effective and robust cybersecurity solutions.

Furthermore, an annotation approach is presented to generate artificial alerts in [49], which includes a pioneering automatic scheduling scheme. This method enables efficient and effective monitoring of data streams, providing timely alerts for potential anomalies or other important events. Similarly, in the work [50], an automated adaptation strategy for stream learning is presented, which has been tested on 36 publicly available datasets and five ensemble ML methods such as the simple adaptive batch learning ensemble (SABLE), dynamic weighted majority (DWM), paired learner (PL), leveraged bagging (LB), and the best last method (BLAST) for stream data analysis. These experiments demonstrate the potential of ML/DL to enable real-time analysis of data streams, helping to detect important patterns and anomalies.

As mentioned in Sects. 2.1 and 2.2, in a wide range of studies, including those mentioned and many others, researchers have conducted experiments that demonstrate promising results in the development and testing of various methods for data analysis and monitoring [51]. These studies have used a variety of public datasets that contain various monitoring information, allowing rigorous testing and comparison of different approaches. The positive results of these experiments suggest that these methods have the potential to be useful in a wide range of applications, from anomaly detection to predictive modeling and beyond. These works have made important contributions to cybersecurity research, their focus has often been on predictive analysis (that is, data analytics with ML methods but without real deployment), using publicly available, synthesis, or artificially generated datasets (Table 4).

Although such datasets can provide valuable information on the performance of different approaches, it is important to recognize that real-world cybersecurity challenges often involve unique and complex data that are not fully represented in them. It is crucial to validate cybersecurity solutions over real data to ensure their

Table 3 Cybersecurity datasets: public, generated, and synthesis

Dataset	Description
DARPA [42]	1998, Defense Advanced Research Projects Agency intrusion detection and offline evaluation datasets used in a series of evaluations to assess IDS
KDDCup'99 [43]	1999, Public dataset, including NSL-KDD, is network traffic data
Kyoto 2006+ [41]	2006–2009, The 3-year dataset of honeypots
UNSW-NB15 [31]	2015, The raw network packets of the data set were created using the IXIA PerfectStorm tool in the Cyber Range Lab of the University of New South Wales Canberra containing normal and attack traffic
ISCX series [44]	2009–2016, Intrusion detection evaluation dataset series, synthesized and provided by the Canadian Institute of Cybersecurity
MAWILab data [45]	2011–today, The large amount of Internet Control Message Protocol (ICMP) traffic from the Analysis of Network Traffic project at the University of Southern California is filtered out to reduce detector computational time. It contains various types of attacks
NAB corpus [21]	2017, Open-source NAB environment contains 58 labeled, artificial time-series, and a scoring mechanism designed for real-time applications
CIDDS dataset [46]	2017, Coburg Intrusion Detection Datasets is an evaluation data set generated for network-based IDS
CIC-IDS series [47]	2017–2018, Dataset series extracted from flows generated from the CICFlowMeter tool, provided by Canadian Institute of Cybersecurity. They include a variety of features of network traffic and has been used for a variety of intrusion detection and malware analysis tasks
CERT series [48]	2014–today, The Computer Emergency Response Team, Division of Software Engineering Institute, Carnegie Mellon University, in partnership with ExactData LLC. and under sponsorship from the DARPA I2O project, generated a collection of synthetic insider threat test datasets. These datasets provide both synthetic background data and data from synthetic malicious actors

effectiveness in practical settings to develop robust and effective cybersecurity solutions.

Network intrusion detection based on anomalies is an area of great interest in cybersecurity research, but its real-world applications pose a significant challenge. Although many studies have focused on developing intelligent approaches to detect anomalies in network traffic, deploying these methods in real production with real stream data and flows is difficult and rarely achieved due to the natural security of the domain as presented in Table 4.

3 Method description

To bridge the gap between research and real production, the AIOps integration process and the challenges involved in the deployment of intelligent approaches for anomaly detection in stream data merges, while technologies are already active [53]. The starting point of our work is the integration of the AI/DL model to cooperate with ZEEK IDS, which supervises the infrastructure network in the online real-time data stream for anomaly detection.

3.1 Cooperation architecture

Network anomalies and detection are critical to ensuring the security of computer networks. Real-time monitoring is essential to detect unusual activity or behavior that may indicate a security breach. To establish a baseline for network behavior, it is crucial to understand what normal activity looks like. This is where behavior baseline comes in, which provides a clear picture of expected network activity.

To make sense of the large amounts of data generated by real-time monitoring and behavior baseline, analysis dashboards are essential. These dashboards provide network administrators with insight into network performance, traffic patterns, and potential security threats.

The cooperation architecture (Fig. 1) in this work is designed with the main components and technologies presented in Table 5. The combination of ZEEK, Apache Kafka, Elasticsearch, Kibana, and Docker and Docker Compose creates a system architecture that is capable of integrating an intelligent module to improve the detection capacity of Zeek. It can also provide a dashboard for visualization analysis with a dynamic behavior baseline produced by the module.

Table 4 Predictive analysis in cybersecurity

Work	Description	Datasets	Methods
[41]	2022, Overview, comparison and development of the cybersecurity datasets	UNSW-NB15, CIC-IDS2017, NSL-KDD, DS2OS, BoT-IoT	–
[21]	2017, Open-source environment for evaluating algorithms for anomaly detection in streaming applications	NAB corpus of public and synthesis datasets	Hierarchical Temporal Memory
[16]	2019, Hybrid anomaly detection to filter network traffic and identify malicious activities	ISCX-2012, ISCX-2017, UNSW-NB15, MAWILab-2018	ELM, MLP, SVM, kNN, DT, LR
[25]	2020, DL for cybersecurity intrusion detection: approaches, datasets, and comparison	CIC-IDS-2018	RNN, DNN, RBM, DBN, CNN, AED
[17]	2021, Comparison of ML methods for cybersecurity intrusion detection, examination of the attack detection capability of IDS datasets	NSL-KDD, ISCX-2012, UNSW-NB15, CIC-IDS-2018, CIDDS-001	SVM, kNN, DT
[30]	2021, Anomaly detection and intrusion detection with semi-supervised learning	Kyoto 2006+, NSL-KDD, UNSW-NB15, CIC-IDS-2017	AED, VAED
[50]	2021, Automated ML and multiple adaptive mechanisms in batch streaming scenario	10 public datasets and 26 synthetic datasets	SABLE, DWM, PL, LB, BLAST
[52]	2021, Network service modeling combining functional data analysis and neuron networks	benchmark functions, Spain private dataset	Threshold, ranking, k-means, PCA, AED, LSTM
[22]	2021, Constructing ensemble learning to handle concept drift in simulated evolving data stream scenarios	Five synthetic datasets, five public datasets	Massive Online Analysis package for simulation
[51]	2022, IoT intrusion detection using deep feature extraction	KDDCUP99, UNSW-NB15, CIC-IDS2017	Deep Feature Extraction
[53]	2022, Connecting data stream production and consumption with ML/AI frameworks	CIFAR-10 public image classification dataset	CNN, data stream, containerization
[54]	2023, Network IDS design combined from feature optimization and classification techniques	UNSW-NB15	Genetic algorithm, DNN
[24]	2023 Incremental learning approach for real-time data processing and decision making for industrial control applications	University of New South Wales IIoT testbed dataset	LSTM-autoencoder

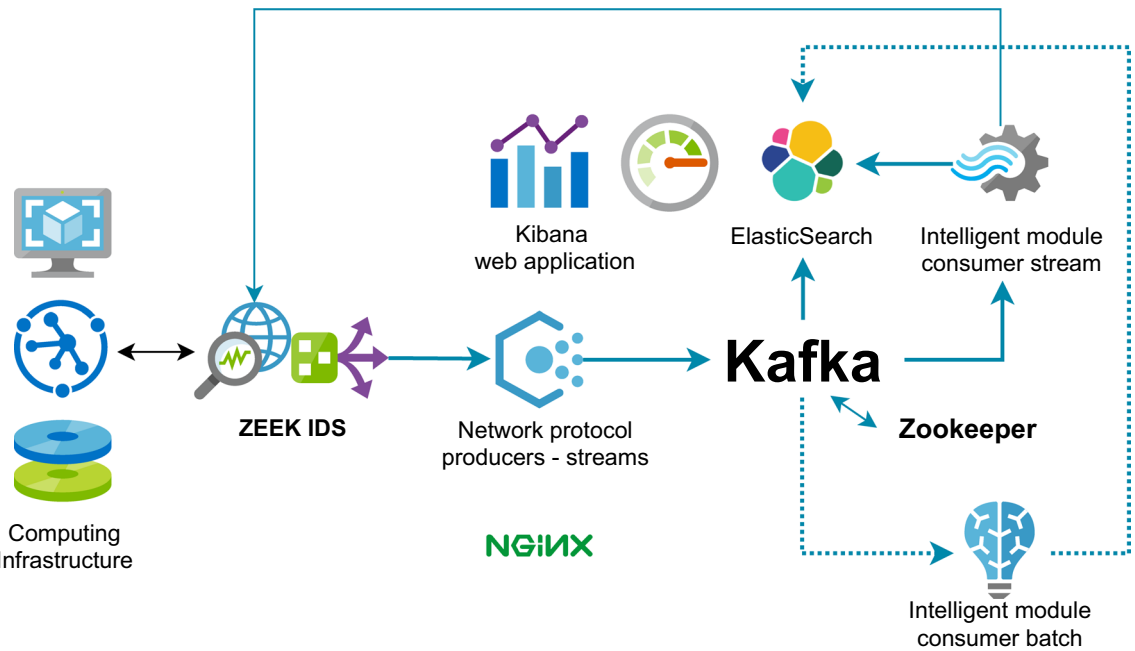


Fig. 1 Cooperation architecture of proactive network monitoring deployment

Table 5 Main software components and technologies of the cooperation architecture

Component	Software and technology
ZEEK	Network IDS that provides real-time monitoring and detection of network anomalies
Apache Kafka	Distributed streaming platform that allows data to be processed in real time
Elasticsearch	Search Engine that enables the storage and retrieval of large volumes of data
Kibana	User-friendly interface for visualizing and analyzing data
Docker	Containerization technologies
Docker Compose	Composition for the deployment of applications in a lightweight and portable manner

The detailed description of each software component in our cooperation architecture with its purpose and key features is as follows:

ZEEK is an open-source IDS that continuously monitors network traffic to detect and identify potential intrusions in real-time. It does so by parsing network traffic and extracting application-level information to analyze and match input traffic patterns with stored signatures. ZEEK is known for its speed and efficiency, and it is capable of performing detection without dropping any network packets. Additionally, it includes an extensive set of scripts designed to minimize false alarms and support the detection of signature-based and event-oriented attacks.

Apache Kafka is an open-source distributed event streaming platform that enables high-performance data pipelines, streaming analytics, data integration, and mission-critical applications. It operates as a cluster of one or more servers communicating through a high-performance transmission control protocol (TCP) network protocol and can span multiple data centers or cloud regions for enhanced scalability and fault tolerance.

It provides a variety of application programming interface (API) in Java and Scala, including higher-level libraries for Go, Python, C/C++, and Representational State Transfer (REST) API. The primary concepts of Kafka include topics, partitions, producers, consumers, brokers, and zookeeper. The main concept of Kafka (Fig. 2) used in our work are:

- *Topics* represent a stream of records or events that are stored in a distributed manner across partitions. They are partitioned data (distributed placements of data) spread over a number of buckets located on different Kafka brokers.
- *Producers* publish data to topics, they are those client applications that publish (write) events to Kafka;
- *Consumers* subscribe to topics to consume the data, they are the clients that read and process these events;
- *Brokers* serve as intermediaries between producers and consumers, managing the storage and replication of data across partitions.
- *Zookeeper* is a centralized service for maintaining configuration information, providing distributed synchronization. Kafka uses it to monitor the status of cluster nodes, as well as topics and partitions. Zookeeper stores the list of existing topics, the number of partitions for each topic, the location of replicas, and access control lists (ACLs).

Elasticsearch is a distributed RESTful search and analysis engine, designed to handle a wide range of use cases. As the core component of the Elastic Stack, it provides lightning-fast search capabilities, fine-tuned relevancy, and robust analytics that can easily scale to handle even the largest datasets. Elasticsearch is the most popular enterprise search engine based on Lucene, offering a distributed multi-tenant capable full-text search engine with a user-friendly hypertext transfer protocol (HTTP) interface and schema-free JavaScript Object Notation (JSON) documents.

Kibana is an open-source front-end application that works seamlessly with the Elastic Stack. Kibana provides search and data visualization capabilities, allowing users to easily explore and analyze data indexed in Elasticsearch. It was originally part of the Elasticsearch-Logstash-Kibana (ELK) stack. Kibana also serves as a user interface to manage, monitor, and secure an Elastic Stack cluster. Its aim is to configure and monitor Elasticsearch indices, visualize data using pre-built dashboards and graphs, and manage security settings to ensure data privacy and access control.

Docker is a standardized unit of software. It is a set of platform-as-a-service products that use operating system (OS) level virtualization to deliver software in packages (containerization). Developers and operation teams use the Docker tool to create and automate the deployment of applications in lightweight containers, for example, on virtual machines, to ensure that applications work efficiently in multiple environments. Using OS-level virtualization, Docker provides a consistent and predictable environment in which applications can run, regardless of the underlying infrastructure [55].

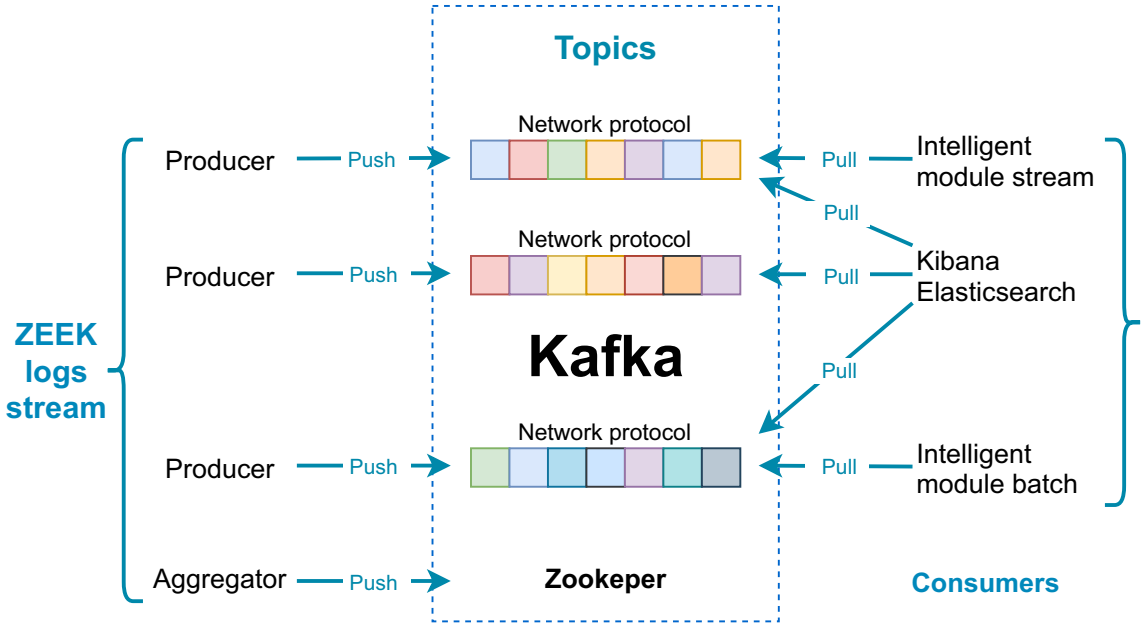


Fig. 2 Kafka concept in the ZEEK log stream context

Docker Compose is used to define and run multi-container docker applications within a single command to create and start all services from the configuration YAML Ain't Markup Language (YAML) file. Docker Compose is used to run multiple containers as a single service. Each of the containers runs in isolation but can interact with each other when required.

The description and the online state of our monitoring stream data are in Sect. 4.2. The snapshot with details of our architecture deployed as the software stack running in its production testing environment is in Sect. 4.1.

In the following sections, the two phases of our proposed intelligent module designed for the cooperation architecture are presented as follows:

- The development of our multihorizon multichannel model is described in Sect. 3.2.
- The online deployment of the model is presented in Sect. 3.3.

The online anomaly detection principle is presented in Sect. 3.4.

3.2 Multihorizon multichannel modeling as dynamic baselines

Complex network monitoring is considered a data function y of the time point t indicated by y_t. It has the form of a multivariate vector containing multiple variables of m multichannel monitoring $(y_t^1, y_t^2, \ldots, y_t^m)$ for proactive modeling of network activity.

$$y_t = \left(y_t^1, y_t^2, \ldots, y_t^m\right) \tag{1}$$

at time t. The mathematical basis for proactive forecasting of the y value at the next q time points $(t+1) \ldots, (t+q)$ is based on the y values at the previous p time points $(t-p+1) \ldots, (t)$ with added/subtracting error terms.

In this work, we extend the motivation based on time-series forecasting backgrounds [56, 57] from forecasting one horizon (or one time point) as in [58] to generalizing the forecast to the k horizon ahead of the direct forecast of the q horizons (Fig. 3) as follows (Eq. 2):

$$\begin{pmatrix} y_{t-p+1}^1 & \cdots & y_t^1 \\ y_{t-p+1}^2 & \cdots & y_t^2 \\ \vdots & & \\ y_{t-p+1}^m & \cdots & y_t^m \end{pmatrix} \rightarrow \begin{pmatrix} y_{t+k}^1 & \cdots & y_{t+k+q-1}^1 \\ y_{t+k}^2 & \cdots & y_{t+k+q-1}^2 \\ \vdots & & \\ y_{t+k}^m & \cdots & y_{t+k+q-1}^m \end{pmatrix} \tag{2}$$

as vector form (Eq. 3):

$$y_{t-p+1}, \ \ldots, \ y_t \rightarrow y_{t+k}, \ \ldots, \ y_{t+k+q-1} \tag{3}$$

where:

$1 \leq k < p$;
$1 \leq q < p$;
k indicates the time points ahead between the current time t and the start time point $(t+k)$ of the forecast;
q indicates the multihorizon forecast from time point $(t+k)$ to time point $(t+k+p-1)$.

The modeling values of y_t, called predicted $\widehat{y}_t$, are used as known behavior (other words: dynamic baseline, dynamic threshold), representing the normal behavior of regular network activities over a period of time t for anomaly detection purposes (more details in Sect. 3.4).

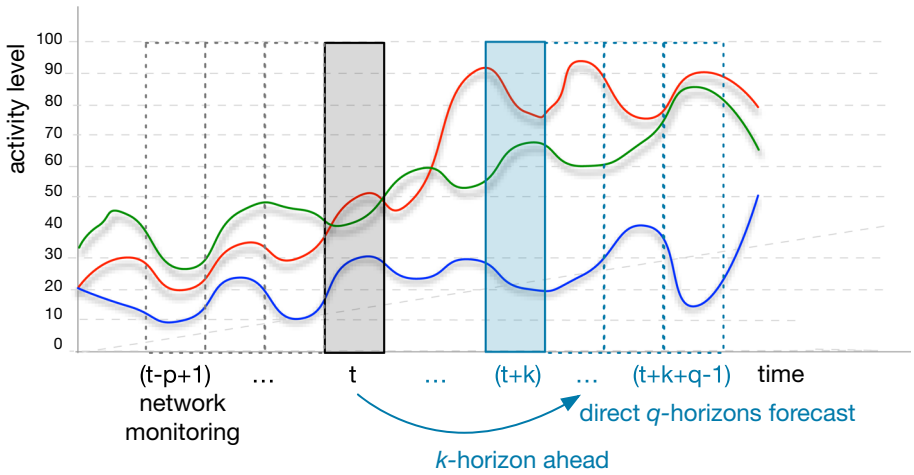


Fig. 3 Multihorizon multichannel modeling

The direct forecasting strategy involves training models using time-series values that have been shifted to the desired number of time periods in the future. It can be improved by using a multichannel forecasting strategy, also known as a multiple-output strategy. With this approach, a single model is developed that can predict the entire forecast sequence in one go. This can make the AIOps deployment easier and more flexible compared to making separate predictions for each time period using multiple models.

Data preprocessing is a crucial step before modeling and prediction. An important aspect of preprocessing is feature engineering, which involves transforming raw logs into time-series data that can be used for modeling. To ensure the quality of transformed data, network protocols are checked against white noise, randomness, and unit root [59] using the augmented Dickey–Fuller (ADF) test [60]. A unit root is a stochastic trend in a time series, sometimes called a random walk with drift. If a time series has a unit root, it shows a systematic pattern that is unpredictable. The origin Dickey–Fuller test is to test the model (Eq. 4).

$$\Delta y_t = y_t - y_{t-1} = \alpha + \beta t + \gamma y_{t-1} + e_t \tag{4}$$

ADF is the augmented version of the Dickey–Fuller test. It allows for higher-order autoregressive processes by including Δy_{t-p} in the model (Eq. 5).

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \delta_1 \Delta y_{t-1} + \delta_2 \Delta y_{t-2} + \cdots \tag{5}$$

For both tests, the null hypothesis (H_0) is a unit root presented in the data. The alternative hypothesis ($H1$) is that the unit root is not present in the data. Mathematically, H_0 states that if $\gamma = 0$ then y is a random walk (or non-stationary) process. If the ADF test returns $p\_value < 0.05$ threshold then y is stationary. The goal of these tests as part of data preprocessing is to identify and filter out protocols that contain excessive noise or randomness, as they are not suitable for modeling. Unlike most of the other forecasting algorithms, DL architecture or models are capable of learning non-linearities and long-term dependencies in the sequence. Then, stationary is less of a concern for DL models. By performing data filtering, the quality of the data is improved, leading to more accurate predictions [61].

To model the complex behavior of network traffic, we employ the state-of-the-art DL architectures (Table 2) to handle online stream data. These DL models were investigated, customized according to our domain data, and compared (Sect. 4.3.2) to select the best, i.e., GRU model for online deployment (Sect. 3.3).

The forecast quality is evaluated using a separated error matrix (Eq. 6) for multihorizon modeling (k horizon before direct forecast of q horizons) of multichannels (m monitoring channels). We denote

$$E = \begin{pmatrix} e^1_{t+k} & e^1_{t+k+1} & \cdots & e^1_{t+k+q-1} \\ e^2_{t+k} & e^2_{t+k+1} & \cdots & e^2_{t+k+q-1} \\ \vdots & & & \\ e^m_{t+k} & e^m_{t+k+1} & \cdots & e^m_{t+k+q-1} \end{pmatrix} = \begin{bmatrix} e^1 \\ e^2 \\ \vdots \\ e^m \end{bmatrix} \tag{6}$$

The average error value $avg(e^j)$ of the series (Eq. 8) of errors e^j of each communication protocol j (Eq. 7) is calculated for the m monitoring channels with direct forecast of the horizons q.

$$e^j = \begin{bmatrix} e^j_{t+k} & e^j_{t+k+1} \cdots & e^j_{t+k+q-1} \end{bmatrix} \tag{7}$$

$$\text{avg}(e^j) = \frac{1}{q} \sum_{i=t+k}^{t+k+q-1} e^j_i \tag{8}$$

Errors are indicated mainly by the smooth version of symmetric mean absolute percentage error (SMAPE) (Eqs. 9). SMAPE is used to avoid zero division. It is also less sensitive to near-zero values compared to mean absolute percentage error (MAPE) (Eqs. 10).

$$\text{SMAPE} = 100\% \frac{1}{n} \sum_{i=1}^{n} \frac{|y_i - \widehat{y_i}|}{\frac{1}{2}(|y_i| + |\widehat{y_i}|)} \tag{9}$$

$$\text{MAPE} = 100\% \frac{1}{n} \sum_{i=1}^{n} \left| \frac{(y_i - \widehat{y_i})}{y_i} \right| \tag{10}$$

Root Mean Square Error (RMSE) (Eqs. 11) and coefficients of determination (R^2) (Eqs. 12) are used as supported metrics.

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \widehat{y_i})^2} \tag{11}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n} \left(y_i - \widehat{y_i} \right)^2}{\sum_{i=1}^{n} \left(y_i - \frac{1}{n} \sum_{j=1}^{n} y_j \right)^2} \tag{12}$$

All the aforementioned metrics measure the inconsistency between the ground truth y and the prediction $\widehat{y}$ of n observations. Mean squared error (MSE) (Eq. 13) and mean absolute error (MAE) (Eq. 14) are used as a loss function in model training.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \widehat{y_i})^2 \tag{13}$$

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \widehat{y_i}| \tag{14}$$

The hyperparameter settings for our DL architectures are presented in Table 7 (Sect. 4.3.1). All features are transformed by normalization scaling [62]. The *dropout* layer works great with our data. The main advantage of using unsupervised (self-supervised) methods is that we do not need to label the data. The main disadvantage is that detected patterns may not be anomalous, but rather intrinsic to the dataset.

3.3 Model online deployment

Teacher forcing is a method of quickly and efficiently training and retraining neural network models using ground truth values. It is a critical training method used in DL for natural language modeling, such as machine translation, text summarization, image captioning, and many other applications. In some cases, teacher forcing is also called changeably incremental online learning. After the model is first trained, it is used to start the process and ensures the predicted output in a predefined interval. The recursive output-as-input process approach can be used when training the model for the first time, but it can result in problems like model instability or poor skill in a dynamic changing environment. Teacher forcing is used to address model skills and stability regularly over time [63].

Algorithm 1 Online model execution over streamed data with teacher forcing periodic model updates

```
1: procedure ExecOnlineModel(topic_in, topic_benign, topic_pred, model, features_in,
   features_out, context_len, update_context_len, slide_step, update_period)
2:     consumer ← new Kafka consumer of topics topic_in, topic_benign
3:     producer ← new Kafka producer of topic topic_pred
4:     buf_in ← empty list
5:     buf_benign ← empty list
6:     last_update ← 0
7:     for message in consumer do
8:         if message.topic == topic_in then
9:             data ← message.value[features_in]
10:            buf_in.append(data)
11:            if |buf_in| ≥ context_len then
12:                X ← buf_in
13:                Y ← model.predict(X)                ▷ online model prediction over streamed data
14:                producer.send(Y)
15:                buf_in ← buf_in[slide_step :]
16:            end if
17:        else if message.topic == topic_benign then
18:            data ← message.value[features_in ∪ features_out]
19:            buf_benign.append(data)
20:            last_update ← last_update + 1
21:            if |buf_benign| ≥ update_context_len & last_update ≥ update_period then
22:                XY ← buf_benign
23:                model ← model.update(XY)            ▷ online model update over streamed data
24:                buf_benign ← buf_benign[update_period :]
25:                last_update ← 0
26:            end if
27:        end if
28:    end for
29: end procedure
```

DL comes with more complex architectures, which promise more accurate learning, but at the cost of model complexity and computing resources. Our choice of GRU architecture is based on the comparison analysis of the DL architecture [58] and based on the analysis done in this work for the implementation of online learning with real-time stream data (Sect. 4.3). GRU behaves well in the development phase, and teacher forcing is supported in realization with production-grade TensorFlow for deployment.

Due to the dynamic nature of the monitoring data, model adaptation and its updating are needed during the deployment, especially when the model is run for a longer period of time. This process is described in (Algorithm 1). The prediction model is deployed within a Kafka consumer that consumes two streams online:

- *topic_in*: used for the prediction, and
- *topic_benign*: used to update the model.

Both streams are processed using a sliding window approach, but each with different length and step settings. The resolution of time (that is, the time step) is set to 10 min according to the domain experience and the delay effect described in Sect. 4.2.2 with the statistical results presented in Fig. 6.

Regarding the prediction task (Algorithm 1, lines 9–15), the sliding window defines the prediction context. The length of the sliding window is *context_len* and corresponds to the number of recent time steps that the model sees in its input. Step *slide_step* is the same as the model forecast horizon, in our case, it is one time step corresponding to 10 min (Table 7).

Predictions are made continuously at each sliding window step (Algorithm 1, line 13) by feeding the model with a sliding window content; that is, a multivariate time series of features *features_in*. The forecasting results in the form of a feature vector *features_out* enriched with model metadata (for example, model name) are sent to the output

stream *topic_pred* with the help of a Kafka producer (Algorithm 1, line 14), where they can be further processed by other consumers, such as a consumer feeding Elasticsearch (Fig. 5).

Updates to the online model are made periodically if sufficient training data are received from the benign data stream (Algorithm 1, line 20). In our case, for the 24-h update period and the training data for 24 h, the sliding window length (*update_context_len*) and its step (*update_period*) are set to 144 (24 h / 10 min = 144).

The complexity of ExecOnlineModel (Algorithm 1) is $O(n)$, where n is the number of *message* in Kafka's *consumer* (Algorithm 1, line 7). The algorithm presents the inference using the pre-trained model in production, which is not computationally or communication intensive. The model works in place with a regular amount of online stream data from the monitoring log flow (Algorithm 1, lines 2 and 3) in the context of one current sliding window (Algorithm 1, from line 7 to line 14).

3.4 Online anomaly detection

Anomalies in network monitoring are identified as significant deviations from the values of normal activity level [14]. The most important point is the precision of the prediction values of $\widehat{y}$ (Sect. 3.2) compared to the real values of y to keep alert warnings at an acceptable level for network administration. The overall anomaly detection score for the m monitoring channels at time t is calculated as:

$$score_t = cosine\left(\widehat{y}_{t,upper}, y_t\right) \tag{15}$$

where:

$$\widehat{y}^i_{t,upper} = (1 + \alpha^i)\ \widehat{y}^i_t;$$

```
  Name                  Command                  State
                                    Ports
-------------------------------------------------------
elastic     /bin/tini -- /usr/local/bi ...   Up
    127.0.0.1:9200->9200/tcp, 9300/tcp
kafka01     start-kafka.sh                   Up
    0.0.0.0:9093->9093/tcp,:::9093->9093/tcp
kafka02     start-kafka.sh                   Up
    0.0.0.0:9094->9094/tcp,:::9094->9094/tcp
kafka03     start-kafka.sh                   Up
    0.0.0.0:9095->9095/tcp,:::9095->9095/tcp
kibana      /bin/tini -- /usr/local/bi ...   Up
    5601/tcp
mods2       python manage.py runserver ...   Up
mods2_db    docker-entrypoint.sh postgres    Up
    5432/tcp
reverse     /docker-entrypoint.sh ngin ...   Up
    0.0.0.0:443->443/tcp,:::443->443/tcp,

    0.0.0.0:80->80/tcp,:::80->80/tcp
zookeeper   /bin/sh -c /usr/sbin/sshd  ...   Up
    2181/tcp, 22/tcp, 2888/tcp, 3888/tcp
```

Fig. 4 The snapshot of the complex software deployment stack running in its production testing environment

Table 6 Component serving instances

Component	Instance
Elasticsearch	`elastic`
Kafka cluster	`kafka`
Kibana instance	`kibana`
DL model	`mods2`
DL model database	`mods2_db`
NGINX	`reserve` proxy service
Zookeeper	`zookeeper`

model is the model architecture in production;
y_t is a vector of real values of m channels at time t;
$\widehat{y}_t$ is predicted values of y_t calculated at time $(t-k)$;
$\widehat{y}_{t,upper}$ is an upper boundary for y_t;
α^i is a threshold coefficient for i^{th} channel in percents;
where $\alpha^{model}_i \geq \frac{1}{2} SMAPE^{model}_i$.

The warning of an abnormal state at time t is called anomaly detection rules applied for multichannel monitoring (Eq. 16), which is activated when

$$score_t > \vartheta \geq 0 \tag{16}$$

where ϑ is a threshold constant reservation.

Algorithm 2 Online anomaly detection

```
1: procedure AnomalyDetection(model, t, k, α, ϑ=0, y)
2:     ŷ_t ← model_{t−k}          ▷ calculated at (t − k)
3:     ŷ_{t,upper} = (1 + α) ŷ_t
4:     score_t = cosine(ŷ_{t,upper}, y_t)   ▷ Equation 15
5:     if score_t > ϑ then                  ▷ Equation 16
6:         warning of an abnormal state at t
7:     end if
8: end procedure
```

Conventional threshold-based methods provide a horizontal baseline line. DL data modeling provides a nuanced soft baseline in the form of the prediction $\widehat{y}$ during the monitoring time of the network (Fig. 3). Algorithm 2 works well under IDS production conditions. The core is to obtain the predicted values $\widehat{y}_t$ and use them as a flexible baseline (or threshold). The result is indicated as the blue line in Fig. 10 (Sect. 4.4), where the green line indicates the real values y.

4 Experiments and evaluation

The cooperation architecture (Fig. 1) based on the Kafka concept for ZEEK log stream processing (Fig. 2) is realized as a production testing environment as follows:

- Deployment of the cooperation architecture (Sect. 3.1) as running software stack (Sect. 4.1) with online model deployment for multihorizon multichannel forecasting.
- Online stream data in cooperation with ZEEK supervising network activities (Sect. 4.2);

The details of the experiments carried out in the production testing environment are presented as follows:

- Stream data processing (Sect. 4.2.1);
- Online delayed log effect (Sect. 4.2.2);
- Forecasting quality (Sect. 4.3);
- Anomaly detection results (Sect. 4.4).

4.1 Software stack deployment

The deployment [64] of the complete software stack in our work for data streams follows automation and orchestration with Ansible semantic declaration. The snapshot of the deployment is shown in Fig. 4 with the component serving instances as shown in Table 6.

The Ansible YAML file uses a declarative composition to easily compose, deploy, and scale the software stack (Fig. 4) for our deployment into the testing environment. YAML is a human-readable data serialization language that serves the same purpose as Extensible Markup Language (XML) but with Python-style indentation for nesting. Compared to XML, YAML has a more compact format.

4.2 Online stream data

The next part of our work is cooperation with one instance of ZEEK working in production. The instance is installed on the network routing machine. Here are two kinds of logs:

- *Compressed log files:* ZEEK collects metadata on ongoing activity within a monitored network. This activity is recorded as transaction logs and organized into compressed textual log files that are grouped by date, protocol, and capture time within specific directories `$PREFIX/logs`. These logs are collated and organized into compressed textual log files for long-term storage [65] that can be easily searched and analyzed to identify patterns and trends in network traffic [58].
- *Online stream log files:* ZEEK analyzes traffic according to the predefined setting policy and produces the results of a real-time network activity in a specific and special directory `$PREFIX/logs/current` on the ZEEK monitoring server. These incremental logs are stored as uncompressed log files [66], which serve as a source of streaming data for subsequent analysis and modeling in this work.

By default, ZEEK regularly takes all logs from `$PREFIX/logs/current` and compresses them by date time and archives to `$PREFIX/logs`. The frequency is set to every hour by default and can be specified in the configuration file as the value `$PREFIX`.

Online logs are in human-readable (ASCII) format and data are organized into tab-delimited columns. Logs that deal with analysis of a network protocol often start like with:

- a timestamp,
- a unique connection identifier (UID),
- a connection 4-tuple (originator host/port and responder host/port),
- protocol-dependent activities that's occurring.

Apart from the conventional network protocol specific log files, ZEEK also generates other important log files based on the network traffic statistics, interesting activity captured in the traffic, and detection-focused log files, for example, `conn.log`, `notice.log`, `known_services.log`, `weird.log`.

4.2.1 Data stream processing

Before any further analysis, an initial analysis of the transaction logs was performed to filter out white noise protocols with stochastic characteristics (Sect. 3.2). This

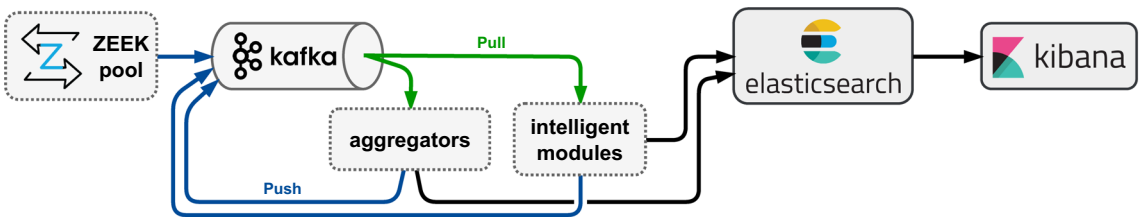


Fig. 5 High-level view on stream data flow

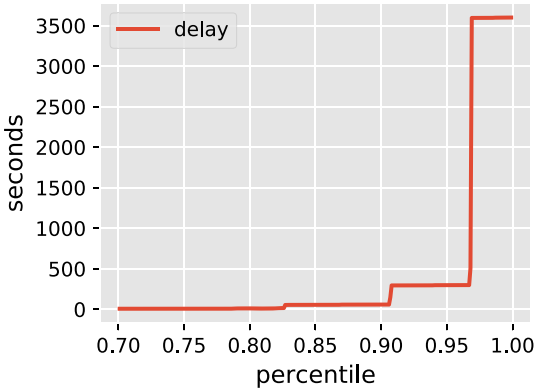


Fig. 6 Statistics of delayed logs in conn.log for a monitoring period of 32 months

filtering process helps to remove irrelevant or noisy data from the dataset, improving the accuracy and efficiency of subsequent analysis steps. For network protocols that pass data validation tests and are used for modeling, it is important to recognize that time-series data have characteristics of ordered time-dependency sequences, where each observation has a temporal dependency on previous observations. Preserving this temporal dependency is crucial for accurate modeling.

For incremental filtered log files `conn.log`, `dns.log`, `http.log`, `sip.log`, `ssh.log`, `ssl.log` of network protocols, dedicated Kafka producers are launched to monitor them for new records in real time. New records are treated as live streams (Fig. 5) and parsed as soon as they appear in the log files.

The parsed logs are immediately sent to the Kafka cluster. They are organized by topic and key (Fig. 2), where the topic refers to message origin (a particular ZEEK instance) and the key refers to originating log files (e.g., `conn`, `http`).

In our work, a concept of aggregators is engaged to consume ZEEK messages from Kafka, and compute statistics over predefined sliding windows (`10min`, `30min` or `1 h`) are applied to preprocess streaming data. Aggregated statistics are pushed back to Kafka as new messages under dedicated topics, e.g., `'zeek1-agg1-10 m'` and keys, e.g., `conn`, `http`. Aggregations are computed according to a specified set of declarative rules. There are two types of aggregation rules:

- `agg` rule computes the number of total and unique connections (for instance `conn` log file), the sum of received and sent bytes, and the average duration of connections within the sliding window.
- `groupby` type of rules is used to group data within the sliding window by a specified column before computing the aggregation statistics.

The `groupby` aggregation rule is useful if we want to compute the statistics for particular column values. An example of such statistics is counting the total number of values `true` and `false` for the column `AA` within the sliding window as follows.

- The rule leads to the creation of two columns in the resulting record; i.e., `dns_AA_T_count` for the `true` value count and `dns_AA_F_count` for the `false` value count.
- Column names are automatically generated according to the original log file and the expansion of the aggregation rules.

Before applying the aggregation rules, the data are grouped within the sliding window by direction of communication into three groups:

- `in`,
- `out`,
- `internal`.

The direction is resolved by IP address, where applicable, and the result is taken from the perspective of the monitored network. It should be noted that before applying any aggregation rules, there is an option to prepare the data consumed using declarative data cleaning rules.

Table 7 Hyperparameter setting

Hyperparameter	Value
m channels	5
q horizons	24
p time points	1, 2, ..., 10
k steps ahead	1, 2, ..., 10
Sliding window size	10 min
Data aggregation	10 min
Teacher forcing update	2 epochs
Early stopping	10 epochs
Restore best weights	True
Stacked units	2
Recurrent units	48
Dense units	128
Dropout rate	0.5
Kernel size	3
Pool size	2
Filters	128
Optimizer	Adam
Activity regularizer	L2
Bias regularizer	L2
Kernel regularizer	L1, L2
Activation function	elu, relu, sigmoid

4.2.2 Online delayed logging effect

The analysis of more than 2.1 billion records collected over a 32-month period revealed that the logs in the incremental files are sometimes delayed. Delays present a problem for online aggregation, because delayed logs are not included in their aggregator sliding window. Figure 6 illustrates the analysis of the `conn.log` data with the graph showing how long an aggregator should wait for delayed logs to receive the desired percentage of incoming logs for the relevant sliding window, i.e., the 300-s interval is needed for delayed log aggregation to obtain 97% of the logs for the actual sliding window.

The `conn.log` file is one of the most important ZEEK logs. It contains logs of TCP/UDP/ICMP connections that have been completed [65]. Log records are written to the incremental log file at the time of completing a connection; i.e., when the connection is closed gracefully or abruptly. There is also a third option when a connection is considered completed, despite the fact that it might still be alive. It is the case when it exceeds the inactivity timer set by ZEEK. This timer is set to 1 h by default, so longer connections are incorrectly broken into multiple sessions.

However, the timer is not the main cause of delays, as there are up to 97% logs with delays below this timer. The user datagram protocol (UDP) and internet control message protocol (ICMP) connections are interpreted by ZEEK using flow semantics; that is, the sequence of packets from a source host/port to a destination host/port is grouped. Each log contains `ts` and `duration` fields, where `ts` is the time of the first packet and `duration` is the elapsed time between the first and last packets in a session. For 3-way and 4-way connection teardowns, the final acknowledgment (ACK) packet is not included in the duration computation, which could be the reason for the delays.

Table 8 MLP, CNN, GRU: model performance by *SMAPE* for multihorizons (1–10)

DL model	Horizons	conn_in	conn_out	dns_out	http_in	ssl_in
MLP	1	32.6074	14.1178	66.9427	65.0237	15.3440
	2	33.6900	16.2179	73.8676	78.1281	15.6130
	3	33.2265	**14.8425**	68.5074	67.8217	16.6031
	4	31.5003	15.6617	71.1183	62.9709	15.6202
	5	29.6596	14.9112	68.9015	65.8445	16.0721
	6	29.6841	15.9388	69.2472	65.8615	16.6836
	7	30.7761	14.4433	62.6327	63.0537	14.6732
	8	32.6167	15.6903	72.2940	70.9173	15.1695
	9	32.9512	15.0010	**68.2066**	75.7919	16.1794
	10	32.5214	15.7100	72.6347	63.8964	15.3866
CNN	1	33.2349	14.9278	70.8350	70.1225	15.3058
	2	34.0094	**15.0491**	71.3221	72.6247	15.3051
	3	34.6965	14.9949	71.7149	74.5933	15.2988
	4	35.7148	**15.2894**	74.2488	72.4131	**15.3917**
	5	35.6431	**15.0365**	73.5564	70.2452	15.3479
	6	36.1117	**15.1240**	73.3048	71.2854	15.3257
	7	36.9947	15.4521	73.1516	74.4957	15.2907
	8	38.9747	**15.7042**	75.1035	73.9931	15.3935
	9	39.3912	**15.7438**	76.1343	72.1810	**15.3259**
	10	39.1869	**15.3281**	76.5981	69.1700	**15.3843**
GRU	1	**24.4384**	**13.4022**	**56.7985**	**39.4903**	14.5497
	2	**24.9317**	15.4523	**66.8484**	52.5418	**14.0335**
	3	26.1116	15.1958	**65.3976**	63.5033	**15.2409**
	4	27.0921	16.1870	**67.2243**	66.7748	15.9430
	5	27.8488	15.1563	69.2780	63.8398	**15.3117**
	6	28.6709	17.1541	**68.6387**	56.2776	16.7705
	7	28.3024	15.4470	**61.3476**	51.5239	14.5048
	8	29.0827	17.1609	**69.4926**	56.7744	14.0241
	9	29.5948	15.8405	68.6654	63.7903	15.3598
	10	29.8149	15.9297	**69.8077**	67.6426	15.9263

Table 9 s2s-GRU, CNN-GRU, Transformer: model performance by *SMAPE* for multihorizons (1-10)

DL model	Horizons	conn_in	conn_out	dns_out	http_in	ssl_in
s2s-GRU	1	27.2171	13.7581	59.9974	45.6975	15.2562
	2	28.9156	16.1512	71.6646	58.4491	14.4694
	3	31.4228	15.6400	68.8700	69.0222	15.9253
	4	32.3798	16.9289	71.1819	74.1978	16.8821
	5	32.4376	14.9325	71.7328	73.8563	15.6125
	6	32.0163	16.1928	73.1490	63.1393	17.0337
	7	31.6855	14.4577	64.7610	**54.7261**	15.0476
	8	31.9523	16.3741	74.0286	63.2969	15.0117
	9	32.5453	15.7045	70.5987	71.9669	16.5346
	10	32.4280	16.0650	71.1921	72.0785	16.7214
CNN-GRU	1	27.3830	14.3280	61.6766	43.3812	14.6482
	2	28.5315	16.2699	68.2829	**49.4702**	14.8999
	3	29.7744	16.3573	68.7906	**58.3108**	15.6417
	4	30.6856	17.0369	72.1817	**64.9133**	15.9550
	5	31.1615	16.3901	73.7276	**58.3675**	15.9773
	6	31.2718	16.4988	70.7121	56.9650	16.3539
	7	30.7577	14.7132	64.9167	**49.7276**	14.5116
	8	30.2340	16.0579	69.5749	**53.6250**	14.6550
	9	30.0093	15.6443	69.0254	**61.7720**	16.0080
	10	**29.5836**	16.6208	70.8198	**66.4263**	16.0096
Transformer	1	25.4973	14.6590	62.4886	57.0054	**10.9180**
	2	25.1883	15.7716	70.6864	60.1888	14.8486
	3	**25.5799**	15.3155	65.2602	64.5912	16.7511
	4	**24.8320**	16.7765	68.0568	67.7150	21.1624
	5	**25.1061**	15.0851	**64.7965**	68.8427	19.4706
	6	**25.5120**	16.0420	70.5444	63.4448	16.2439
	7	**26.9128**	**14.3541**	66.2955	62.0608	**10.6629**
	8	**26.8877**	16.0263	74.0908	61.5305	**13.9453**
	9	**27.5950**	15.0606	68.5557	69.7200	16.7184
	10	**29.0168**	15.9175	74.0419	71.2068	20.8841

The delays in the analysis were calculated as follows: The logs from the beginning are chronologically iterated as they were recorded, while computing the end timestamp `ts_end` for each transaction log as `ts + duration` (Eq. 17).

$$ts\_end = ts + duration \tag{17}$$

Logs with missing `duration` were skipped. The difference between consecutive logs is computed as in Eq. 18. The negative difference *diff* is considered a delay.

$$diff = ts\_end_t - ts\_end_{t-1} \tag{18}$$

Based on our experience and experiments with domain data, the optimal sliding window of 10 min is set with a 10-min step for data aggregation. The waiting time for the delayed logs is set to 5 min, which according to the analysis presented above should cover up to 97% of the actual window logs. Following these settings, the data aggregator is deployed as a Kafka consumer/producer, which reads new Kafka logs and performs data aggregation.

4.3 Forecasting quality

In addition to DL data modeling (Sect. 3.2), the cross-validation based on a rolling window is used, that is, the data are divided into fixed-size sliding windows. This approach allows us to train the model on a subset of the data and then test it on a different subset of the data. By repeating this process, we assess the model performance and make sure that it generalizes well to new data without overfitting and to ensure that our model makes accurate predictions on incoming online stream data.

Table 10 MLP, CNN, GRU: model performance by *cosine* for multihorizons (1-10)

DL model	Horizons	conn_in	conn_out	dns_out	http_in	ssl_in
MLP	1	**0.9470**	0.9807	0.8403	0.6360	0.9744
	2	**0.9425**	0.9761	**0.7764**	0.5771	0.9731
	3	0.9395	**0.9792**	**0.7975**	0.5785	0.9723
	4	0.9376	0.9753	0.7718	0.5751	0.9727
	5	**0.9363**	**0.9779**	**0.7823**	0.5743	**0.9740**
	6	**0.9367**	0.9749	0.7634	0.6023	0.9712
	7	**0.9371**	0.9794	0.8152	0.6282	0.9741
	8	**0.9362**	0.9751	0.7588	0.5801	0.9729
	9	**0.9365**	**0.9787**	0.7858	0.5891	0.9723
	10	**0.9362**	0.9756	0.7654	0.5891	0.9726
CNN	1	0.9281	0.9772	0.7734	0.6034	0.9740
	2	0.9280	0.9772	0.7732	0.6035	0.9740
	3	0.9279	0.9772	0.7731	**0.6035**	0.9739
	4	0.9278	0.9772	**0.7725**	**0.6030**	0.9739
	5	0.9277	0.9773	0.7733	**0.6028**	0.9739
	6	0.9277	**0.9772**	**0.7730**	**0.6031**	**0.9739**
	7	0.9276	0.9772	0.7727	0.6026	0.9740
	8	0.9277	0.9773	0.7723	**0.6031**	0.9739
	9	0.9279	0.9772	0.7723	**0.6031**	**0.9738**
	10	0.9279	**0.9773**	**0.7717**	**0.6027**	**0.9739**
GRU	1	0.9433	**0.9831**	**0.8420**	**0.8853**	**0.9754**
	2	0.9420	**0.9769**	0.7606	0.6065	**0.9750**
	3	**0.9397**	**0.9785**	**0.7815**	0.4165	**0.9738**
	4	**0.9379**	0.9736	0.7693	0.3845	0.9721
	5	0.9348	0.9759	0.7624	0.3613	0.9735
	6	0.9312	0.9716	0.7534	0.4702	0.9711
	7	0.9312	0.9771	0.8137	0.6317	**0.9755**
	8	0.9262	0.9718	0.7432	0.5002	**0.9751**
	9	0.9223	0.9763	0.7745	0.4168	0.9731
	10	0.9233	0.9747	0.7689	0.4243	0.9723

4.3.1 Hyperparameter setting

The complete setting of the hyperparameters is presented in Table 7. Hyperparameter tuning is performed on the basis of the Bayesian optimization sequential design strategy [67] in combination with preliminary exploratory online data analysis (Sect. 4.2.2) and our previous experience [68] with data and DL modeling in the domain.

4.3.2 Model performance

The deployment model works well to monitor simultaneously *conn_in*, *conn_out*, *dns_out*, *http_in*, *ssl_in* network protocols. *SMAPE* and *cosine* are used as main metrics for the performance measurements of the model, and the rest are auxiliary supporting metrics. In practice, the number of metrics used is greater for the development and deployment of models. It is appropriate to mention that the range *SMAPE* is $<0\%, 200\%>$ (Eqs. 9).

- The forecasting quality for multiple (1–10) horizons in one go with *SMAPE* (Tables 8, 9). The best model performance value (the lower value is the better quality) of combinations (*protocol* and *horizon*) is highlighted in bold for the values in these two tables together for DL models: MLP, CNN, GRU, s2s-GRU, CNN-GRU, and transformer.
- The forecasting quality for multiple (1–10) horizons in one go with *cosine* (Tables 10, 11). The best model performance value (the higher value is the better quality) of combinations (*protocol* and *horizon*) is highlighted in bold for the values in these two tables together for DL models: MLP, CNN, GRU, s2s-GRU, CNN-GRU, and transformer.

Table 11 s2s-GRU, CNN-GRU, Transformer: model performance (*cosine*) for multihorizons (1–10)

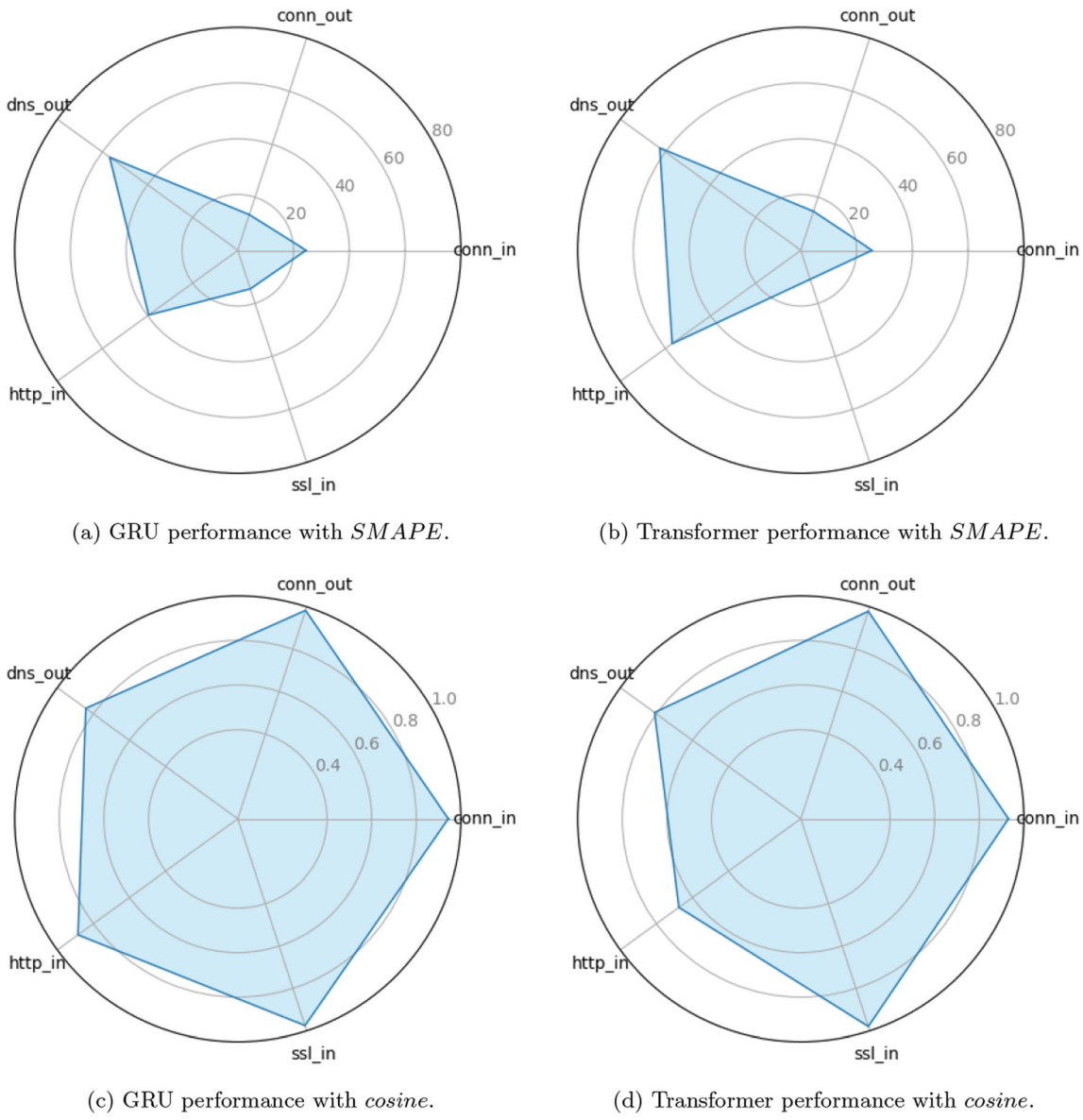
DL model	Horizons	conn_in	conn_out	dns_out	http_in	ssl_in
s2s-GRU	1	0.9367	0.9814	0.8486	0.7417	0.9735
	2	0.9313	0.9743	0.7572	0.5243	0.9737
	3	0.9247	0.9774	0.7738	0.4406	0.9717
	4	0.9202	0.9714	0.7634	0.4058	0.9699
	5	0.9197	0.9766	0.7603	0.3932	0.9727
	6	0.9177	0.9738	0.7458	0.5637	0.9699
	7	0.9160	0.9802	**0.8241**	0.7025	0.9735
	8	0.9121	0.9736	0.7327	0.5295	0.9729
	9	0.9097	0.9771	0.7549	0.4503	0.9705
	10	0.9094	0.9738	0.7488	0.4659	0.9699
CNN-GRU	1	0.9338	0.9811	0.7998	0.8202	0.9751
	2	0.9289	0.9756	0.7488	0.6104	0.9733
	3	0.9249	0.9759	0.7523	0.5139	**0.9724**
	4	0.9209	0.9721	0.7193	0.4193	0.9722
	5	0.9207	0.9729	0.7277	0.4761	0.9725
	6	0.9218	0.9722	0.7364	0.5088	0.9715
	7	0.9247	**0.9783**	0.7981	0.6943	0.9750
	8	0.9246	0.9745	0.7659	0.5126	0.9739
	9	0.9260	**0.9776**	**0.7751**	0.4518	0.9715
	10	0.9294	0.9747	0.7628	0.4008	0.9721
Transformer	1	0.9285	0.9807	0.8313	0.7707	0.9755
	2	0.9279	0.9755	0.7626	**0.6153**	**0.9732**
	3	0.9270	0.9781	0.7882	0.4326	0.9707
	4	0.9260	0.9744	0.7605	0.4414	0.9689
	5	0.9243	0.9760	0.7663	0.4356	0.9717
	6	0.9232	0.9749	0.7572	0.5165	0.9689
	7	0.9243	0.9793	0.8201	**0.7126**	0.9746
	8	0.9231	0.9748	0.7613	0.5548	0.9731
	9	0.9227	0.9781	**0.7896**	0.4154	0.9702
	10	0.9220	0.9746	0.7613	0.4258	0.9691

Table 12 GRU forecasting stability in *n*-step ahead forecast (*SMAPE*)

n	conn_in	conn_out	dns_out	http_in	ssl_in
1	24.4384	13.4022	56.7985	39.4903	14.5497
2	29.2106	13.8107	63.4759	45.1138	15.0773
3	30.1755	15.2928	70.0952	43.3214	14.7377
4	33.1680	14.6043	69.1565	44.4586	14.6016
5	32.2127	15.4750	63.9237	48.2072	15.0173
6	29.2470	14.6225	64.2450	50.4903	14.7420
7	28.6201	14.4581	60.3089	44.9094	14.2955
8	31.4441	14.3926	61.6404	39.8739	14.7126
9	29.5515	14.4690	62.4645	43.3955	14.3298
10	30.3792	14.9451	62.4086	44.0573	14.7379

Table 13 GRU forecasting stability in *n*-step ahead forecast (*cosine*)

n	conn_in	conn_out	dns_out	http_in	ssl_in
1	0.9433	0.9831	0.8420	0.8853	0.9754
2	0.9324	0.9823	0.8188	0.8349	0.9747
3	0.9204	0.9790	0.7991	0.8422	0.9750
4	0.9233	0.9807	0.8005	0.8577	0.9756
5	0.9126	0.9782	0.7755	0.8493	0.9758
6	0.9262	0.9805	0.8055	0.7907	0.9757
7	0.9245	0.9804	0.8002	0.8492	0.9768
8	0.9192	0.9806	0.8018	0.8719	0.9756
9	0.9186	0.9807	0.7980	0.8479	0.9766
10	0.9115	0.9798	0.7856	0.8259	0.9763



(a) GRU performance with $SMAPE$.

(b) Transformer performance with $SMAPE$.

(c) GRU performance with *cosine*.

(d) Transformer performance with *cosine*.

Fig. 7 GRU versus transformer performance for multiple horizon forecast in one go. The lower value indicates the better quality of the model by *SMAPE* in (**a**, **b**). The higher value indicates the better quality of the model by *cosine* in (**c**, **d**)

- The forecasting stability for one-step-ahead of *GRU* with *SMAPE* (Table 12, Fig. 8) and *cosine* (Table 13, Fig. 9).

Experiments were carried out to measure the quality of direct multihorizon forecasts for multichannels. The selected results are presented as follows: Tables 8, 9, 10 and 11 present the quality of the model for multiple horizons (1, 10) in one go simultaneously forecast with metrics *SMAPE* and *cosine*. The model works simultaneously for all selected network protocols. The main key findings based on the evaluations of the experiments are as follows:

- The forecasting quality is not equal for all protocols due to their dynamic nature in the monitored network environment. Specifically, *dns_out* and *http_in* have sharper and more spiky diagrams (Fig. 10), and their fluctuations are more significantly reflected in the metrics, especially in *SMAPE*.
- The *cosine* metric corresponds more or less to *SMAPE* for all monitoring protocols. The average values of metrics for multihorizon forecast in one go are higher in comparison with one horizon forecast.
- The result of the multichannel forecast for one horizon in one go is slightly better than the multihorizon multichannel forecast in one go (Tables 12, 13).
- The quality of the model decreases gracefully with time, which is the expectation and the reason why teacher forcing is used for online deployment.

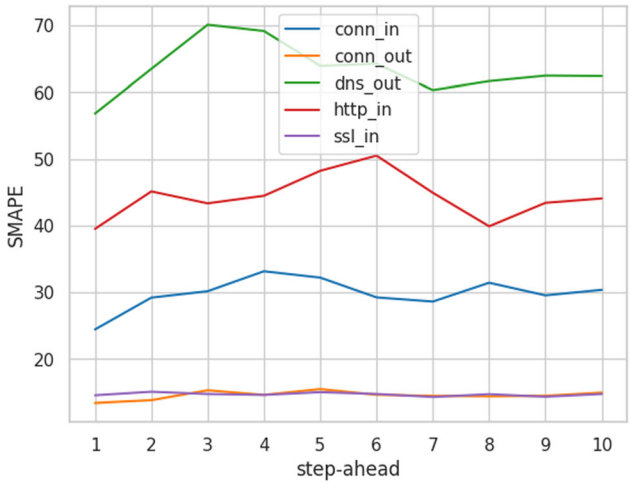


Fig. 8 GRU forecasting stability with *SMAPE* for multiple-step ahead

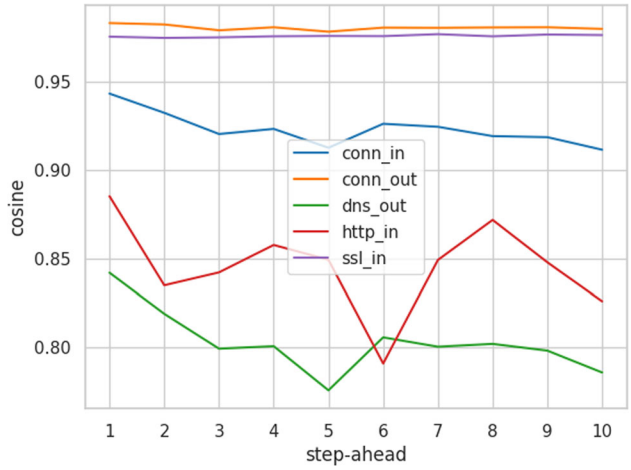


Fig. 9 GRU forecasting stability with *cosine* for multiple-step ahead

Tables 12 and 13 present the prediction stability in the *n*-step ahead forecast with the GRU model. Values are in stable ranges for various metrics (as visualized in Figs. 8, 9); then, we do not emphasize ranges, nor the best values in bold style like in other tables.

Regarding the selection of the best trained model to deploy, the following key findings are concluded:

- GRU, CNN-GRU, and transformer are the top three best models;
- MLP model can be considered as the baseline for performance comparison, and there is no big difference between MLP and CNN models;
- s2s-GRU acts as an encoder-decoder with GRU blocks. The difference between the performance of GRU and the performance of s2s-GRU is not significant in favor of GRU;
- CNN-GRU can be seen as an improvement that combines the advantages of both CNN and RNN. It is better than CNN and nearly as good as GRU;
- Transformer gives interesting and impressive performance. Its performance in the four tables is an average of 10 times training. Sometimes, the model gives superior performance, which outperforms other models. Another time, the training is not always stable, which can be caused by the dropout rate and normalization layers. The consequence of that is that the teacher forcing for transformer is more complicated in comparison with other models, concrete GRU such as the need of learning rate warm-up and linear decay of the learning rate to ensure the model stability.

In summary, GRU belongs to the best models trained with our data. We chose GRU over transformer to deploy after considering and comparing the possible deployment obstacles of both models. GRU (Fig. 7a, c) has simpler requirements with comparable or better performance quality compared to the transformer (Fig. 7b, d).

The effect of direct multihorizon forecast in network monitoring. Multihorizon forecasts (Tables 8, 9, 10, 11) have slightly higher errors compared to the forecast for one horizon (Table 12, 13). This effect can be expected due to the stochastic character of network monitoring and the more complex computational models of multihorizon forecasting on multichannels (Figs. 8, 9). It also reflects the average errors of multiple horizon modeling (Sect. 3.2).

4.4 Anomaly detection results

Figure 10 illustrates the proactive prediction of the normal state as the blue soft line, which takes place 10 min in advance compared to the real monitoring value of the protocols *http* and *ssl* (green lines). The *http* protocol has a clear pattern of connections, which is well modeled by the DL approach. Such patterns are difficult to model with traditional approaches (Sect. 2).

The abnormal state in proactive network monitoring is also illustrated in Fig. 10, where the spikes of the green lines (real monitoring value) significantly exceed the blue soft line of the proactive predicted states. The difference between two lines (blue and green) is calculated in Sect. 3.4 as the angle between two vectors (Eq. 15) with threshold rules (Eq. 16).

It is suitable to note that monitoring systems usually do not have two lines (blue, green) as in Fig. 10 of our Kibana's dashboard, but only one line of real monitoring values.

The next detected anomaly is shown in Fig. 11, where the alerts are displayed in red alert areas. The proactive effect is 10 min in advance (with $k = 1$, that is, one horizon

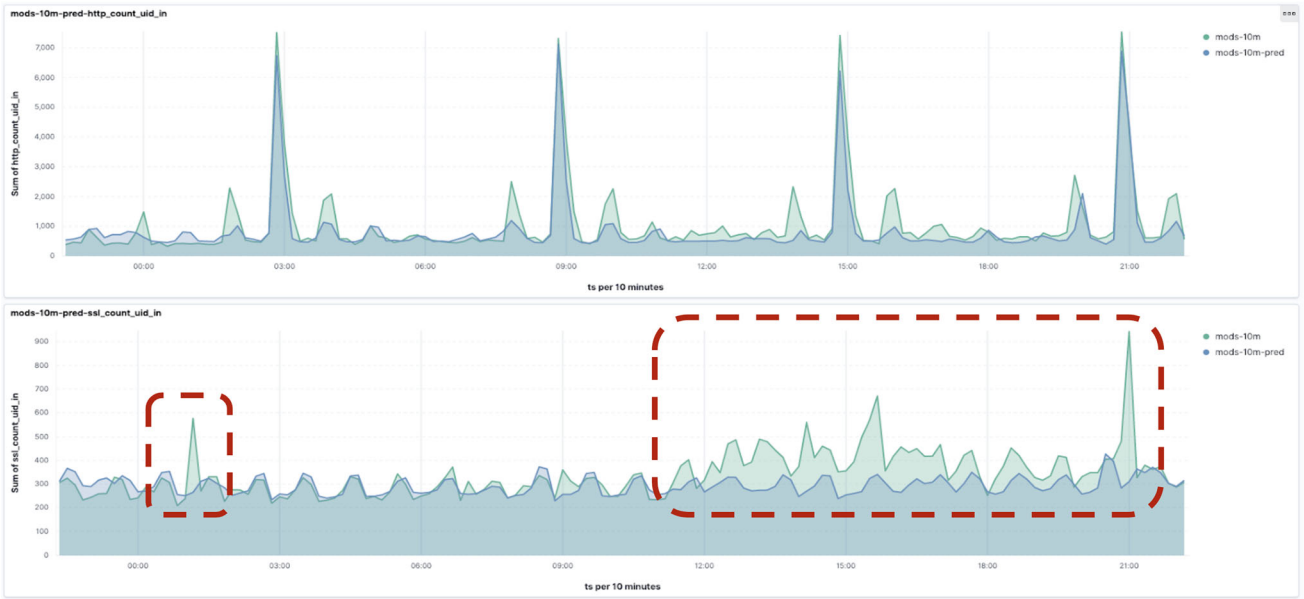


Fig. 10 Kibana's dashboard for online monitoring (HTTP and SSL protocols) backed with Kafka and Elasticsearch in production with GRU model deployment on stream data. The blue line is the forecast used as the baseline over time. The green line is the real monitoring values with anomalies indicated as significant excesses in comparison with the baseline (color figure online)

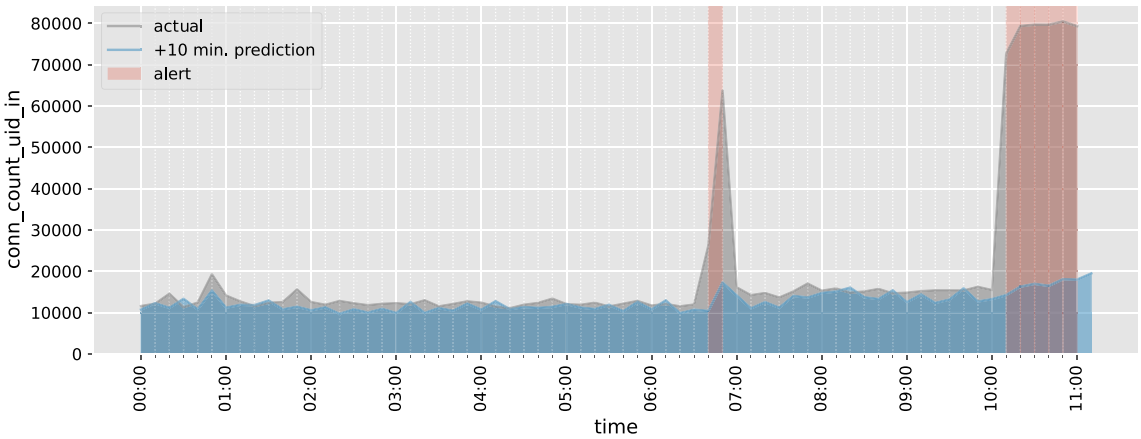


Fig. 11 Alert warning in the Kibana's production dashboard with GRU model deployment: the forecast baseline (gray color) and the actual state (blue color), i.e., the real number of unique incoming connections within a sliding 10 min window. Further inspection of alert-flagged windows (red color) revealed excessive port scanning activity during the Russian–Ukrainian conflict from the outside to the monitored network (color figure online)

ahead forecasting, Fig. 11) in the flavor of the network administration to react to the abnormal situation.

However, the concrete type of anomaly *excessive port scanning* is not detected using our anomaly-based detection but reported as the feedback of the network administrator after log examination. Our anomaly-based detection only alarms the anomaly state (red alert area) in network security according to the high deviation from the baseline (blue area) of the actual connection number (gray area) using the anomaly detection rule (Sect. 3.4, Eq. 16).

The proactive time effect is configurable (with the value of steps ahead $k > 1$) which can give network administrators more time to react. The number of warning alerts is set by the value α as described in Eq. 15. The higher the value of α (in the range of 0 to 1), the lower the number of alerts for anomaly warnings. Our default value α is set to 0.5 for the testing production environment (Fig. 4).

The deployed dashboard (Fig. 10) allows us to add a dynamic adaptive threshold *baseline* to improve awareness of the situation. The development of the entire software stack using a declarative composition simplifies AIOps deployment in practice.

Currently, control over the benign data stream is up to the SOC operators, who filter malicious data out of this

stream based on the monitoring results. In the event of a detected anomaly based on the rules (Eq. 16), the update of the model could also be performed automatically without manual intervention in the automatic teacher forcing process (update and exchange model). The initial model, which is used as the starting model for online prediction, is always trained on benign historical data. When the model is already running online, the quality of the model can be validated backward. If the accuracy of the prediction is below, a threshold or the anomaly level of the data does not exceed the predefined threshold (the parameter α in Eq. 16), then the data can be considered benign and used to update the model. If the accuracy is above the threshold, an anomaly alert is raised and the monitoring data are excluded from the teacher forcing training process (Fig. 5).

We prefer to deploy one model over multiple or ensemble models. The deployment of a single DL model is really not an easy task. Deployment is not a simple process and requires building a complicated pipeline for every model retraining. The combination of more models in production certainly increases the deployment and production cost with model and data curation online in real time.

5 Conclusion

The work presented in this paper presents the development of a DL-based multihorizon forecast model that simultaneously models multiple monitoring channels in an unsupervised manner, allowing a comprehensive view of network activities. With the trained DL model, the deployment of an AIOps module for proactive network security monitoring is realized in cooperation with the IDS working in real-time production.

The deployment of an intelligent module into a production environment is often accompanied by various obstacles, which are presented in this work. Moving forward, there are several directions for future work. A promising approach involves integrating innovative concepts for hybrid detection that combine anomaly-based detection with misuse-based detection for attack classification. It is also imperative to validate anomaly detection scores using well-defined labeled reference datasets. Data curation and model maintenance in production can be time intensive and require thorough verification, especially in the context of evolving data distribution and scale. Another direction is distributed and federated learning for interoperability to reduce the sensitive data sharing silos in cybersecurity.

Acknowledgements This work is funded by the European Union through the Horizon Europe AI4EOSC project under grant number 101058593.

Author contributions G.N. was contributed conceptualization, formal analysis, investigation, methodology, project administration, software, supervision, visualization, writing–original draft, and writing–review and editing. S.D. was involved in data curation, investigation, resources, software, validation, visualization, writing–original draft, and writing–review and editing. V.T.: was performed data curation, resources, software, validation, and writing–review and editing. A.L.G. was responble for methodology, software, validation, and writing–review and editing.

Funding Open access funding provided by The Ministry of Education, Science, Research and Sport of the Slovak Republic in cooperation with Centre for Scientific and Technical Information of the Slovak Republic.

Data availability https://github.com/privacy-security/aiops. We provide a sample of anonymized and preprocessed buffer data of 6 weeks when raw stream data cannot be stored in static form.

Declarations

Conflict of interest The authors declare that they have no known conflict of interest or competing financial interests that could appear to influence the work reported in this paper.

References

1. Tan L, Yu K, Ming F, Cheng X, Srivastava G (2021) Secure and resilient artificial intelligence of things: a honeynet approach for threat detection and situational awareness. IEEE Consum Electron Mag 11(3):69–78. https://doi.org/10.1109/MCE.2021.3081874
2. Fahy C, Yang S, Gongora M (2022) Scarcity of labels in non-stationary data streams: a survey. ACM Comput Surv 55(2):1–39. https://doi.org/10.1145/3494832
3. Chica JCC, Imbachi JC, Vega JFB (2020) Security in SDN: a comprehensive survey. J Netw Comput Appl 159:102595. https://doi.org/10.1016/j.jnca.2020.102595
4. Xin Y, Kong L, Liu Z, Chen Y, Li Y, Zhu H, Gao M, Hou H, Wang C (2018) Machine learning and deep learning methods for cybersecurity. IEEE Access 6:35365–35381. https://doi.org/10.1109/ACCESS.2018.2836950
5. Nisioti A, Loukas G, Laszka A, Panaousis E (2021) Data-driven decision support for optimizing cyber forensic investigations.

IEEE Trans Inf Forensics Secur 16:2397–2412. https://doi.org/10.1109/TIFS.2021.3054966
6. Chen Z, Xu G, Mahalingam V, Ge L, Nguyen J, Yu W, Lu C (2016) A cloud computing based network monitoring and threat detection system for critical infrastructures. Big Data Res 3:10–23. https://doi.org/10.1016/j.bdr.2015.11.002
7. Bhatia S, Liu R, Hooi B, Yoon M, Shin K, Faloutsos C (2022) Real-time anomaly detection in edge streams. ACM Trans Knowl Discov Data 16(4):1–22. https://doi.org/10.1145/3494564
8. Khraisat A, Gondal I, Vamplew P, Kamruzzaman J (2019) Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecurity 2(1):1–22. https://doi.org/10.1186/s42400-019-0038-7
9. Dasgupta D, Akhtar Z, Sen S (2022) Machine learning in cybersecurity: a comprehensive survey. J Def Model Simul 19(1):57–106. https://doi.org/10.1177/1548512920951275
10. Azeez NA, Bada TM, Misra S, Adewumi A, Vyver C, Ahuja R (2020) Intrusion detection and prevention systems: an updated review. In: Data management, analytics and innovation: proceedings of ICDMAI 2019, vol 1, pp 685–696. https://doi.org/10.1007/978-981-32-9949-8_48
11. Cooper S (2023) Intrusion detection systems explained: 14 Best IDS software tools reviewed. Accessed 11 Nov 2023. https://www.comparitech.com/net-admin/network-intrusion-detection-tools/
12. BasuMallick C (2022) Top 10 network behavior analysis software in 2022. Accessed 11 Nov 2023. https://www.spiceworks.com/tech/networking/articles/best-nba-software/
13. Ibrahim A, Thiruvady D, Schneider J, Abdelrazek M (2020) The challenges of leveraging threat intelligence to stop data breaches. Front Comput Sci 2:36. https://doi.org/10.3389/fcomp.2020.00036
14. Melgar-García L, Gutiérrez-Avilés D, Rubio-Escudero C, Troncoso A (2023) Identifying novelties and anomalies for incremental learning in streaming time series forecasting. Eng Appl Artif Intell 123:106326. https://doi.org/10.1016/j.engappai.2023.106326
15. Nguyen G, Nguyen BM, Tran D, Hluchy L (2018) A heuristics approach to mine behavioural data logs in mobile malware detection system. Data Knowl Eng 115:129–151. https://doi.org/10.1016/j.datak.2018.03.002
16. Monshizadeh M, Khatri V, Atli BG, Kantola R, Yan Z (2019) Performance evaluation of a combined anomaly detection platform. IEEE Access 7:100964–100978. https://doi.org/10.1109/ACCESS.2019.2930832
17. Kilincer IF, Ertam F, Sengur A (2021) Machine learning methods for cyber security intrusion detection: datasets and comparative study. Comput Netw 188:107840. https://doi.org/10.1016/j.comnet.2021.107840
18. Le DC, Zincir-Heywood N (2020) Exploring anomalous behaviour detection and classification for insider threat identification. Int J Netw Manag. https://doi.org/10.1002/nem.2109
19. Djeddi AZ, Hafaifa A, Hadroug N, Iratni A (2022) Gas turbine availability improvement based on long short-term memory networks using deep learning of their failures data analysis. Process Saf Environ Prot 159:1–25. https://doi.org/10.1016/j.psep.2021.12.050
20. Sayed SA, Abdel-Hamid Y, Hefny HA (2023) Artificial intelligence-based traffic flow prediction: a comprehensive review. J Electr Syst Inf Technol. https://doi.org/10.1186/s43067-023-00081-6
21. Ahmad S, Lavin A, Purdy S, Agha Z (2017) Unsupervised real-time anomaly detection for streaming data. Neurocomputing 262:134–147. https://doi.org/10.1016/j.neucom.2017.04.070
22. Sun Y, Dai H (2021) Constructing accuracy and diversity ensemble using pareto-based multi-objective learning for evolving data streams. Neural Comput Appl 33:6119–6132. https://doi.org/10.1007/s00521-020-05386-5
23. Aguiar GJ, Cano A (2023) Enhancing concept drift detection in drifting and imbalanced data streams through meta-learning. In: 2023 IEEE international conference on big data (BigData). IEEE Computer Society, pp 2648–2657. https://doi.org/10.1109/BigData59044.2023.10386364
24. Takele AK, Villány B (2023) LSTM-autoencoder based incremental learning for industrial Internet of Things. IEEE Access. https://doi.org/10.1109/ACCESS.2023.3339556
25. Ferrag MA, Maglaras L, Moschoyiannis S, Janicke H (2020) Deep learning for cyber security intrusion detection: approaches, datasets, and comparative study. J Inf Secur Appl 50:102419. https://doi.org/10.1016/j.jisa.2019.102419
26. Thaseen IS, Chitturi AK, Al-Turjman F, Shankar A, Ghalib MR, Abhishek K (2020) An intelligent ensemble of long-short-term memory with genetic algorithm for network anomaly identification. Trans Emerg Telecommun Technol. https://doi.org/10.1002/ett.4149
27. Tang B, Matteson DS (2021) Probabilistic transformer for time series analysis. In: Advances in neural information processing systems, vol 34, pp 23592–23608. https://proceedings.neurips.cc/paper/2021/file/c68bd9055776bf38d8fc43c0ed283678-Paper.pdf
28. Zerveas G, Jayaraman S, Patel D, Bhamidipaty A, Eickhoff C (2021) A transformer-based framework for multivariate time series representation learning. In: Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining, pp 2114–2124. https://doi.org/10.1145/3447548.3467401
29. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I (2017) Attention is all you need. In: Advances in neural information processing systems, vol 30. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
30. Pérez D, Alonso S, Morán A, Prada MA, Fuertes JJ, Domínguez M (2021) Evaluation of feature learning for anomaly detection in network traffic. Evol Syst 12(1):79–90. https://doi.org/10.1007/s12530-020-09342-5
31. Moustafa N, Slay J (2015) Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In: 2015 Military communications and information systems conference (MilCIS). IEEE, pp 1–6. https://doi.org/10.1109/MilCIS.2015.7348942
32. Pérez SI, Moral-Rubio S, Criado R (2021) A new approach to combine multiplex networks and time series attributes: building intrusion detection systems (IDS) in cybersecurity. Chaos Solitons Fractals 150:111143. https://doi.org/10.1016/j.chaos.2021.111143
33. Ismail Fawaz H, Forestier G, Weber J, Idoumghar L, Muller P-A (2019) Deep learning for time series classification: a review. Data Min Knowl Disc 33(4):917–963. https://doi.org/10.1007/s10618-019-00619-1
34. Lim B, Zohren S (2021) Time-series forecasting with deep learning: a survey. Philos Trans R Soc A 379(2194):20200209. https://doi.org/10.1098/rsta.2020.0209
35. Lim B, Arık SÖ, Loeff N, Pfister T (2021) Temporal fusion transformers for interpretable multi-horizon time series forecasting. Int J Forecast 37(4):1748–1764. https://doi.org/10.1016/j.ijforecast.2021.03.012
36. Quaedvlieg R (2021) Multi-horizon forecast comparison. J Bus Econ Stat 39(1):40–53. https://doi.org/10.1080/07350015.2019.1620074
37. Yang Y, Lv H, Chen N (2022) A survey on ensemble learning under the era of deep learning. Artif Intell Rev. https://doi.org/10.1007/s10462-022-10283-5

38. Ganaie MA, Hu M, Malik A, Tanveer M, Suganthan P (2022) Ensemble deep learning: a review. Eng Appl Artif Intell 115:105151. https://doi.org/10.1016/j.engappai.2022.105151
39. Paleyes A, Urma R-G, Lawrence ND (2022) Challenges in deploying machine learning: a survey of case studies. ACM Comput Surv 55(6):1–29. https://doi.org/10.1145/3533378
40. Gil L, Liska A (2019) Security with AI and machine learning, 1st edn. O'Reilly Media, Inc., Sebastopol
41. Alshaibi A, Al-Ani M, Al-Azzawi A, Konev A, Shelupanov A (2022) The comparison of cybersecurity datasets. Data 7(2):22. https://doi.org/10.3390/data7020022
42. Lippmann R, Haines JW, Fried DJ, Korba J, Das K (2000) The 1999 DARPA off-line intrusion detection evaluation. Comput Netw 34(4):579–595. https://doi.org/10.1016/S1389-1286(00)00139-0
43. Tavallaee M, Bagheri E, Lu W, Ghorbani AA (2009) A detailed analysis of the KDD cup 99 data set. In: 2009 IEEE symposium on computational intelligence for security and defense applications. IEEE, pp 1–6. https://doi.org/10.1109/CISDA.2009.5356528
44. UNB: Canadian Institute for Cybersecurity ISCX datasets. Accessed 11 Nov 2023 (2016). https://www.unb.ca/cic/datasets/index.html
45. Fontugne R, Borgnat P, Abry P, Fukuda K (2010) Mawilab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In: ACM CoNEXT 10. http://www.fukuda-lab.org/mawilab/index.html
46. Ring M, Wunderlich S, Grüdl D, Landes D, Hotho A (2017) Creation of flow-based data sets for intrusion detection. J Inf Warf 16(4):41–54
47. Sharafaldin I, Lashkari AH, Ghorbani AA (2018) Toward generating a new intrusion detection dataset and intrusion traffic characterization. ICISSp 1:108–116. https://doi.org/10.5220/0006639801080116
48. Division C (2023) Insider threat test dataset. Accessed 11:2016. https://doi.org/10.1184/R1/12841247.v1
49. Perdices D, García-Dorado JL, Ramos J, De Pool R, Aracil J (2021) Towards the automatic and schedule-aware alerting of internetwork time series. IEEE Access 9:61346–61358. https://doi.org/10.1109/ACCESS.2021.3073598
50. Bakirov R, Fay D, Gabrys B (2021) Automated adaptation strategies for stream learning. Mach Learn 110(6):1429–1462. https://doi.org/10.1007/s10994-021-05992-x
51. Basati A, Faghih MM (2022) DFE: efficient IoT network intrusion detection using deep feature extraction. Neural Comput Appl 34(18):15175–15195. https://doi.org/10.1007/s00521-021-06826-6
52. Perdices D, Vergara JEL, Ramos J (2021) Deep-FDA: using functional data analysis and neural networks to characterize network services time series. IEEE Trans Netw Serv Manag 18(1):986–999. https://doi.org/10.1109/TNSM.2021.3053835
53. Martín C, Langendoerfer P, Zarrin PS, Díaz M, Rubio B (2022) Kafka-ML: connecting the data stream with ML/AI frameworks. Futur Gener Comput Syst 126:15–33. https://doi.org/10.1016/j.future.2021.07.037
54. Keserwani PK, Govil MC, Pilli ES (2023) An effective NIDS framework based on a comprehensive survey of feature optimization and classification techniques. Neural Comput Appl 35(7):4993–5013. https://doi.org/10.1007/s00521-021-06093-5
55. García ÁL, De Lucas JM, Antonacci M, Zu Castell W, David M, Hardt M, Iglesias LL, Moltó G, Plociennik M, Tran V et al (2020) A cloud-based framework for machine learning workloads and applications. IEEE Access 8:18681–18692. https://doi.org/10.1109/ACCESS.2020.2964386
56. Box GE, Jenkins GM, Reinsel GC, Ljung GM (2015) Time series analysis: forecasting and control. Wiley, Hoboken
57. Hyndman RJ, Athanasopoulos G (2018) Forecasting: principles and practice. OTexts, Melbourne
58. Nguyen G, Dlugolinsky S, Tran V, López García Á (2020) Deep learning for proactive network monitoring and security protection. IEEE Access 8:1–21. https://doi.org/10.1109/ACCESS.2020.2968718
59. Goubeaud M, Joußen P, Gmyrek N, Ghorban F, Kummert A (2021) White noise windows: data augmentation for time series. In: 2021 7th International conference on optimization and applications (ICOA). IEEE, pp 1–5. https://doi.org/10.1109/ICOA51614.2021.9442656
60. Dickey DA, Fuller WA (1979) Distribution of the estimators for autoregressive time series with a unit root. J Am Stat Assoc 74(366a):427–431. https://doi.org/10.1080/01621459.1979.10482531
61. Cerqueira V, Torgo L, Soares C (2023) Model selection for time series forecasting an empirical analysis of multiple estimators. Neural Process Lett. https://doi.org/10.1007/s11063-023-11239-8
62. Talavera E, Iglesias G, González-Prieto Á, Mozo A, Gómez-Canaval S (2023) Data augmentation techniques in time series domain: a survey and taxonomy. Neural Comput Appl. https://doi.org/10.1007/s00521-023-08459-3
63. Lamb AM, Alis Parth Goyal AG, Zhang Y, Zhang S, Courville AC, Bengio Y (2016) Professor forcing: a new algorithm for training recurrent networks. In: Advances in neural information processing systems, vol 29. https://proceedings.neurips.cc/paper/2016/file/16026d60ff9b54410b3435b403afd226-Paper.pdf
64. Pokhrel SR (2022) Learning from data streams for automation and orchestration of 6g industrial IoT: toward a semantic communication framework. Neural Comput Appl 34(18):15197–15206. https://doi.org/10.1007/s00521-022-07065-z
65. ZEEK (2023) Zeek's example logs. Accessed 11 Nov 2023. https://docs.zeek.org/en/current/examples/logs/
66. Zeek (2024) Zeek documentation—Quick Start Guide—Book of Zeek. Accessed 26 Jan 2024. https://docs.zeek.org/en/master/quickstart.html
67. Snoek J, Larochelle H, Adams RP (2012) Practical Bayesian optimization of machine learning algorithms. In: Advances in neural information processing systems, vol 25
68. Nguyen G, Dlugolinsky S, Bobák M, Tran V, López García Á, Heredia I, Malík P, Hluchý L (2019) Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey. Artif Intell Rev 52(1):77–124. https://doi.org/10.1007/s10462-018-09679-z