IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON DEEP LEARNING: SECURITY AND FORENSICS RESEARCH ADVANCES AND CHALLENGES

Received December 17, 2019, accepted January 3, 2020, date of publication January 22, 2020, date of current version January 31, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2968718

Deep Learning for Proactive Network Monitoring and Security Protection

GIANG NGUYEN[1], STEFAN DLUGOLINSKY[1], VIET TRAN[1], AND ÁLVARO LÓPEZ GARCÍA[2]
[1]Institute of Informatics, Slovak Academy of Sciences (IISAS), 845 07 Bratislava, Slovakia
[2]Instituto de Fisica de Cantabria (IFCA, CSIC-UC), 39005 Santander, Spain

Corresponding author: Giang Nguyen (giang.nguyen@savba.sk)

This work was supported by the Designing and Enabling E-Infrastructures for Intensive Processing in a Hybrid DataCloud (DEEP-Hybrid-DataCloud) Project through the European Union's Horizon 2020 Research and Innovation Programme under Grant 777435 and VEGA 2/0125/20 New Methods and Approaches for Distributed Scalable Computing.

ABSTRACT The work presented in this paper deals with a proactive network monitoring for security and protection of computing infrastructures. We provide an exploitation of an intelligent module, in the form of a as a machine learning application using deep learning modeling, in order to enhance functionality of intrusion detection system supervising network traffic flows. Currently, intrusion detection systems work well for network monitoring in near real-time and they effectively deal with threats in a reactive way. Deep learning is the emerging generation of artificial intelligence techniques and one of the most promising candidates for intelligence integration into traditional solutions leading to quality improvement of the original solutions. The work presented in this paper faces the challenge of cooperation between deep learning techniques and large-scale data processing. The outcomes obtained from extensive and careful experiments show the applicability and feasibility of simultaneously modelled multiple monitoring channels using deep learning techniques. The proper joining of deep learning modelling with scalable data preprocessing ensures high quality and stability of model performance in dynamic and fast-changing environments such as network traffic flow monitoring.

INDEX TERMS Deep learning, proactive forecasting, network monitoring, cyber security, anomaly detection, neural machine translation.

I. INTRODUCTION

Computing infrastructures are constant targets of cyber attacks in order to gain access to their valuable assets such as data or computing power [1]–[3]. In order to enforce security policies such as confidentiality, integrity and availability; information risk management has to be built based on cyber security strategy consisting of (at least) the following steps: network monitoring, security protection, intrusion prevention, incident management, user education and awareness, and secure configuration [4], [5]. Each of these steps is challenging and comprises a large portion of research and development.

In general, network attacks can be divided into *passive* and *active* attacks. The purpose of passive attacks is to silently gain information about the target while not changing any data on the target's side; e.g., active or passive reconnaissance. Active attacks aim to change the status of the target, or they introduce new or alter existing data. Regarding the purpose of an attack, computer network attacks can be classified into probe, concurrency, traverse, cheat, infection, exploiting and specific attacks attempting to gain access to the target system by using various types of system weaknesses [6], [7].

Whenever an attack has started it can be detected by analyzing network activities with the help of the intrusion detection system (IDS) supervising computing infrastructure. An ongoing network activity is expected to create a monitoring anomaly, if the activity stands for a behaviour that deviates from the normal or expected one [8], [9]. From the symptomatic viewpoint, computer network anomalies can be divided into:

- *Volumetric-temporal anomalies* indicate changes in usual traffic conditions for particular time slots; e.g., traffic protocol volume, number of connections or connected machines at a specific time.
- *Connection anomalies* including geographic anomalies like changes in the network connection pattern and user anomalies such as suspicious activity of users like unauthorized or malicious use of resources.

The associate editor coordinating the review of this manuscript and approving it for publication was Kim-Kwang Raymond Choo.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

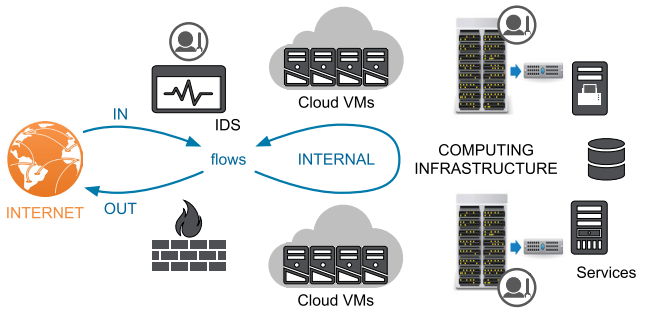


FIGURE 1. Network monitoring and data flows.

Other specific anomalies such as *protocol anomalies* (related to attacks that violate protocol specifications) or *business-defined anomalies* (that require special treatments) are not in the focus of this work.

Most of the current IDSs are capable to respond in real-time or near real-time to network activities. This is primarily achieved by reactive solutions typically as a set of rules [10]–[12]. However, there is a lack of proactive solutions within existing IDSs (Section II-A), which is an open challenge for research and development in this area.

Deep learning (DL) is the emerging generation of artificial intelligence techniques [13], [14]. Recent advancements in this field have made DL one of the most promising candidates for intelligence integration into traditional solutions to improve their quality. DL evolution as a part of natural-inspired techniques has tailored to proactive time series forecasting with applications covering industrial and life areas such as energy consumption, predictive maintenance, remote sensing and financial services [15]–[17]. A common feature of these applications is processing of continuous information flows and adaptation to situations. In this context the cooperation of AI, in particular DL techniques with IDSs, is unavoidable although their usage and potential are still not fully exploited.

The scope of this work covers the proactive approach of network monitoring towards detection of abnormal states in monitored channels in order to protect computing infrastructures (Figure 1). The work proposes a design of an intelligent module that aims to help network administrators to improve monitoring and security protection. The outstanding feature of the work is the cooperation of the cutting edge technologies such as DL [18], large-scale data processing [19] and modern network IDS [10]. Obtained outcomes show the applicability and feasibility of modelling multiple monitoring channels using DL techniques. The work also presents the close to the production state of the intelligent module deployment in cooperation with scalable data processing and IDS supervising network flows.

The structure of this paper is as follows. Section II reviews existing studies and explains our work direction and contributions. Section III presents our system architecture (both for development and deployment) of the intelligent module using deep learning techniques, which cooperates with IDS supervising network traffic flows. Section IV describes the learning phase with proactive forecasting for network traffic monitoring. Details about large-scale data processing and feature engineering are presented in Section V. Section VI contains experiments and evaluations of our approach to prove its effectiveness and feasibility in real production. Section VII concludes main points of the work as well outlines its future development and extension.

II. BACKGROUND AND RELATED WORK

A. INTRUSION DETECTION SYSTEMS

Intrusion detection systems (IDSs) are well-known tools used to monitor and detect suspicious activities in computer networks. They are software applications tailored to monitor network traffic in order to detect malicious activities or non allowed network traffic. These systems are designed to process massive volumes of continuously produced time-ordered raw data with time-limited processing constraints. IDSs must react promptly to occurring events, thus alerts are raised in time and threats can be handled appropriately. Currently, IDSs are based on the following approaches [7]:

- *Misuse-based detection*, also known as knowledge-based detection or signature-based detection, focuses on the detection of intrusions based on the signatures of already known threats, specified in a predefined rule set.
- *Anomaly-based detection* considers an anomaly as a deviation from a known behavior (or profiles), representing the normal or expected behaviors derived from monitoring regular activities over a period of time.

Other detection approaches such as specification-based (or also known as stateful protocol analysis), which depend on vendor-developed generic profiles for specific protocols are not in the focus of this work. All of the above approaches respond to (recent) past events or situations.

According to the systems that can be monitored, IDSs are divided into network IDSs, host IDSs, wireless IDSs, and their combinations. There is a large number of IDSs for network monitoring and the most well-known are:

- *ZEEK/Bro* [10] is an open source misuse detection system that monitors network traffic to identify intrusions in real-time by parsing the network traffic and extracting semantics for application level information. Subsequently, it uses a module to analyze and match the input traffic pattern against stored signatures. ZEEK/Bro is capable of performing detection without dropping any network packet and being faster than its competitors. It includes an extended set of scripts to support detection of signature-based and event-oriented attacks with low number of false alarms.
- *Snort* [20] is a lightweight signature-based IDS that inspects TCP/IP traffic to identify network intrusions based on feature rules and a content pattern matching procedure. Once a packet matches a specified rule pattern, Snort offers three actions: pass rules to drop the packet, log rules to write the full packet selected by a user at runtime, and alert rules to generate a certain set of alarms as specified by the user.

- *Nessus* [11] is an open source plug-in based detection system for vulnerability identification. Its plug-ins are expected to be written very carefully so that they identify their known types of intrusions (knowledge-based detection) with minimum false alarms.
- *Suricata* [21] is a network IDS, which uses the signature method to detect a cyber attack. It operates by fetching one packet at a time from the system to be preprocessed and then feeds the core engine running the detection algorithms. Suricata evaluates packet from the feed and raises an alert if it considers it to be malicious.

Apart from the above mentioned network IDSs, there is a number of other products that can be used to perform intrusion detection to some extent: *Zabbix* [22] is a monitoring engine with trend prediction (regression); the host monitoring system *OSSEC* [23] that uses both signature and profile-based methods to detect cyber attacks; or *Rapid7* [24] the secure facility for IoT devices. However, the main scope of this work is focused on IDSs for network surveillance.

Most of the current IDSs are capable to respond to network activities in real-time or near real-time. This is primarily achieved by reactive solutions (typically as if-then statements/rules) covering specific types of known usage patterns (misuse detection) or deviation from standard conditions (anomaly-based detection). The weakness of misuse detection is the lack of real labelled datasets of known patterns in real production. The weakness of anomaly-based detection is its low accuracy in dynamic environments where observed events change over time [7]–[9].

Based on above references and to the best of our knowledge, there is evidently lacking of proactive monitoring solutions within network IDSs, which stands as an open challenge for research and development in the field of network monitoring and security protection.

B. PROACTIVE FORECASTING AND DEEP LEARNING

Proactive techniques tend to be based on control theory like queueing models [25] or probability estimation [26]. If the monitoring is based on time-ordered data sequences then data mining using sequence modeling, especially time series forecasting [27], [28] is in the center of many research and development efforts. Its evolution is as follows.

Statistic approaches such as autoregressive integrated moving average (ARIMA), autoregressive moving average (ARMA), moving average (MA) and variances, are aimed on prediction of the future values of a time series using regression over the past values of the dependent variable and modeling the stochastic part of them [29], for example, for failure detection [30]. The main limitation is their linear form and that they are not appropriate for a stationary time series (i.e., mean, variance, and autocorrelation should be approximately constant in time [31]). In addition, with the growing size of datasets [32] statistical models hardly capture non-linear patterns to cover the variety that is characteristic for dynamic sequence data [33].

Machine learning (ML) is mainly used in a reactive way. For instance, fault diagnostics and prognostics in energy sector [34] or failure detection in high-performance computing system [35] with decision tree, support vector machines (SVM), artificial neural network (ANN) and k-nearest neighbors (kNN). ML is also used for proactive forecasting such as trend monitoring with regression implementation and threshold specification in Zabbix [12]. The main limitation of ML approach is the prediction accuracy, which is often improved using fuzzy techniques [34], [36].

Meta-heuristic approach with natural-inspired algorithms [37] is considered as a higher level of heuristic approach (e.g., ML). They are also considered as more generalized and less domain-knowledge dependent. These techniques trade quality of the solution for run time, by finding good but not necessarily the optimal solution within a feasible time. These algorithms include genetic algorithms (GA) [38], particle swarm optimization (PSO) [39] with variances such as coral reefs optimization (CRO) [40], galactic swarm optimization (GSO) [41], whale optimization algorithm (WOA) [42] as well as their hybridization and evolution. Meta-heuristic algorithms, for example, are used to optimize neural networks for proactive cloud resource auto-scaling [43], [44].

Neural networks as a sub-field of ML provide advantages of nonlinear modeling. Recently, *deep neural networks* are considered to be capable of learning high-level features with more complexity and abstraction due to the larger number of hidden layers. They can be effectively used to achieve higher accuracy in inference tasks [18].

Deep learning success [13], [14] is believed to be due to the confluence of 3 different factors: 1) new algorithmic advances that have significantly improved application accuracy and broadened applicable domains; 2) availability of huge amounts of data to train neural networks; 3) increasing computing power. DL techniques are compute-intensive, but they are promising in predictive quality [18]. DL can significantly contribute to proactive forecasting as well as anomaly detection in time-ordered data as presented in the recent literature and application report [45].

As a part of DL, recurrent neural networks (RNNs) [46] with internal self-looped cells have been shown to be a promising approach in many domains, for example in fault diagnosis of power plants [15] due to their ability to model long term non-linear dependencies in time. However, RNNs experience vanishing gradient and exploding gradient problems. Long short-term memory (LSTM) is a special RNN block, which was designed to address RNN drawbacks [46]. LSTM is able to learn from time series with long time spans; it is currently considered as the most well-known for its time series forecasting performance from the accuracy viewpoint for applications; e.g., in aircraft industry [17] or remote sensing [16]. It also provides more consistency in predictions over time in comparison to ML models such as decision tree or SVM [47]. LSTMs come in many variances such as gated recurrent unit (GRU), bidirectional LSTM [48], attention LSTM [49] and stacked variances that promise

model quality improvements [50]–[52] at the cost of model complexity and consequently higher requirement of computational power.

It is also suitable to mention the convolutional neural networks (CNNs), autoencoders, generative adversarial networks (GANs), fully connected (FC) networks and any of their combinations. They are also candidates for proactive time series forecasting under neural network umbrella [53]–[55]. The combination of CNN and attention function [50] brings the new DL architecture called temporal convolutional nets (TCN). They have been reported to outperform RNN in several language-to-language translation benchmarks in both speed and accuracy performance [56].

Based on our knowledge, the fusion of DL models with IDSs for proactive monitoring and security protection has not yet been fully investigated and exploited. There is a growing interest in proactive monitoring solutions, meaning that the monitoring systems are expected to provide an estimation of the near future behaviour for multiple monitoring channels simultaneously. If the environment is extremely fast-changing, it is a challenge to keep a normal profile of the multi-channel monitoring activities up-to-date in a *proactive way*. Normal profile with the near future behaviour estimation has to be built proactively based on the knowledge of the past normal activities. Hence, significant deviations from the normal profile should be considered as anomalous activity [6], [57]. The training of DL model is thus required to forecast the normal activity profile in the near future as accurate as possible.

From the speed point of view, DL is making profits from using specialized hardware in accelerated computing environments. The current mainstream solution is to use Graphics Processing Unit (GPU) as general purpose processors to accelerate computing [14], [58], or other advancements such as Tensor Processing Units (TPU) [13], [59].

C. MOTIVATION AND CONTRIBUTIONS OF THE WORK

Based on the context presented in Section II-A and II-B, the motivation of our work is to provide a *proactive network monitoring* solution, which collaborates with IDS supervising computing infrastructure. The aim is to challenge and enhance security protection. The solution joins the newest trend in multivariate time series forecasting with DL promise (Section II-B) for simultaneous modeling of multiple monitoring channels. The major contributions of the work presented in this paper are:

- Full architecture design for both development and deployment phases of the intelligent module cooperating with the IDS supervising network flows.
- Bridging the recent state-of-the-art DL architectures in the problem of multi-channel sequence modeling as the core of the intelligent module for proactive network monitoring and anomaly detection.
- Joining self-supervised learning with data preprocessing while ensuring high quality model performance in dynamic and evolving environment.
- Building a data preprocessing module leveraging a Big Data facility to deliver feature engineering automation, working directly with raw logs and logging process.
- Employing advanced technologies such as DL modeling with GPU support, Big Data processing and data analytic, modern generation IDS to work together for network monitoring and security protection benefits.
- Providing extensive experiments on production datasets with a thorough evaluation of model behavior, design complexity and robustness of the chosen technologies.

III. EMBEDDING ARCHITECTURE FOR INTELLIGENT MODULE

A. SYSTEM ARCHITECTURE

The proposed architecture of a computer network supervising IDS with embedded intelligent module is presented in Figure 2. The solid lines depict data flow, dashed lines control flow, and dotted lines with double-headed arrows stand for labels. The architecture describes workflows for both development and deployment phases with configurable parameters (see Table 1). System components can be seen under 3 main modules, where each module is built of several blocks:

1) The IDS module provides monitoring, logging, and security management of computing infrastructure (green blocks labelled with ''supervising IDS'' in Figure 2).
2) The data processing module manages online and offline processing (dark green blocks) as well as feature engineering (involves data cleaning, data transformation, feature extraction, and feature selection as can be seen in Figure 4). It interacts with the data repository block (Figure 2) and controls model building (the orange block) in deployment phase.
3) The proactive forecasting module produces DL models based on ML methodology and DL modeling. The module closely works with the data processing module to get access to data in the data repository and enable data analytic as well as training and testing models during the development phase.

After a model is developed, it undergoes usual training process known from the ML application development lifecycle. Once the training is complete, the model performance (or model quality as described in Section IV-D) is evaluated and the best model is used in the deployment phase on online data. Evaluation results are reported to the security management system as situational awareness (Section III-C) thus the network administrators can evaluate possible threats.

B. PRODUCTION DATASET AND DATA LABELING

Due to real situations with monitoring services and protocols as well as increase of network bandwidth, the building process of the intelligent proactive module faces Big Data problems (Volume, Velocity, Variety and Veracity). Except that, there is a lack of real datasets with labeled data. It is because the data is sensitive and contains information such as internal details on the system, network security information as well as certain personal information. In general, providers of computing resources apprehensively concern with their monitoring

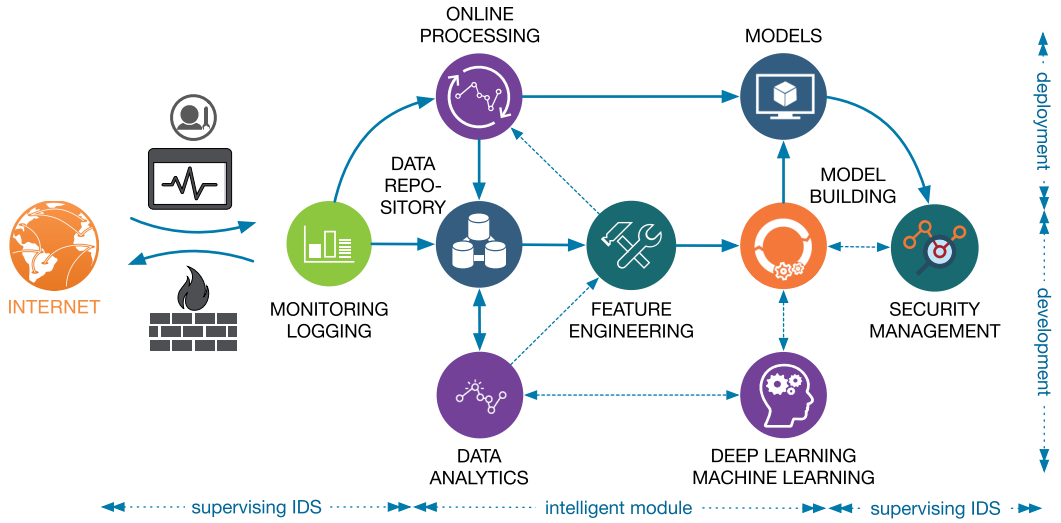


FIGURE 2. Development and deployment of intelligent module for network supervising IDS functionality improvement.

TABLE 1. Parameters for building intelligent module.

Parameter	Description and condition
w	window size (Figure 3)
s	sliding step, $s \leq w$ (Figure 3)
∂	ODE stabilization (Equation 6)
p	sequence dependency length (Equation 4)
k	forecasting horizon, $1 \leq k < p$ (Equation 4)
D	data resource (datapool) specification (Section V)
R	time range specification (Section V)
X	excluded intervals $X \subset R$ (Section III-C)
T	model retrain period, $\tau \leq \delta \leq T$ (Algorithm 1)
τ	transformation period (Algorithm 2)
Q	data selection query (Section V)
m	the number of monitoring channels (Section V)

log data, which also fall under data protection regulations like GDPR in EU [60]. Therefore, real datasets are usually kept private and are not freely distributed [3], [61], [62].

In this work, the dataset is built from a real network monitoring, which is hanging on a production computing infrastructure of the project partners [63]. Raw monitoring logs are collected by ZEEK/Bro network IDS.

From the point of data labelling, ML techniques with DL as its sub-field can be divided into: 1) supervised learning, which learns from labeled training examples, and 2) unsupervised learning, which attempts to extract information from unlabelled train dataset. However, the distinction between supervised and unsupervised learning is not strict, because there is no objective distinguishing whether a value is a feature or a target provided by a supervisor.

In this work, the *self-supervised* approach is used for multivariate time series preprocessing and labeling (Equation 4). It is a part of the data processing (Section V) with parameters (Table 1).

C. ZEEK/BRO AND HYBRID SECURITY SOLUTION

In our architecture, monitoring, logging and security management of network activities is performed by employing ZEEK/Bro (Section III-B). The network IDS comes with an extended set of scripts to support functionality enhancements in a reactive way. However, the design presented in this work can be easily adapted to another network IDS with minimal integration effort. The IDS is supervising network segment in production with hybrid (star) network topology connecting computing infrastructure (Figure 1). The number of nodes is around 200 in real production site and includes nodes of the HPC, Grid, Cloud, and Mesos cluster for high performance computing and Big Data processing as well as web servers, DNS servers, data storage nodes (FTP, SFTP, gridFTP services), service and control nodes.

It is important to mention that an universal solution, which solves all the security problems, does not exist. Here is a need for combining solutions into a complex solution. The hybrid strategy for anomaly detection is composed of misuse detection, outliers detection and proactive monitoring [64]. It works in an extendable way with external intelligent modules as follows: 1) external modules provides their detection results as probabilities [62] to the supervising IDS; 2) based on these probabilities, IDS leverages security protection actions required by the external ML modules.

Misuse detection works fast and in near real-time like reactive solutions. Concretely, ZEEK/Bro is ready to work with external intelligent modules for IDS functionality enhancements. Examples of such modules include our prior work [62] and the works [65], [66].

Outliers detection in a reactive way is also well-examined. ZEEK/Bro Analysis Tools [67] also contain the Isolation Forest method, which can point out anomaly data points in protocol logs as outliers. The IDS is ready to work with other implementations such as logistic regression, SVM, multilayer perceptron, decision tree or kNN if delivered.

Proactive monitoring module in our work detects abnormal states of monitored channels. Anomalies are identified as significant deviations from the normal activity level values. The most important point is the accuracy of forecasting $\widehat{y}$ in comparison with real values of y in order to keep false positive warnings as low as possible. The overall anomaly detection

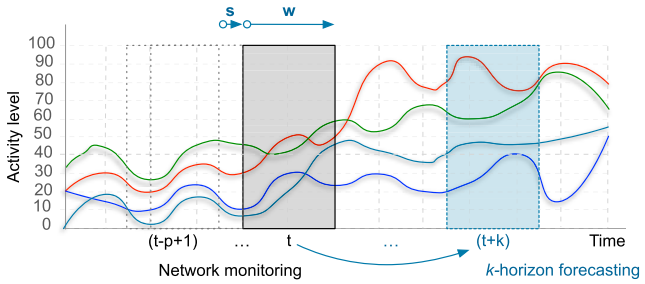


FIGURE 3. Sliding window *w*, *k*-horizon forecasting and *p* sequence dependency.

score for m monitoring channels at time t is calculated in Equations 1 and 2.

$$score_t = cosine\left(\widehat{y}_t^{upper}, y_t\right) \tag{1}$$

$$\widehat{y}_{t,i}^{upper} = (1 + \alpha_i) \cdot \widehat{y}_{t,i} \tag{2}$$

where

- y_t is a vector of real values of m channels at time t;
- $\widehat{y}_t$ is a vector of predicted values of y_t computed at time $(t - k)$;
- $\widehat{y}_t^{upper}$ is an upper boundary for y at time t;
- α_i is a threshold coefficient for i^{th} monitoring channel, where $\alpha_i^{model} > (1/2) \cdot SMAPE_i^{model}$ and $SMAPE$ value is in 200% range;
- $model$ is a type of the selected model in production

Abnormal state warning in time t is triggered when the condition in Equation 3 is valid.

$$score_t > \theta_{score} > 0 \tag{3}$$

Fine-tuning of α and θ is realized in cooperation with network administrators. Experimental results (Section VI-B) show that trained DL models (Section IV) have a stable behavior with low error levels.

IV. PROACTIVE FORECASTING MODULE

A. FORMAL DESCRIPTION

The mathematical ground of proactive forecasting of the y value at the next time point $(t + 1)$ is based on the y values at previous p time points [68], [69] with added/subtracted error terms. It is also generalized to k-horizon forecasting (Equation 4 and Figure 3) as follows:

$$y_{t-p+1}, \ \ldots, \ y_{t-1}, \ y_t \rightarrow y_{t+k} \tag{4}$$

where $(1 \leq k < p)$.

In our domain data, y_t is a multi-dimensional vector containing values $y_{t,1}, \ y_{t,2}, \ldots, \ y_{t,m}$ of m monitoring channels for proactive modeling.

$$y_t = \left(y_{t,1}, \ y_{t,2}, \ldots, \ y_{t,m}\right) \tag{5}$$

Usually, real data is not stationary, however different methods can be applied in order to keep it in its trend [68]. Ordinary differential equation (ODE) describes dynamically changing phenomena using mathematical differentials and derivatives. We denote the use of ODE stabilization as ∂ parameter in Table 1. The first order of ODE expresses the evolution of y in time denoted as y' (Equation 6).

$$y' = \frac{\partial y}{\partial t} = \left(\frac{\partial y_1}{\partial t}, \frac{\partial y_2}{\partial t}, \ldots, \frac{\partial y_m}{\partial t}\right) \tag{6}$$

B. LEARNING PHASE

Based on DL architecture templates, the learning phase employs the following models, with detailed descriptions presented in the remainder of this subsection:

(a) multilayer perceptron (MLP),
(b) autoencoder MLP,
(c) long short-term memory (LSTM),
(d) gated recurrent unit (GRU),
(e) bidirectional LSTM,
(f) autoencoder (seq2seq) LSTM,
(g) attention LSTM,
(h) convolutional neural network (CNN),
(i) temporal convolutional nets (TCN),
(j) stacked models (LSTM, TCN).

Neural networks are effective universal approximators, which have recently shown promising results in many ML tasks including those with sequence inputs [70]. The simplest and widely used formal description of a neural network is the nonlinear weighted sum (Equation 7)

$$y = f_A(x) = f_A\left(b + \sum_i^n w_i x_i\right) \tag{7}$$

where y is the output, f_A is the activation function (e.g., tanh, sigmoid, softmax or rectifier), b is the bias; w is the weight vector; and x is the input vector of size n.

1) MULTILAYER PERCEPTRON (MLP)

MLP is build as a baseline model for comparison. Our MLP model is a feed-forward neural network (FFNN) with (at least) 3 fully-connected (FC) layers: input, hidden and output. The number of hidden layers is customizable and MLP model is trained using back-propagation.

2) AUTOENCODER MLP

MLPs can be deeper with more layers in special structures, such as non-recurrent autoencoder with MLP. The autoencoder consists of two parts: encoder and decoder [18], [71]. The encoder (Equation 8) compresses data from the input (X) into a short code (latent vector). The decoder (Equation 9) decompresses the code back, attempting to minimize the reconstruction error through the euclidean distance (Equation 10).

$$\phi \ : \ X \rightarrow F \tag{8}$$

$$\psi \ : \ F \rightarrow X \tag{9}$$

$$\phi, \psi = \underset{\phi,\psi}{arg\,min} \ \|X - (\psi \cdot \phi)X\|^2 \tag{10}$$

When observations are modeled as multi-channel vectors, our MLP encoder consists of FC 128, 64, and 32 neuron layers. The decoder is constructed the same way, but the in reverse order. The latent layer is a FC 16 neuron layer. The sizes of the filters are based on experiments conducted on domain data.

3) LONG SHORT-TERM MEMORY (LSTM)

As mentioned in Section II-B, RNNs handle sequences by having a recurrent hidden state, whose activation at each time is dependent on previous time steps [18]. The most famous RNN special units (building blocks) are LSTM (Equations 11 to 16) and GRU (Equations 17 to 20). They are similar in overall structure, but GRU has a simpler form [46], [72].

$$\text{Input gate } i_t = \sigma\left(x_t U^i + h_{t-1}\mathcal{W}^i\right) \quad (11)$$

$$\text{Forget gate } f_t = \sigma\left(x_t U^f + h_{t-1}\mathcal{W}^f\right) \quad (12)$$

$$\text{Hidden state } \tilde{\mathcal{C}}_t = tanh\left(x_t U^g + h_{t-1}\mathcal{W}^g\right) \quad (13)$$

$$\text{Internal memory } \mathcal{C}_t = \sigma\left(f_t * \mathcal{C}_{t-1} + i_t * \tilde{\mathcal{C}}_t\right) \quad (14)$$

$$\text{Output hidden state } h_t = tanh(\mathcal{C}_t) * o_t \quad (15)$$

$$\text{Output gate } o_t = \sigma\left(x_t U^o + h_{t-1}\mathcal{W}^o\right) \quad (16)$$

where x is the input vector; $\mathcal{W}$ is the recurrent connection at the previous hidden and current hidden layer; U is the weight matrix connecting the inputs to the current hidden layer; $*$ is the element-wise multiplication and ignore bias term; σ is a logistic sigmoid function; $\tilde{\mathcal{C}}$ is a hidden state computed based on the current input and the previous hidden state.

4) GATED RECURRENT UNIT (GRU)

GRU has a reset (r) gate and an update gate (z) instead of the input (i), output (o), and forget (f) gates of the LSTM. GRU also has shown to be faster in training than LSTM [72].

$$\text{Update gate } z_t = \sigma\left(x_t U^z + h_{t-1}\mathcal{W}^z\right) \quad (17)$$

$$\text{Reset gate } r_t = \sigma\left(x_t U^r + h_{t-1}\mathcal{W}^r\right) \quad (18)$$

$$\text{Hidden state } \tilde{h}_t = tanh\left(x_t U^h + (r_t * h_{t-1})\mathcal{W}^h\right) \quad (19)$$

$$\text{Output hidden state } h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \quad (20)$$

LSTM (as well as GRU and RNN) comes with more complex versions such as bidirectional LSTM, autoencoder LSTM, or stacked LSTM with more RNN blocks.

5) BIDIRECTIONAL LSTM

This architecture [48] combines an LSTM that moves forward through time from the start of the sequence, with another LSTM that moves backward through time from the end of the sequence. This allows o_t to compute a representation that depends on both the past and the future, with most sensitive input gate i_t, without fixed time-step specification around t.

6) AUTOENCODER (seq2seq) LSTM

The idea is borrowed from natural language processing and video representation [73] as sequence to sequence (seq2seq) modeling of time series data. Our autoencoder LSTM model is built with two recurrent blocks (which can be LSTM or GRU) connected by repeat vector.

7) ATTENTION LSTM

The attention mechanism for sequence to sequence modeling is recently published in [49], [50], [74] for neural machine translation. The mechanism is an improvement to the model that allows it to pay attention to different words in the input sequence as it outputs each word in the output sequence. The attention LSTM model in this work is built by adding one self-attention layer with local attention to bidirectional LSTM block.

8) CONVOLUTIONAL NEURAL NETWORK (CNN)

CNNs were originally developed for working with two-dimensional image data. However, they also show effectiveness in automatic feature extraction and feature learning from sequence data. Our CNN model consists of a convolutional layer (Conv1D) that reads across the subsequence, followed by a max pooling layer that reduces to the most salient features and the output vector goes through flatten and FC layer.

9) TEMPORAL CONVOLUTIONAL NETS (TCN)

The recent combination of CNN and attention function brings a new TCN architecture [53], [75], which promises outstanding performance in speed and accuracy in natural language processing [56]. Our TCN model has a customizable structure and it is built using the keras-tcn library [76].

10) STACKED MODELS

Stacked models and stacked hidden layers make the model deeper [71]. This promises more accurate learning but at the cost of the model complexity and the cost of computing resources during training. Stacked model consists of multiple hidden layers (or blocks) where each layer contains customizable multiple memory cells, one on top of another. An example of a stacked model is stacked LSTM or stacked TCN.

C. MODEL BUILDING

Our model building process is presented in Algorithm 1 as a simplified workflow of the proactive forecasting module. The common characteristic of all of our models is that they have the same multi-channel input layer (as modeling variables) and the same multi-channel output FC layer with *sigmoid* activation. Details about the loss functions, stochastic optimization, activation functions, model quality evaluation and optimization of the learning phase are as follows.

1) LOSS FUNCTIONS

Loss functions measure the inconsistency between the ground truth vector y and the forecasting vector $\hat{y}$ of n observations. Robustness of a model increases with the decrease of the loss function value. In this work we use the mean

Algorithm 1 Model Building: Training and Validation

1 **Train** $(arch, D, R, Q)$
  Input : $arch$: DL architecture template
    D: datapool
    R: time range
    Q: data query selection
  Output : $model$
    $quality$
  Defaults: p: sequence dependency length
    k: horizon forecasting $(1 \leq k < p)$
    ∂: ODE stabilization
    m: monitoring channels
2 $R = \{R_{train}, R_{test}\}$
3 $DS = \{ds_{train}, ds_{test}\} \leftarrow Select(D, R, Q)$
4 **foreach** $ds \in DS$ **do**
5 $n \leftarrow |ds|$
6 **if** ∂ **then**
7 $ds \leftarrow ds'$ (Equation 6)
8 **end**
9 $ds \leftarrow Scale(ds) \leftarrow Normalize(ds)$
10 $ds = \begin{bmatrix} ds_{1,1}, ds_{1,2}, \cdots, ds_{1,m} \\ ds_{2,1}, ds_{2,2}, \cdots, ds_{2,m} \\ \cdots \\ ds_{n,1}, ds_{n,2}, \cdots, ds_{n,m} \end{bmatrix} = \begin{bmatrix} ds_1 \\ ds_2 \\ \cdots \\ ds_n \end{bmatrix}$
11 **for** $t \leftarrow (p + k - 1)$ **to** $n - (p + k - 1)$ **do**
12 $ts_t = [ds_{t-p+1}, \ldots, ds_{t-1}, ds_t][ds_{t+k}]$ (Equation 4)
13 $ts \leftarrow ts + ts_t$
14 **end**
15 $ds \leftarrow ts$
16 **end**
17 $model \leftarrow DefineModel(arch)$
18 $model \leftarrow Train(model, ds_{train})$
19 $quality \leftarrow Validate(model, ds_{test})$ (Equations 25, 26)
20 **return** $model, quality$
21 **while** *True* **do**
22 $R \leftarrow R + T$
23 $model \leftarrow Train(arch, D, R, Q)$ in period T
24 **end**

absolute error (MAE) (Equation 22) and the mean squared error (MSE) (Equation 21).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2 \tag{21}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y_i}| \tag{22}$$

Other metrics such as the cosine similarity *cosine* or the mean absolute percentage error (*MAPE*) are available as optional reference metrics.

2) STOCHASTIC OPTIMIZATION

Stochastic optimization is used to optimize *loss* functions. In our work, *Adam* (adaptive moment estimation) [77] is applied. Adam uses the first-order gradients to compute individual adaptive learning rates for different parameters.

3) ACTIVATION FUNCTIONS

Activation functions of the hidden units are used to capture non-linearity. Without the non-linearity, the network would not be more powerful than plain perceptrons without hidden layers. The most frequently used activation functions (also in our work) are sigmoid function, tanh, Rectified Linear Unit (ReLU) and softmax [78].

In this work, *sigmoid* (Equation 23) is used for the output layer and *ReLU* (Equation 24) is used for the hidden layers.

$$sigmoid\ \sigma(x) = \frac{1}{1 + e^{-x}} \tag{23}$$

$$ReLU\ f(x) = ln(1 + e^{x}) \tag{24}$$

D. MODEL QUALITY EVALUATION

In order to evaluate model quality with real values, the symmetric mean absolute percentage error (*SMAPE*) (Equations 25) and *cosine* similarity (Equations 26) are chosen for the main metrics. *SMAPE* formula provides results in **200% scale interpretation** instead of the frequently used 100% scale interpretation. It is less sensitive to low-volume items (i.e., near-zero) in comparison with other similar metrics such as *MAPE* [79].

$$SMAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \widehat{y_i}|}{(|y_i| + |\widehat{y_i}|)/2}100\% \tag{25}$$

$$cosine = \frac{y \cdot \widehat{y}}{\|y\| \cdot \|\widehat{y}\|} = \frac{\sum_{i=1}^{n} y_i \widehat{y_i}}{\sqrt{\sum_{i=1}^{n} y_i^2}\sqrt{\sum_{i=1}^{n} \widehat{y_i}^2}} \tag{26}$$

Other metrics such as the mean absolute percentage error (*MAPE*), the coefficient of determination R^2, and the root mean square error (*RMSE*) are also available as optional reference metrics. In this work, we use phrases ''model quality'' and ''model performance'' interchangeably.

The difference between loss functions and the model quality evaluation is the data scale dimension. Loss functions deal with y and $\widehat{y}$ in transformed *min-max* scale, while the model quality evaluation compares y and $\widehat{y}$ in the real scale. Data transformation for k-horizon forecasting (Equation 4) is done in Algorithm 1 (line 11 to 15).

E. MODEL OPTIMIZATION AND SELECTION

ML and optimisation are two important fields of AI interacting frequently with each other in order to improve learning and/or search capabilities [80]. In the context of our work, the optimization process focuses on finding the best (optimal) model from a group of alternative candidates based on the already described evaluation metrics (Section IV-D and Equations 25, 26). The most well-known approach is to perform a grid search over different parameters. More advanced optimization algorithms follow meta-heuristic direction [37]. However, they are very complex for ML/DL modeling with potentially large datasets due to

TABLE 2. Hyperparameter settings and values for experiments. The first value is default, if not stated otherwise. The number of monitoring channels and data selection query are as in Section V.

Hyperparameter	Value
memory cells of recurrent block	12
number stacked blocks	3
training epochs	50 with early stopping
learning rate	0.001
R_{train}	1 year, 6 months
R_{test}	2 months, 1 month, 3 months
w	1 hour, 10 minutes
s	10 minutes, 1 minute
datapool	w01h-s10m, w10m-s01m
k	in range <1, 12>
p	in range <6, 17>, default=12
∂	True
τ	10 minutes, 1 day
T, δ	1 month

higher computational cost. Bayesian optimization based on a sequential design strategy is used for our model behavior estimations (Section VI); i.e., hyper-parameter (Table 2) tuning in the multi-dimensional space [81].

F. LEARNING COMPLEXITY

The complexity of the production learning phase (Algorithm 1), which has to be retrained periodically in T, is

$$T_{learning} = O_{ts}(n \cdot m) + O_{model}(n, m, p) \tag{27}$$

where

- O_{ts} is the complexity of creating training data in the predefined time series format (Equation 5);
- O_{model} is the complexity of model training, which depends on the DL architecture;
- p is the sequence dependency length (Table 1);
- m is the number of monitoring channels;
- n is the number of observations in the training data.

Usually, O_{ts} does not get a large value, O_{model} is in the range of hours for one year data (Table 6) even for the most expensive DL model, and T is in the range of months for model adaptive retraining (Table 2, Section VI-A - Computing resource). Therefore, the complexity of the learning phase is not critical in production even with DL modeling.

It is suitable to remark that the learning phase is highly compute-intensive in ML development due to model optimization and selection (Section IV-E).

V. DATA PROCESSING

A. FEATURE ENGINEERING

Data processing, especially data preprocessing, is a crucial part of the whole model training process. It is periodically triggered and controlled in the *feature engineering* phase (Figure 2). Although DL has a strong feature-learning ability, in our case it still cannot learn the domain knowledge directly from all the types of data logs. Therefore, data preprocessing provides an efficient way to help deep neural networks to work properly and to provide better quality and stability of models.

It cooperates with the IDS in-situ filtering to challenge massive online data streams coming from the network monitoring as raw sensitive logs, which are continuously stored in the *data repository* (Figure 2). Based on the IDS configuration, logs of the selected protocols (connection, DNS, HTTP, SIP, SSL, and SSH) are streamed into separate log files with time-based ordered data. These files are further organized in a hierarchical directory structure by year, month and day. Each monitored network activity is recorded with various features including timestamps and unique ID.

In order to cope with off-line and intensive model development phase, state-of-the-art Big Data processing and analytic stacks are used. These comprise of Apache Spark [19], Apache Parquet [82] (columnar data storage format) and Apache Arrow [83] (cross-language development platform for columnar in-memory data). Apache Arrow also provides computational libraries and zero-copy streaming messaging and inter-process communication.

Smooth sliding transformation is an operation that helps to remove short-term variations from a series in order to reveal long-term trends [84]. Therefore, the transformation from raw logs into ML data is realized with settable window size w and sliding step s, where $s \leq w$ (Figure 3). When following this approach, dependencies among observations are reinforced. Here, the Spark SQL [85] is employed, which includes a cost-based optimizer, columnar storage and code generation to make queries fast.

B. FEATURE EXTRACTION

The feature extraction process is presented in Algorithm 2. It aims to cover already mentioned processing mechanisms in Section V. The most complex function here is the data transformation $mapping(tx^{(p)} \rightarrow tx^{(l)})$, which transforms and aggregates data (sliding window) into a logical taxonomy. The transformation is based on physical taxonomy mapping of raw logs [10]. The mapping is realized with Spark SQL [85], which enables processing of time-growing datasets at a scale.

C. FEATURE SELECTION

Due to the large number of *protocol* features that can be extracted from the datapool, data selection query (Q) is designed with descriptive syntax. The point is to obtain a narrower set of features (i.e., multiple channels for monitoring) across different *protocols* and consequently join them by sliding window #*window_start*, *window_end*. An example of such selection query is as follows:

```
  data_select_query =
conn|in_count_uid~in|out_count_uid~out;
dns|in_count_uid~in|in_distinct_query~in_distinct;
http|in;
sip|in_count_uid~in;
ssh|in;
ssl|in
#window_start,window_end
```

This query is used for experiments presented in this work (Section VI). The query syntax provides renaming manner

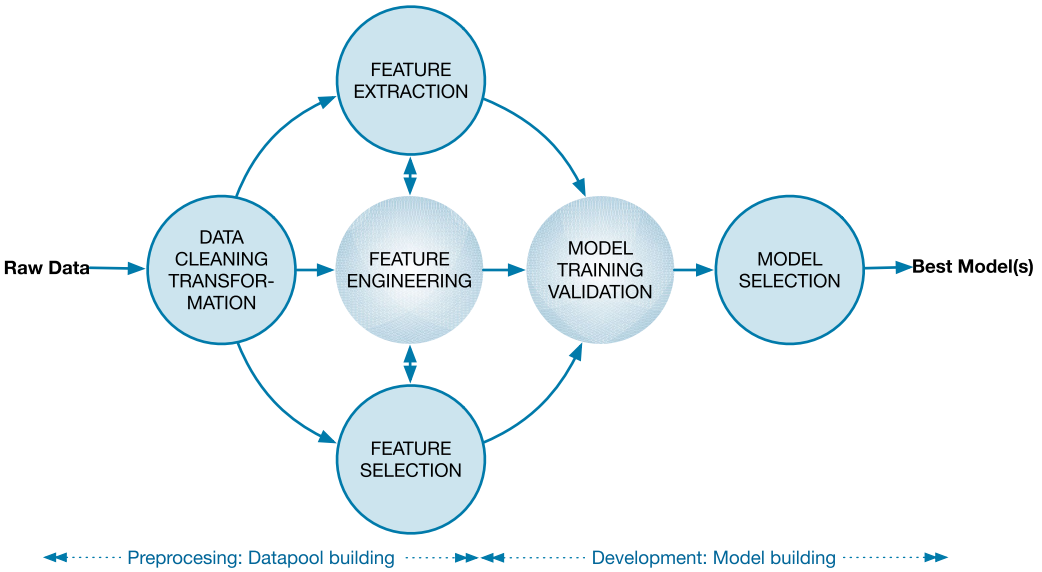


FIGURE 4. Development workflow and data preprocessing.

Algorithm 2 Data Preprocessing: Feature Extraction

Transform (D, R)
  Input : D: datapool
    R: time range $\langle begin, end\rangle$
  Output : D': datapool
  Defaults: $tx^{(p)}$: physical taxonomy of protocols
    $tx^{(l)}$: logical taxonomy of ML data
    w: window duration
    s: slide duration
  $i \leftarrow |D|$
  $beg \leftarrow R_{begin}$
  $end \leftarrow R_{end}$
  foreach $protocol \in protocols$ **do**
    $d \leftarrow \emptyset$
    while $beg < end$ **do**
      $end \leftarrow beg + w$
      $channel \leftarrow channel$ in range $\langle beg, end\rangle$
      $d_i \leftarrow \underset{tx^{(l)}_{channel} \leftarrow tx^{(p)}_{channel}}{\longleftarrow} channel$
      $d \leftarrow d + d_i$
      $beg \leftarrow beg + s$
      $i \leftarrow i + 1$
    end
    $D'_{chanel} \leftarrow D'_{chanel} + d$
  end
  return D'
while *True* **do**
  $R \leftarrow R + \tau$
  $D \leftarrow Transform(D, R)$ in period τ
end

(using $\sim$) applied on column names in the output ML dataset (D) thus a name conflict can be solved easily. Monitoring activities are classified as protocols and then for each protocol into incoming (*in*), outgoing (*out*) and internal (*internal*) flows (Figure 1 and 5). While the main interest is towards the incoming activities, outgoing and internal flows are also interesting from the anomaly monitoring point of view. In this way, data selection query defines logical taxonomy of the ML data. The query syntax makes it easy to change, extend and adapt to the needs of the ML development phase.

Features extracted in the data preprocessing phase are tested against linear correlations by matrix reduced row echelon form (RREF), intermittent demand, autocorrelation (ACF) and unit root using Augmented Dickey-Fuller (ADF) test with the critical value $\theta = \pm\frac{1.96}{\sqrt{n}}$ where n is the number of observations. Features that fail the tests are not further considered for modelling.

Non-linear feature correlations are tested using the mutual information (MI or I) for variables with continuous distribution. The MI of two random variables X and Y is a measure of the mutual dependence between them. MI has a straightforward interpretation, it is insensitive to the size of X and Y [86]. The concept of MI is linked to information entropy ($\mathcal{H}$) [87], [88] denoted as in Equation 28:

$$\mathcal{H}(X) = -\int dx\ \mu(x)\ log\mu(x) \tag{28}$$

The unit of information depends on the base of the logarithm. If the base is 2, the most used, then the information is measured in *bits*.

$$I(X,Y) = -\int\int dx\ dy\ \mu(x,y)\ log\frac{\mu(x,y)}{\mu_x(x)\ \mu_y(y)} \tag{29}$$

In case of variables with continuous value distribution, it is hard to calculate MI from real values (Equation 29). Therefore, MI estimation relies on non-parametric methods based on entropy estimation from κ-nearest neighbors distances to avoid the binning approach [89], [90]. Small values of κ reduce general systematic errors, while large κ leads to smaller statistical errors. Monitoring channels with higher non-linear correlations are selected for joint multivariate modeling.

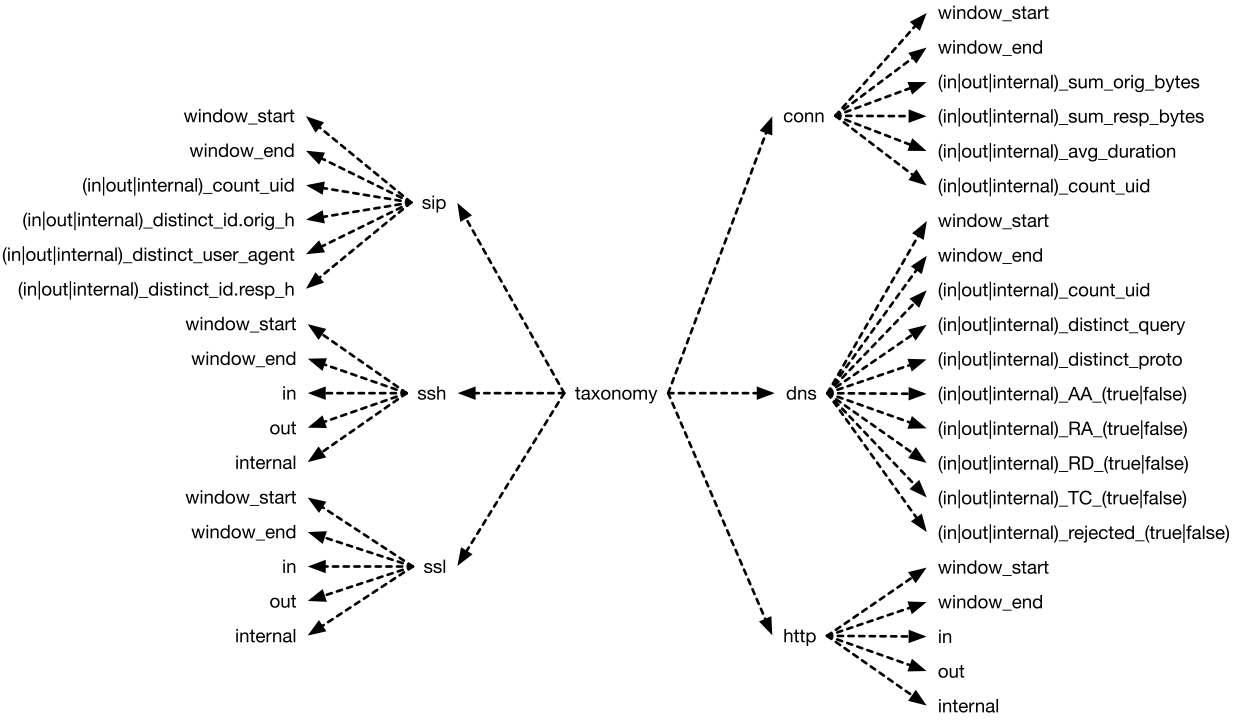


FIGURE 5. A part of the physical taxonomy of raw sensitive monitoring logs.

D. DATA PROCESSING COMPLEXITY

Data preprocessing deals with Big Data and is compute-intensive in the ML development phase. This is challenging due to exploratory data analysis natural for ML application building process, which has to be done (in addition) repeatedly. However, in production (i.e., deployment phase) in each period τ, the preprocessing (Algorithm 2) is triggered with the complexity:

$$T_{preprocessing} = protocols \cdot O(w) \tag{30}$$

where

protocols	is the number of protocols in IDS logs as in physical taxonomy (Figure 5),
w	is the window size (Table 1).

The number of *protocols* is constant [10]. In production, the period ($\tau = s$) is in 1 to 10 minutes range, which is always much bigger than $T_{preprocessing}$. Technological support and realization (Section V) ensure a fast and light-weight data transformation in deployment as well as robustness to challenge large-scale data transformation in development.

VI. EXPERIMENTS AND EVALUATIONS

A. SETUP

1) DATASET DESCRIPTION

The work presented in this paper deals with real data collected from an existing network IDS (ZEEK/Bro, as already stated in Section III), which supervises computing infrastructure network, services and activities. Collected data is available as raw logs and continuous stream with sensitive security information. Online traffic data contains approximately 2TB/day of network traffic monitoring. The data preprocessed from massive raw binary form (Section V) is continuously growing in time. Currently, its total size is 150 GB. Each day, more than 3.7 millions (10^6) events ordered in time is stored in more than 400 files. There are more than 1.35 billion (10^9) events after one year.

2) DATA TAXONOMY

Physical taxonomy of the raw collected data is described in the online ZEEK/Bro documentation [10]. ML data is prepared using the Algorithm 2 for all the evaluated DL models that are described in Section IV-B. As depicted in Figure 1, monitoring channels are divided in incoming (*in*), outgoing (*out*) and internal (*internal*) flows taking into account preference order of the system administrator. As described in Section V, they are part of protocols and built from customizable data selection queries; ML datasets for multiple channel modeling are also selected. While the main interest of the system administrator is towards incoming activities, we present experiment results towards incoming flows, especially for connections *conn* and *ssh* protocol in more details in comparison to other protocols.

3) DATAPOOLS

Data transformation (Algorithm 2) based on the data taxonomy and predefined parameter settings (Table 2) results in a datapool with time ordered data. The ML dataset for sequence modeling can be queried from the datapool by user-defined data selection queries.

4) CROSS VALIDATION

Due to the nature of time series data, the standard k-fold cross validation cannot be performed. Instead of it, we employ the

TABLE 3. Model quality evaluation with *SMAPE*(*avg*) and ODE stabilization (∂ = *True*). The best (smallest) values are highlighted.

Model	conn_in	conn_out	dns_in	dns_distinct	sip_in	http_in	ssh_in	ssl_in
MLP	3,30689	3,48848	16,03254	8,56750	34,03927	19,03174	6,73230	1,67921
CNN (Conv1D)	3,18254	3,65328	14,97823	8,70684	**31,85029**	17,16017	6,03622	1,73655
Autoencoder MLP	2,96999	3,62707	**10,79580**	**8,48765**	32,37452	**13,38382**	5,68191	**1,64501**
LSTM	3,03977	**3,31007**	14,14889	8,68462	34,59853	14,93213	6,43597	1,72878
GRU	3,12748	3,58815	14,77306	8,60974	38,88640	17,61472	6,82961	1,69887
Bidirectional LSTM	3,56821	3,50438	14,74810	8,76107	34,88928	16,67879	6,79666	1,74918
Seq2seq LSTM	**2,81782**	3,39191	24,13544	8,58021	32,79495	14,72260	5,69795	1,66890
Stacked LSTM	3,00958	3,32924	12,83822	8,58803	32,97449	15,33202	7,33681	1,70360
Attention LSTM	2,98891	3,67506	11,68680	8,56685	32,82612	13,77930	**5,48529**	1,71277
TCN	3,00981	3,53901	13,17807	8,64869	32,79507	14,72382	6,67648	1,74077

time-based hold-out validation [91], [92], for which training data has always lower time indexes than testing data. Time ranges for training and testing data are also hyper-parameters (Table 2). In this work, Bayesian optimization is chosen as the sequential strategy for optimal model selection and hyper-parameter tuning.

5) MODEL QUALITY AND MODEL STABILITY

In general, neural networks depend on the initial weights, which are initialized randomly. In addition, different random seed value will lead to inconsistent prediction performance from one training sessions to another. For this reason, the evaluation is repeated multiple times (10x in our case) to ensure model quality and stability in production. After that, average *avg* (Equation 31), standard deviation *std* (Equation 32) and coefficient of variation *cv* (Equation 33) are used to quantify the dispersion of values in resulting series $r = \{r_1, \ldots, r_n\}$.

$$avg = \frac{1}{n}\sum_{i=1}^{n} r_i = \bar{r} \tag{31}$$

$$std = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(r_i - \bar{r})^2} \tag{32}$$

$$cv = \frac{std}{\bar{r}} \tag{33}$$

6) PARAMETER SETTING

Parameter description is presented in Table 1 with default values in Table 2. The first value is the default one used in the experiments unless a different one is not explicitly declared. Libraries and source code of the DL models are defined and built as customizable architectures based on developer's settings using DL building layers and blocks with extensions.

7) LIBRARIES

The main libraries that we work with in this work are Keras and Tensorflow for DL modeling [13] and Apache Spark [19] for data preprocessing. Other python libraries for data analytic, data transformation as well as extended packages are mentioned by in-place references.

8) COMPUTING RESOURCES

For the most of our experiments, a diverse set of hardware resources with GPU is used through the DEEP-Hybrid-DataCloud project [63]. These resources are provided by distributed computing e-Infrastructures and accessed through the DEEP stack. However, runtime measurements in Section VI-B are performed on fixed hardware resources, which is Intel® Core™ i7-7700K CPU @ 4.20GHz with 32GB RAM and a NVIDIA GeForce GTX 1070 8GB GDDR5.

9) OPEN-SOURCE

Source code of the proactive forecasting module is available as a set of open-source components included in the DEEP Open Catalogue [63], [93]. The module can be downloaded in the form of a source code [94] or as a Docker container [95], which leverages the DEEPaaS API [96] and is ready to use. Due to the collected data sensitivity, raw datasets are not publicly available, but the ML datasets used for DL modeling are publicly available in [97].

B. RESULTS AND EVALUATION

In order to evaluate the intelligent module functionality, we have carried out the following experiments with the setup already described in Section VI-A:

1) Model quality evaluation by *SMAPE* of multiple monitoring channels with ODE stabilization (Table 3).
2) Model stability comparison by *SMAPE* with *std*, *cv* for *conn* and *ssh* (Table 4).
3) Model stability comparison by *cosine* with *std*, *cv* for *conn* and *ssh* (Table 5).
4) Model performance comparison by *loss* and *runtime* (Table 6).
5) *k*-horizon forecasting with LSTM (Table 8).
6) *p* sequence dependency with GRU (Table 7).
7) Model quality evaluation by *SMAPE* with *avg*, *std*, *cv* for *conn* and *ssh* without ODE stabilization (Table 9).
8) Model quality evaluation without sliding windows (presented in Section V) and without ODE stabilization (Table 10).

Based on the experimental results presented in Tables 3, 4, 5, 6, 8, 7, 9 with vizualization in Figures 6, 7, 8, 9, 10, 11 the following evaluation points are conducted:

1) QUALITY OF DL MODELS

The cooperation of data preprocessing works well with DL modeling. That ensures high quality of models by means of

TABLE 4. Model stability evaluation with *SMAPE*(*avg*, *std*, *cv*) and ODE stabilization (∂ = *True*). The best (smallest) values are highlighted.

	conn_in			conn_out			ssh_in		
Model	avg	std	cv	avg	std	cv	avg	std	cv
MLP	3,30689	0,25699	0,07771	3,48848	0,11517	0,03301	6,73230	1,38950	0,20639
CNN (Conv1D)	3,18254	0,36257	0,11392	3,65328	0,10568	0,02893	6,03622	0,89894	0,14892
Autoencoder MLP	2,96996	**0,15327**	**0,05161**	3,62707	**0,03021**	**0,00833**	5,68191	1,45777	0,25656
LSTM	3,03977	0,37795	0,12433	**3,31007**	0,24310	0,07344	6,43597	1,05100	0,16330
GRU	3,12748	0,28067	0,08974	3,58815	0,26076	0,07267	6,82961	1,09636	0,16053
Bidirectional LSTM	3,56821	0,65955	0,18484	3,50438	0,46146	0,13168	6,79666	0,93229	0,13717
Seq2seq LSTM	**2,81782**	0,18213	0,06463	3,39191	0,26181	0,07719	5,69795	0,52359	0,09189
Stacked LSTM	3,00958	0,26390	0,08769	3,32924	0,16935	0,05087	7,33681	0,79802	0,10877
Attention LSTM	2,98891	0,18143	0,06070	3,67506	0,09808	0,02669	**5,48529**	**0,34495**	**0,06289**
TCN	3,00981	0,19892	0,06609	3,53901	0,32197	0,09098	6,67648	0,88727	0,13289

TABLE 5. Model stability evaluation with cosine (*avg*, *std*, *cv*) and ODE stabilization (∂ = *True*). The best values are highlighted.

	conn_in			conn_out			ssh_in		
Model	avg	std	cv	avg	std	cv	avg	std	cv
MLP	0,99373	0,00091	0,00092	0,99548	0,00102	0,00103	0,99483	0,00079	0,00080
CNN (Conv1D)	0,99329	0,00024	0,00024	0,99417	0,00027	0,00027	0,99552	0,00007	0,00007
Autoencoder MLP	0,99334	**0,00004**	**0,00004**	0,99399	**0,00001**	**0,00001**	0,99562	0,00017	0,00017
LSTM	0,99517	0,00045	0,00045	0,99604	0,00046	0,00047	0,99372	0,00258	0,00260
GRU	0,99482	0,00065	0,00065	0,99516	0,00064	0,00064	0,99435	0,00095	0,00095
Bidirectional LSTM	0,99477	0,00050	0,00050	0,99573	0,00084	0,00084	0,99401	0,00139	0,00140
Seq2seq LSTM	0,99362	0,00021	0,00021	0,99463	0,00064	0,00065	0,99556	0,00018	0,00018
Stacked LSTM	**0,99519**	0,00020	0,00020	**0,99606**	0,00036	0,00036	0,99311	0,00137	0,00138
Attention LSTM	0,99334	0,00005	0,00005	0,99398	0,00003	0,00003	**0,99568**	**0,00003**	**0,00003**
TCN	0,99373	0,00070	0,00070	0,99401	0,00092	0,00092	0,99489	0,00079	0,00079

TABLE 6. Model evaluation with *loss*(*avg*) and *runtime*(*seconds*). The best values are highlighted.

	loss		runtime		
Model	MSE	MAE	avg	std	cv
MLP	0,00044	0,00636	2468,20	1015,01	0,41124
CNN (Conv1D)	0,00049	0,00642	**2324,32**	1223,81	0,52652
Autoencoder MLP	0,00048	0,00611	3351,42	780,06	0,23276
LSTM	**0,00041**	0,00603	5995,33	426,41	0,07112
GRU	0,00044	0,00640	6179,18	437,67	0,07083
Bidirectional LSTM	0,00043	0,00637	8810,88	1248,39	0,14169
Seq2seq LSTM	0,00046	0,00610	8064,29	3095,95	0,38391
Stacked LSTM	0,00040	**0,00599**	7730,66	**234,34**	**0,03031**
Attention LSTM	0,00048	0,00618	5508,47	1257,57	0,22830
TCN	0,00048	0,00626	14812,71	7260,50	0,49015

low prediction error (*SMAPE* and *cosine*) and better model stability (low *std* and *cv*).

- Although there is not a clear winning model of the experiments, LSTM derivates (including GRU) are the best candidates regarding the quality with comparatively similar performance.
- The best top-3 models, which work properly with our domain data, are Attention LSTM, LSTM and GRU (unordered).
- Shallower models like MLP or Conv1D are less stable in prediction as well as in runtime.
- Autoencoder with LSTM (Seq2seq LSTM) provides better prediction in comparison to Autoencoder MLP.

2) ATTENTION LSTM AND TCN

Attention LSTM and TCN models are currently in the center of DL research community. Alhough these models were originally designed for neural machine translation, their outstanding feature is that they can deal with the variability of sequences. This feature is not yet fully exploited by our domain data (Equation 5), but we observe the following results so far:

- Attention LSTM often in the top-3 rank of prediction quality. In many cases its runtime performance is comparable to or better than other LSTM variances.
- TCN is a complex model with the longest runtime (Table 6). It has significantly better quality than MLP, Con1D and Autoencoder MLP, but worse than LSTM variances (Table 9).

3) IMPACT OF DATA PREPROCESSING ON MODEL QUALITY

- All the models have obviously worse performance with less proper data preprocessing (Figure 7).

TABLE 7. ***p*** **sequence dependency with GRU: *SMAPE*(*avg*), 6 month train data, 2 month test data. The best values (smallest) are highlighted.**

	SMAPE								runtime
p	conn_in	conn_out	dns_in	dns_distinct	sip_in	http_in	ssh_in	ssl_in	ratio
6	2,68176	2,87414	**13,46127**	8,68940	35,08185	14,25741	5,86626	1,81448	**1,00000**
7	2,80451	2,93097	17,39224	**8,57037**	37,44951	14,87755	**5,78405**	1,72665	1,02970
8	2,84178	**2,86972**	14,91492	8,67455	34,73789	15,13111	6,20911	1,73243	1,07274
9	2,68298	2,88613	16,89135	8,63581	**33,56974**	14,47731	5,88191	1,75550	1,11105
10	2,71607	2,91707	15,00933	8,57480	37,94110	14,11984	6,06302	1,76346	1,14879
11	2,74095	3,03752	27,20190	8,61486	35,98750	14,99586	6,97941	1,74777	1,19576
12	2,97787	2,96635	15,26157	8,78358	34,98217	13,38596	6,94083	2,00350	1,23507
13	2,66391	3,14315	20,08388	8,65449	34,66176	15,10253	6,94758	**1,67933**	1,27015
14	2,74310	3,10622	14,65596	8,63084	36,15420	13,70668	6,26418	1,69673	1,30134
15	2,69094	3,19307	14,27943	8,59324	35,82219	13,16589	6,93777	1,71214	1,32183
16	**2,64484**	3,07861	24,50614	8,65319	34,94473	**13,01539**	6,78339	1,71071	1,35017
17	3,02483	3,27625	18,64653	8,61169	35,92165	14,18568	6,81141	1,71543	1,37452

TABLE 8. ***k*****-horizon forecasting with LSTM: *SMAPE*(*avg*), 6 month train data, 2 month test data. The best values (smallest) are highlighted.**

k	conn_in	conn_out	dns_in	dns_distinct	sip_in	http_in	ssh_in	ssl_in
1	**2,63939**	3,06290	17,84072	8,57357	41,60133	13,31360	6,72937	1,70070
2	2,75136	3,17429	22,26940	8,53085	36,24915	14,44386	7,22286	1,65533
3	2,68694	3,19830	16,25056	8,64921	**32,69475**	13,65866	6,14218	1,75209
4	2,74368	3,06348	22,43242	8,56534	35,60060	13,96215	6,19500	1,73902
5	2,69967	**3,03292**	14,02392	8,63619	35,76468	**12,55945**	6,71858	1,78011
6	3,08741	3,23188	16,33750	8,55518	35,43627	15,49413	6,60666	1,86559
7	3,15067	3,66348	17,76897	8,54153	33,72200	15,78865	6,35456	1,67351
8	3,05914	3,61004	**12,07876**	8,51715	33,03453	13,36550	5,54008	1,72042
9	2,95561	3,64672	19,88710	8,51214	36,19762	13,66934	**5,36231**	**1,63297**
10	2,97907	3,61976	13,26928	8,54730	33,75210	14,55015	6,02440	1,70097
11	3,03026	3,65715	13,79529	8,52022	34,96663	14,14280	5,81025	1,75997
12	2,97433	3,64114	13,74140	**8,48913**	33,71838	14,62030	5,78196	1,68214

TABLE 9. **Model evaluation with *SMAPE*(*avg*, *std* , *cv*) without ODE stabilization ($\partial = False$). The best values are highlighted.**

	conn_in			conn_out			ssh_in		
Model	avg	std	cv	avg	std	cv	avg	std	cv
MLP	8,33598	2,71211	0,32535	6,78544	0,86267	0,12714	34,68881	4,81358	0,13876
CNN (Conv1D)	8,01503	3,28656	0,41005	5,84616	1,18225	0,20223	19,35134	3,16984	0,16380
Autoencoder MLP	10,05656	2,12057	0,21086	9,39986	1,00517	0,10693	51,03975	3,25719	**0,06382**
LSTM	4,13511	0,40043	0,09684	4,50222	0,55422	0,12310	17,12519	**1,39404**	0,08140
GRU	4,53315	0,90838	0,20039	4,68212	0,63814	0,13629	18,16634	3,19159	0,17569
Bidirectional LSTM	**4,03790**	**0,25879**	**0,06409**	4,57340	0,44316	0,09690	15,75507	2,36393	0,15004
Seq2seq LSTM	4,10700	0,47024	0,11450	4,83108	0,39899	0,08259	18,95165	2,93552	0,15490
Stacked LSTM	4,49102	1,23528	0,27506	**4,34981**	**0,25943**	**0,05964**	16,07973	1,75108	0,10890
Attention LSTM	4,25764	0,46213	0,10854	4,76331	2,01536	0,42310	**14,37249**	2,32580	0,16182
TCN	5,07388	1,53888	0,30330	4,80786	0,83467	0,17360	18,69377	3,18486	0,17037

TABLE 10. **Model performance with tumbling windows, $w = 10m$, $s = 10m$, without ODE stabilization ($\partial = False$). The best values are highlighted.**

	SMAPE			*cosine*		
Model	conn_in	conn_out	ssh_in	conn_in	conn_out	ssh_in
MLP	19,54993	15,28342	50,19192	0,89120	0,93162	0,71061
CNN (Conv1D)	18,34072	14,70530	42,83442	0,84640	**0,94310**	0,63938
Autoencoder MLP	17,93147	18,12417	48,74830	0,89945	0,92651	0,83514
LSTM	13,79754	15,51034	39,52140	**0,92571**	0,91807	0,82976
GRU	13,22399	15,09612	43,09676	0,92313	0,91676	0,74296
Bidirectional LSTM	**11,97487**	14,43501	35,18824	0,92236	0,91652	0,86901
Seq2seq LSTM	12,48779	14,08209	42,10278	0,92163	0,91805	0,86129
Stacked LSTM	13,28014	15,96957	45,36115	0,91789	0,91479	0,67036
Attention LSTM	13,51417	**13,95289**	**32,10520**	0,92043	0,92583	**0,90188**

- Model quality is in a narrow range with proper data preprocessing for $\partial = True$. Table 3 shows the power of the data preprocessing, which provides the best quality ML data.

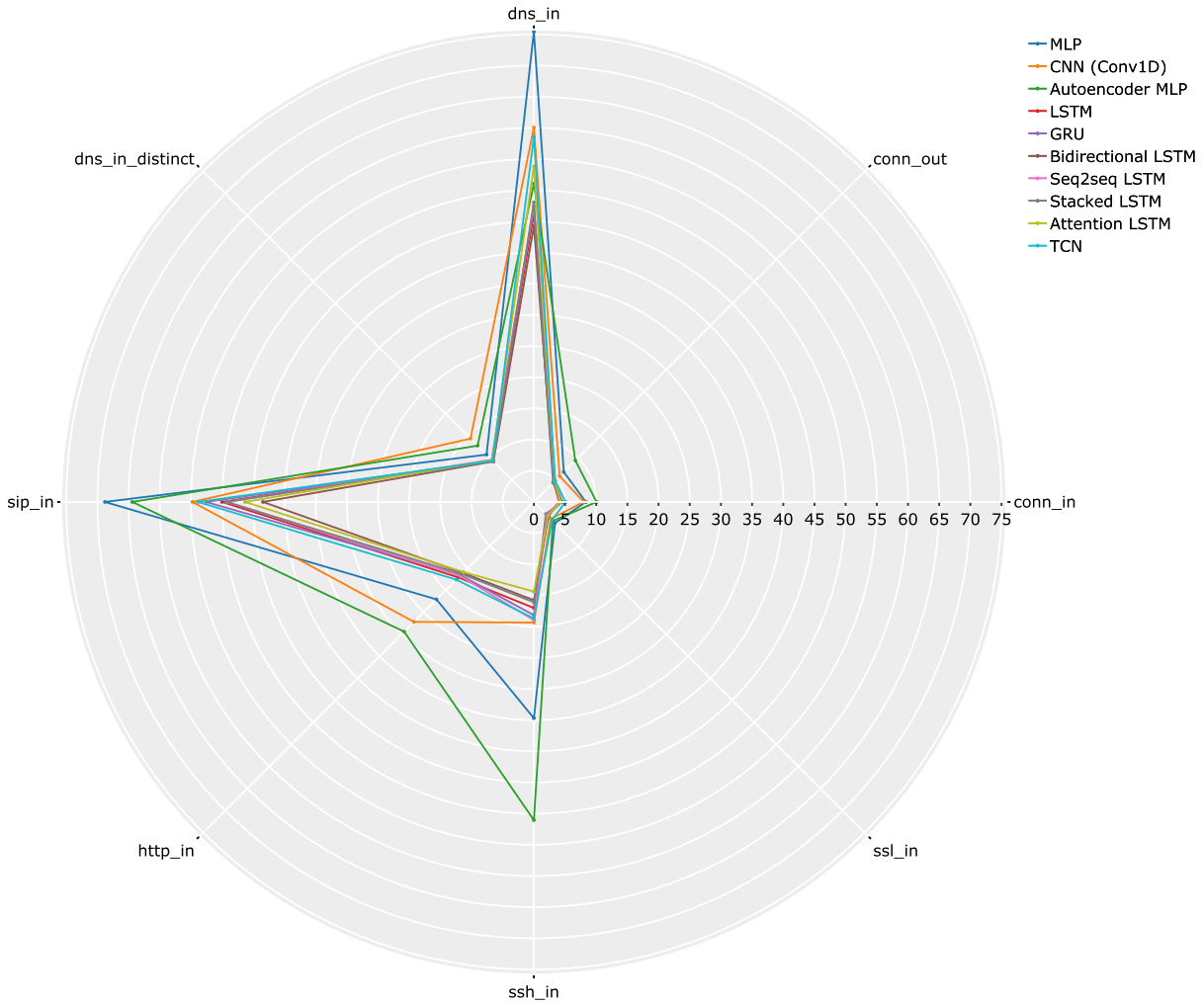


FIGURE 6. Model quality illustration based on results in Table 9 with $\partial = False$.

- Table 9 and Figure 6 with $\partial = False$ (i.e., less proper data preprocessing) provide error-prone insight into model quality.
- The comparison of $\partial = True$, $\partial = False$ and tumbling windows effect in prediction quality is depicted in Figure 7, which also supports the previous conclusion on cooperation of data preprocessing and DL modeling.

Figure 10 and Figure 11 depict predictions over two days of unseen data with Attention LSTM model with and without ODE stabilization (∂). All models have better performance and stability for $\partial = True$ (Table 4) in comparison to $\partial = False$ (Table 9).

4) HYPER-PARAMETERS k-HORIZON AND p SEQUENCE DEPENDENCY

- k-horizon forecasting parameter is important for the near future forecasting. k defines the horizon as k^{th} sliding step from the time t. Table 8 with visualization in Figure 9 show the stability of our models with parameter conditions as described in Table 1.
- p sequence dependency length influences the modelling complexity as described in Algorithm 1 and analyzed in Section III-C. The range ($6 \leq p < 18$) is fixed based on exploratory data analysis and feature selection tests (SectionV) for the experiment data. Table 7 and Figure 8 show increasing training time in correlation with increasing value of p.

5) MODEL PERFORMANCE FROM THE SPEED POINT OF VIEW

Apart from the modelling performance regarding the accuracy, the speed factor is also interesting and useful. Table 6 presents runtime comparison of the models with default settings in Table 2.

- In general, the runtime is quite stable with low cv values for LSTM variances including GRU. The exception is Seq2seq LSTM, which is the 4^{th} worst model out of 10 models in runtime stability.
- MLP, Conv1D and TCN have high cv values not only in accuracy ($SMAPE$) but also in runtime.

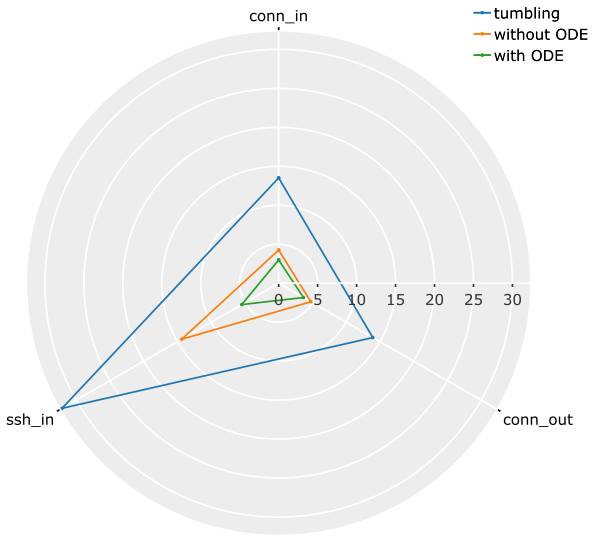


FIGURE 7. Comparison of $\partial = True$ (with ODE), $\partial = False$ (without ODE) and tumbling windows effect on prediction quality with Attention LSTM model.

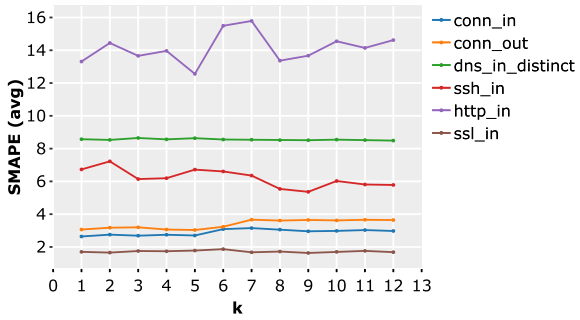


FIGURE 8. Runtime ratio for p sequence dependency with GRU model, 6 month train data, 2 month test data.

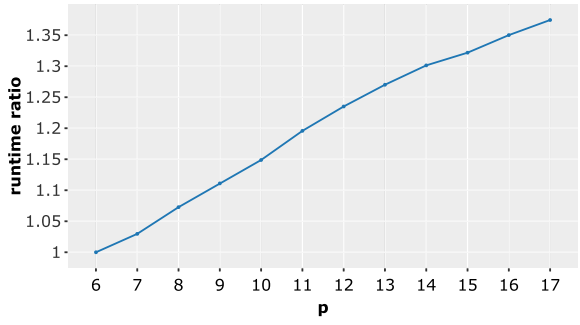


FIGURE 9. k-horizon forecasting with LSTM model: $SMAPE(avg)$, 6 month train data, 2 month test data.

- Attention LSTM has one more layer (attention layer) in comparison to LSTM. However, its average runtime is shorter in our experiments with early stopping setting.
- Theoretically, as presented in Section IV-B, GRU is slightly faster than LSTM. The early stopping effect also causes slightly vice-versa runtime results in our experiments.
- It is clear that deeper models like seq2seq LSTM, stacked LSTM and TCN have higher complexity, which reflects in longer runtime. Simpler models like MLP, Conv1D and Autoencoder MLP are faster in training;
- From the balance viewpoint between prediction quality and model complexity, models like Bidirectional LSTM, Seq2seq LSTM, TCN including stacked TCN are too expensive.

6) PREDICTIVE POWER OF MONITORING CHANNEL DATA SERIES

As ML/DL models learn from data, various monitoring channels have various prediction quality as presented in (Table 3) or Table 10.

- TCN is the most sensitive to intermittent demand especially with *sip* protocol; TCN performance is not included in Table 10.
- *sip_in* can be modeled for prediction with proper data preprocessing. The decision of feature selection is done as described in Section V.

However, data characteristics evaluation depends on collected data by the specific monitoring site. It can be changed when applicable to other sites; i.e., with another network traffic characteristics.

7) PAYLOAD MONITORING

Exploratory data analysis was also performed for payload prediction modeling, when multiple monitoring channels were used; i.e., features extracted from the datapool. The results show that from 4 extracted features —*in_sum_orig*, *in_sum_resp*, *out_sum_orig*, and *out_sum_resp*—only the last one, which indicates mainly the web server payload traffic, can be modeled with good results (*SMAPE* from 6% to 8% with LSTM model). Payload features are extracted as a volume summaries of sent (*orig*) and received (*resp*) data in combination with data flow direction (*in* and *out*). They indicate data volume transfer in interaction with computing infrastructure services and its users. The reason of such results is the high stochastic character of download and upload processes of the monitored computing infrastructure. The exploratory data analysis diagram shows near-zero activity for long periods with very high peaks; i.e., some users download or upload large data volumes without any predictable behavior. From the anomaly detection point of view, network administrators get warnings about high peaks based on a simple rule in the monitoring system. Therefore, there is no need for sophisticated modeling of simple cases like the one described.

8) FINAL LOOK BACK ON EXPERIMENTS

In Figure 10, the model is Attention LSTM with forecasting horizon of one step in advance (10 minutes) and data preprocessing with ODE. Although it seems from the graph that the prediction (dashed line) adheres the real line, there are differences where it deflects from the real values. These differences are numerically calculated (Section IV-D) and presented in Table 3. With a less proper data preprocessing

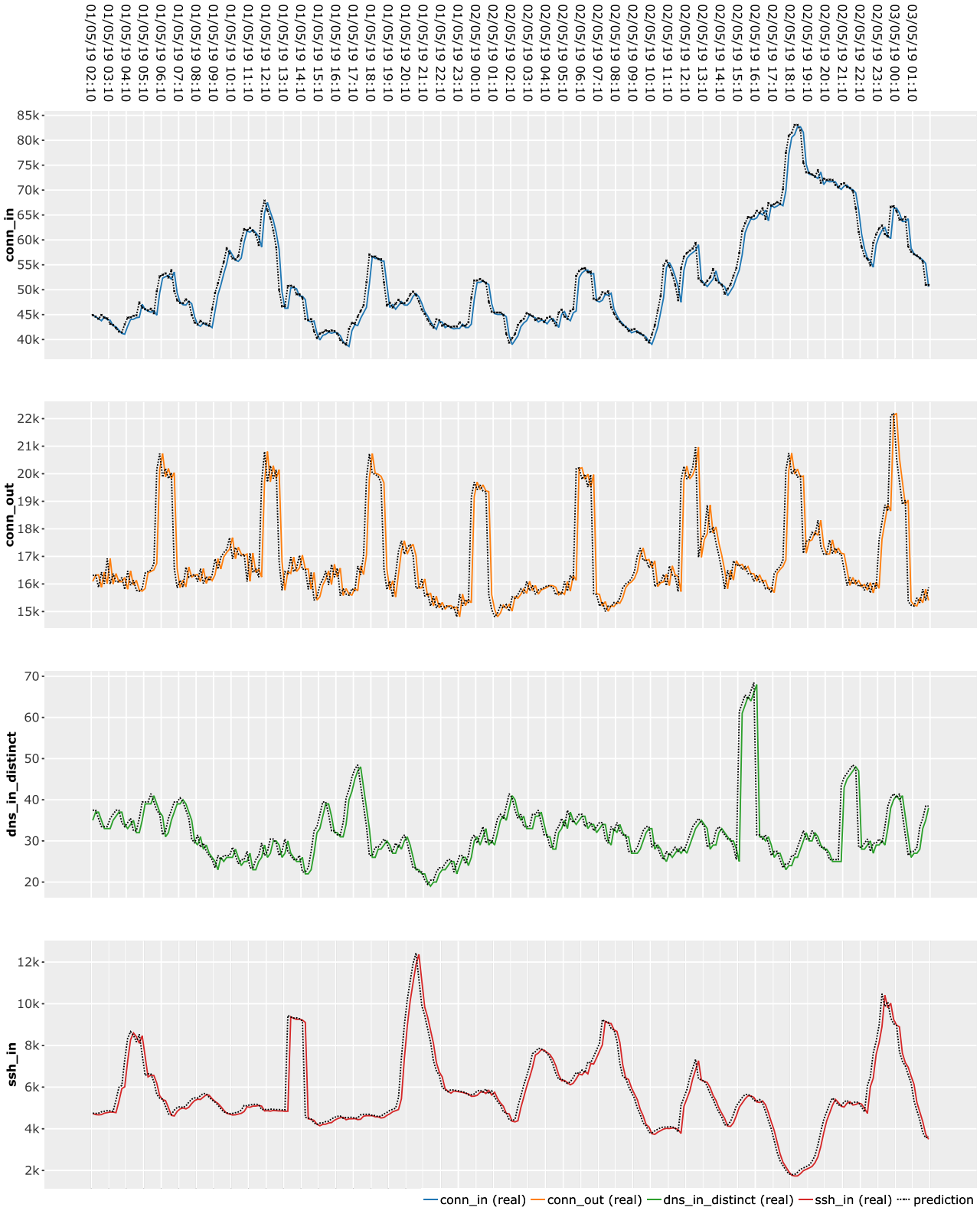


FIGURE 10. Prediction (dashed) with Attention LSTM, ∂ = *True*, 3 months train data, prediction 1 step forward (10 min.) over 2 days of unseen data.

(without ODE), the prediction (dashed line) adheres less with the real line, which is reflected in worse model stability (Figure 11) as numerically presented in Table 9. Based on the results presented in Table 3, Table 9 and Table 10 the effect of preprocessing quality (with ODE, without ODE and tumbling windows) on forecasting quality is depicted in Figure 7.

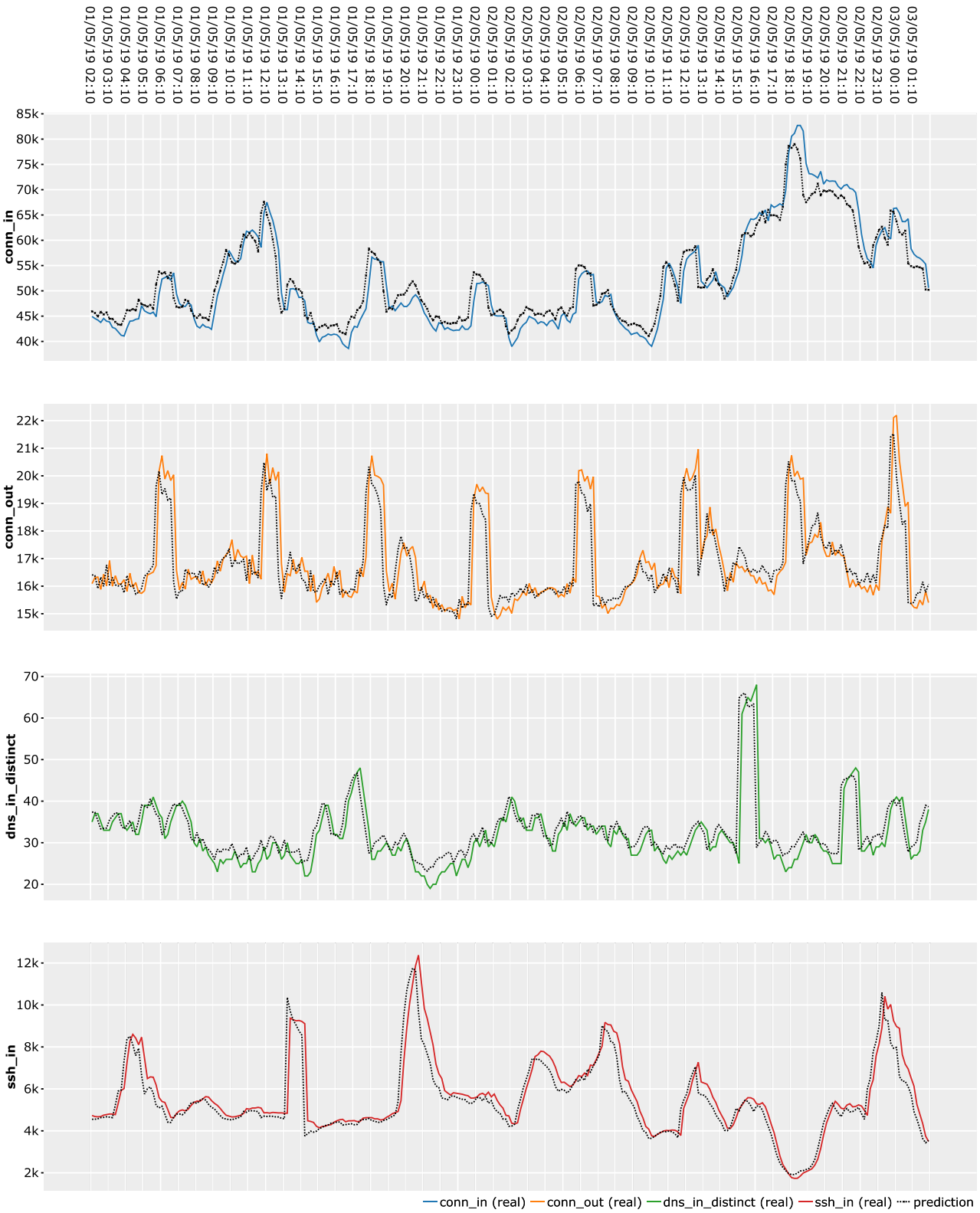


FIGURE 11. Prediction (dashed) with Attention LSTM, ∂ = *False*, 3 months train data, prediction 1 step forward (10 min.) over 2 days of unseen data.

VII. CONCLUSION AND PERSPECTIVE

The work presented in this paper provides a deep insight into the building process of an intelligent module for proactive network monitoring and security protection for computing infrastructures. The intelligent module extends the monitoring capacity of the IDS supervising network traffic flows through a proactive solution using DL techniques, simultaneously modelling multiple monitoring channels.

We presented the system's architecture, covering both the development and deployment phases of the module as ML application. The proper connection of DL modelling with scalable data preprocessing ensures high quality model performance in dynamic and fast-changing environments. The data preprocessing module leverages Big Data technologies to deliver feature engineering automation at scale. The proposed solution is carefully evaluated with experiments and presented in Section VI-B.

If there is not an universal solution that solves all the problems, it is suitable to combine several partial into a hybrid solution, where each partial solution can be strong and suitable for fitting different cases (Section III-C). A considerable amount of work needs to be done to deliver an AI solution to a real production environment, in close cooperation with network administrators, in order to cover security incident recognizing, abnormal range exclusion specification (Table 1) as well as feed-backs for next level detection.

ACKNOWLEDGMENT

The computing and storage resources used in our work are obtained through the DEEP-Hybrid-DataCloud services. The authors would like to thank all colleagues and partners for sharing knowledge and for technological support. They also would like to thank V. Y. Kozlov from the Karlsruhe Institute of Technology (KIT), Steinbuch Centre for Computing (SCC) for proofreading.

REFERENCES

[1] ENISA. (2019). *The European Union Agency for Network and Information Security—ENISA Threat Landscape Report 2018*. Accessed: Sep. 9, 2019. [Online]. Available: https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2018

[2] Cisco. (2017). *Cisco 2017 Midyear Cybersecurity Report Predicts New, Destruction of Service Attacks; Scale and Impact of Threats Grow*. Accessed: Sep. 9, 2019. [Online]. Available: https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1868060

[3] S. Dua and X. Du, *Data Mining and Machine Learning in Cybersecurity*. New York, NY, USA: Auerbach, Apr. 2016.

[4] CyberHub. (2019). *6 Steps to Cyber Security*. Accessed: Sep. 9, 2019. [Online]. Available: https://www.cyberhubengage.com/6-steps

[5] M. R. Endsley, *Designing for Situation Awareness: An Approach to User-Centered Design*, 2nd ed. Boca Raton, FL, USA: CRC Press, 2016.

[6] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *J. Netw. Comput. Appl.*, vol. 60, pp. 19–31, Jan. 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1084804515002891

[7] D. K. Bhattacharyya and J. K. Kalita, *Network Anomaly Detection: A Machine Learning Perspective*. London, U.K.: Chapman & Hall, Jun. 2013.

[8] H.-J. Liao, C.-H. Richard Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *J. Netw. Comput. Appl.*, vol. 36, no. 1, pp. 16–24, Jan. 2013. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1084804512001944

[9] (2019). *Introduction to Intrusion Prevention Systems—Detect and Block Attacks in Real Time*. Accessed: Sep. 9, 2019. [Online]. Available: https://www.ibm.com/developerworks/library/se-intrusion/

[10] Zeek/Bro. (2019). *The Zeek Network Security Monitor*. Accessed: Sep. 9, 2019. [Online]. Available: https://www.zeek.org/

[11] Nessus. (2019). *Nessus Professional—Vulnerability Scanner—Tenable*. Accessed: Sep. 9, 2019. [Online]. Available: https://www.tenable.com/products/nessus/nessus-professional

[12] Zabbix. (2019). *Zabbix Documentation: Predictive Trigger Functions*. Accessed: Sep. 9, 2019. [Online]. Available: https://www.zabbix.com/documentation/current/manual/config/triggers/prediction

[13] G. Nguyen, S. Dlugolinsky, M. Bobák, V. Tran, L. L. García, I. Heredia, P. Malík, and L. Hluchý, "Machine learning and deep learning frameworks and libraries for large-scale data mining: A survey," *Artif. Intell. Rev.*, vol. 52, no. 1, pp. 77–124, Jun. 2019. [Online]. Available: http://link.springer.com/10.1007/s10462-018-09679-z

[14] A. Cano, "A survey on graphic processing unit computing for large-scale data mining," *WIREs Data Mining Knowl. Discovery*, vol. 8, no. 1, p. e1232, Jan. 2018, doi: 10.1002/widm.1232.

[15] X. Wang, L. Ma, B. Wang, and T. Wang, "A hybrid optimization-based recurrent neural network for real-time data prediction," *Neurocomputing*, vol. 120, pp. 547–559, Nov. 2013. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0925231213004876

[16] X. Ma, Z. Tao, Y. Wang, H. Yu, and Y. Wang, "Long short-term memory neural network for traffic speed prediction using remote microwave sensor data," *Transp. Res. C, Emerg. Technol.*, vol. 54, pp. 187–197, May 2015. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0968090X15000935

[17] K. Chen, S. Pashami, Y. Fan, and S. Nowaczyk, "Predicting air compressor failures using long short term memory networks," in *Proc. EPIA Conf. Artif. Intell.* Cham, Switzerland: Springer, 2019, pp. 596–609. [Online]. Available: https://link.springer.com/chapter/10.1007%2F978-3-030-30241-2_50

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (Adaptive Computation and Machine Learning Series). Cambridge, MA, USA: MIT Press, 2016.

[19] A.S. Foundation. (2019). *Apache Spark—Unified Analytics Engine for Big Data*. Accessed: Sep. 9, 2019. [Online]. Available: https://spark.apache.org/

[20] Snort. (2019). *Snort—Network Intrusion Detection and Prevention System*. Accessed: Sep. 9, 2019. [Online]. Available: https://www.snort.org/

[21] Suricata. (2019). *Suricata | Open Source IDS/IPS/NSM Engine*. Accessed: Sep. 9, 2019. [Online]. Available: https://suricata-ids.org/

[22] Zabbix. (2019). *Monitor Anything Zabbix—Solutions for any Kind of it Infrastructure, Services, Applications, Resources*. Accessed: Sep. 9, 2019. [Online]. Available: https://www.zabbix.com

[23] Ossec. (2019). *Ossec—World's Most Widely Used Host Intrusion Detection System*. Accessed: Sep. 9, 2019. [Online]. Available: https://www.ossec.net/

[24] Rapid7. (2019). *Rapid7 Secure Advancement*. Accessed: Sep. 9, 2019. [Online]. Available: https://www.rapid7.com/

[25] B. M. Nguyen, D. Tran, and G. Nguyen, "Enhancing service capability with multiple finite capacity server queues in cloud data centers," *Cluster Comput*, vol. 19, no. 4, pp. 1747–1767, Dec. 2016. [Online]. Available: http://link.springer.com/10.1007/s10586-016-0653-y

[26] S. Dlugolinsky, G. Nguyen, M. Seleng, and L. Hluchy, "Decision influence and proactive sale support in a chain of convenience stores," in *Proc. Int. Conf. Intell. Eng. Syst.*, 2017, pp. 277–284. [Online]. Available: http://ieeexplore.ieee.org/document/8118570/

[27] J. D. Hamilton, *Time Series Analysis*, vol. 2. Princeton, NJ, USA: Princeton Univ. Press, 1994.

[28] T.-C. Fu, "A review on time series data mining," *Eng. Appl. Artif. Intell.*, vol. 24, no. 1, pp. 164–181, Feb. 2011. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0952197610001727

[29] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "Statistical and machine learning forecasting methods: Concerns and ways forward," *PLOS ONE*, vol. 13, no. 3, Mar. 2018, Art. no. e0194889. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0194889

[30] M. Baptista, S. Sankararaman, I. P. de Medeiros, C. Nascimento, H. Prendinger, and E. M. Henriques, "Forecasting fault events for predictive maintenance using data-driven techniques and arma modeling," *Comput. Ind. Eng.*, vol. 115, pp. 41–53, Jan. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S036083521730520X

[31] A.-C. Petrică, S. Stancu, and A. Tindeche, "Limitation of ARIMA models in financial and monetary economics," *Theor. Appl. Econ.*, vol. 23, no. 4, pp. 19–42, 2016. [Online]. Available: http://store.ectap.ro/articole/1222.pdf

[32] V. Cerqueira, L. Torgo, and C. Soares, "Machine learning vs statistical methods for time series forecasting: Size matters," 2019, *arXiv:1909.13316*. [Online]. Available: https://arxiv.org/abs/1909.13316

[33] Y. Kajitani, K. W. Hipel, and A. I. Mcleod, "Forecasting nonlinear time series with feed-forward neural networks: A case study of Canadian LYNX data," *J. Forecasting*, vol. 24, no. 2, pp. 105–117, Mar. 2005, doi: 10.1002/for.940.

[34] M. Tahan, E. Tsoutsanis, M. Muhammad, and Z. A. Karim, "Performance-based health monitoring, diagnostics and prognostics for condition-based maintenance of gas turbines: A review," *Appl. Energy*, vol. 198, pp. 122–144, Jul. 2017. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0306261917304415

[35] B. Mohammed, I. Awan, H. Ugail, and M. Younas, "Failure prediction using machine learning in a virtualised HPC system and application," *Cluster Comput.*, vol. 22, no. 2, pp. 471–485, 2018. [Online]. Available: https://link.springer.com/article/10.1007/s10586-019-02917-1

[36] D. Tran, N. Tran, G. Nguyen, and B. M. Nguyen, "A proactive cloud scaling model based on fuzzy time series and SLA awareness," *Procedia Comput. Sci.*, vol. 108, pp. 365–374, 2017. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1877050917306865

[37] M. Abdel-Basset, L. Abdel-Fatah, and A. K. Sangaiah, "Metaheuristic algorithms: A comprehensive review," in *Computational Intelligence for Multimedia Big Data on the Cloud With Engineering Applications*. Amsterdam, The Netherlands: Elsevier, 2018, pp. 185–231. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9780128133149000104

[38] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[39] J. Kennedy, "Particle swarm optimization," *Encyclopedia Mach. Learn.*, vol. 93, pp. 760–766, Jan. 2010. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-0-387-30164-8_630.pdf

[40] S. Salcedo-Sanz, J. Del Ser, I. Landa-Torres, S. Gil-López, and J. Portilla-Figueras, "The coral reefs optimization algorithm: A novel metaheuristic for efficiently solving optimization problems," *Sci. World J.*, vol. 2014, Jul. 2014, Art. no. 739768, doi: 10.1155/2014/739768.

[41] V. Muthiah-Nakarajan and M. M. Noel, "Galactic swarm optimization: A new global optimization metaheuristic inspired by galactic motion," *Appl. Soft Comput.*, vol. 38, pp. 771–787, Jan. 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1568494615006742

[42] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Adv. Eng. Softw.*, vol. 95, pp. 51–67, May 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0965997816300163

[43] T. Nguyen, B. M. Nguyen, and G. Nguyen, "Building resource auto-scaler with functional-link neural network and adaptive bacterial foraging optimization," in *Theory and Applications of Models of Computation* (Lecture Notes in Computer Science), vol. 11436. Cham, Switzerland: Springer, 2019, pp. 501–517. [Online]. Available: http://link.springer.com/10.1007/978-3-030-14812-6_31

[44] T. Nguyen, T. Nguyen, B. M. Nguyen, and G. Nguyen, "Efficient time-series forecasting using neural network and opposition-based coral reefs optimization," *Int. J. Comput. Intell. Syst.*, vol. 12, no. 2, p. 1144, 2019. [Online]. Available: https://www.atlantis-press.com/article/125921354

[45] J. C. B. Gamboa, "Deep learning for time-series analysis," 2017, *arXiv:1701.01887*. [Online]. Available: https://arxiv.org/abs/1701.01887

[46] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[47] D. Bruneo and F. De Vita, "On the use of LSTM networks for predictive maintenance in smart industries," in *Proc. IEEE Int. Conf. Smart Comput.*, Jun. 2019, pp. 241–248. [Online]. Available: https://ieeexplore.ieee.org/document/8784003

[48] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, 1997. [Online]. Available: http://ieeexplore.ieee.org/document/650093/

[49] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2015, pp. 1412–1421. [Online]. Available: http://aclweb.org/anthology/D15-1166

[50] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008. [Online]. Available: http://papers.nips.cc/paper/7181-attention-is-all-you-need

[51] T. Hollis, A. Viscardi, and S. E. Yi, "A comparison of LSTMs and attention mechanisms for forecasting financial time series," 2018, *arXiv:1812.07699*. [Online]. Available: https://arxiv.org/abs/1812.07699

[52] N. Tran, T. Nguyen, B. M. Nguyen, and G. Nguyen, "A multivariate fuzzy time series resource forecast model for clouds using LSTM and data correlation analysis," *Procedia Comput. Sci.*, vol. 126, pp. 636–645, 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1877050918312754

[53] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 2018, *arXiv:1803.01271*. [Online]. Available: https://arxiv.org/abs/1803.01271

[54] S.-Y. Shih, F.-K. Sun, and H.-Y. Lee, "Temporal pattern attention for multivariate time series forecasting," *Mach. Learn.*, vol. 108, nos. 8–9, pp. 1421–1441, Sep. 2019. [Online]. Available: http://link.springer.com/10.1007/s10994-019-05815-0

[55] D. Li, D. Chen, B. Jin, L. Shi, J. Goh, and S.-K. Ng, "MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks," in *Proc. Int. Conf. Artif. Neural Netw.* Cham, Switzerland: Springer, 2019, pp. 703–716, doi: 10.1007/978-3-030-30490-4_56.

[56] W. Vorhies. (2018). *Temporal Convolutional Nets (TCNs) Take Over From RNNs for NLP Predictions*. Accessed: Sep. 9, 2019. [Online]. Available: https://www.datasciencecentral.com/profiles/blogs/temporal-convolutional-nets-tcns-take-over-from-rnns-for-nlp-pred

[57] J. M. Kizza, *Guide to Computer Network Security* (Computer Communications and Networks), 4th ed. London, U.K.: Springer-Verlag, 2017. [Online]. Available: https://link.springer.com/book/10.1007/978-3-319-55606-2

[58] NVIDIA. (2019). *High Performance Computing. Build Scalable GPU-Accelerated Applications. Faster*. Accessed: Sep. 9, 2019. [Online]. Available: https://developer.nvidia.com/hpc

[59] Google. (2019). *Tensor Processing Unit (TPU)*. Accessed: Oct. 9, 2019. [Online]. Available: https://cloud.google.com/tpu/docs/

[60] P. Voigt and A. Von dem Bussche, *The EU General Data Protection Regulation (GDPR)*. Cham, Switzerland: Springer, 2017. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-3-319-57959-7.pdf

[61] L. Gil and A. Liska, *Security with AI and Machine Learning*, V. Wilson, Ed., 1st ed. Newton, MA, USA: O'Reilly Media, 2019. [Online]. Available: https://get.oreilly.com/ind_security-with-ai-and-machine-learning.html

[62] G. Nguyen, B. M. Nguyen, D. Tran, and L. Hluchy, "A heuristics approach to mine behavioural data logs in mobile malware detection system," *Data Knowl. Eng.*, vol. 115, pp. 129–151, May 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0169023X17303063

[63] DEEP-Hybrid-DataCloud. (2019). *Designing and Enabling e-Infrastructures for Intensive Processing in a Hybrid Datacloud*. Accessed: Sep. 9, 2019. [Online]. Available: https://deep-hybrid-datacloud.eu/

[64] R. Sun, S. Zhang, C. Yin, J. Wang, and S. Min, "Strategies for data stream mining method applied in anomaly detection," *Cluster Comput.*, vol. 22, no. 2, pp. 399–408, 2018. [Online]. Available: https://link.springer.com/article/10.1007/s10586-018-2835-2

[65] P. Burnap, R. French, F. Turner, and K. Jones, "Malware classification using self organising feature maps and machine activity data," *Comput. Secur.*, vol. 73, pp. 399–410, Mar. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167404817302535

[66] M. Monshizadeh, V. Khatri, B. G. Atli, R. Kantola, and Z. Yan, "Performance evaluation of a combined anomaly detection platform," *IEEE Access*, vol. 7, pp. 100964–100978, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8771247/

[67] (2017). *Bro Analysis Tools (BAT), GitHub*. Accessed: Sep. 9, 2019. [Online]. Available: https://github.com/SuperCowPowers/bat

[68] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*. Hoboken, NJ, USA: Wiley, 2015.

[69] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*. OTexts, 2018. [Online]. Available: https://otexts.com/

[70] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*. [Online]. Available: https://arxiv.org/abs/1412.3555

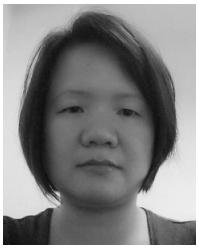
[71] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, Dec. 2010. [Online]. Available: http://www.jmlr.org/papers/v11/vincent10a.html

[72] C. Olah. (2015). *Understanding LSTM Networks*. Accessed: Sep. 9, 2019. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[73] N. Srivastava, E. Mansimov, and R. Salakhudinov, "Unsupervised learning of video representations using lstms," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 843–852. [Online]. Available: http://www.jmlr.org/proceedings/papers/v37/srivastava15.pdf

[74] CyberZHG. (2019). *Keras Self-Attention GitHub*. Accessed: Sep. 9, 2019. [Online]. Available: https://pypi.org/project/keras-self-attention/

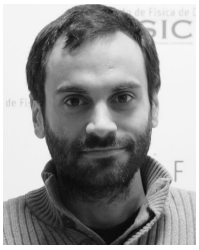
[75] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "WaveNet: A generative model for raw audio," 2016, *arXiv:1609.03499*. [Online]. Available: https://arxiv.org/abs/1609.03499
[76] P. Remy. (2019). *Keras TCN GitHub*. Accessed: Sep. 9, 2019. [Online]. Available: https://pypi.org/project/keras-tcn/
[77] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: https://arxiv.org/abs/1412.6980
[78] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," 2018, *arXiv:1811.03378*. [Online]. Available: https://arxiv.org/abs/1811.03378
[79] S. Kim and H. Kim, "A new metric of absolute percentage error for intermittent demand forecasts," *Int. J. Forecasting*, vol. 32, no. 3, pp. 669–679, Jul. 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0169207016000121
[80] H. Song, I. Triguero, and E. Özcan, "A review on the self and dual interactions between machine learning and optimisation," *Prog. Artif. Intell.*, vol. 8, no. 2, pp. 143–165, 2019. [Online]. Available: https://link.springer.com/article/10.1007/s13748-019-00185-z
[81] R. Astudillo and P. I. Frazier, "Bayesian optimization of composite functions," 2019, *arXiv:1906.01537*. [Online]. Available: https://arxiv.org/abs/1906.01537
[82] A. S. Foundation. (2019). *Apache Parquet—Columnar Storage Format*. Accessed: Sep. 9, 2019. [Online]. Available: https://parquet.apache.org/
[83] A. S. Foundation. (2019). *Apache Arrow: A Cross-Language Development Platform for in-Memory Data*. Accessed: Sep. 9, 2019. [Online]. Available: https://arrow.apache.org/
[84] A. B. Downey, *Think DSP: Digital Signal Processing in Python*. Newton, MA, USA: O'Reilly Media, 2016.
[85] A. S. Foundation. (2019). *Spark SQL: Apache Spark's Module for Working With Structured Data*. Accessed: Sep. 9, 2019. [Online]. Available: https://spark.apache.org/sql/
[86] B. Frénay, G. Doquire, and M. Verleysen, "Is mutual information adequate for feature selection in regression?" *Neural Netw.*, vol. 48, pp. 1–7, Dec. 2013. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0893608013001883
[87] G. Peyre. (2019). *Mathematical Foundations of Data Sciences. CNRS and DMA, Ecole Normale Superieure*. [Online]. Available: https://mathematical-tours.github.io
[88] J. González-López, S. Ventura, and A. Cano, "Distributed selection of continuous features in multilabel classification using mutual information," *IEEE Trans. Neural Netw. Learn. Syst.*, to be published. [Online]. Available: https://ieeexplore.ieee.org/document/8877992
[89] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 69, no. 6, Jun. 2004, Art. no. 066138, doi: 10.1103/PhysRevE.69.066138.
[90] B. C. Ross, "Mutual information between discrete and continuous data sets," *PLoS ONE*, vol. 9, no. 2, Feb. 2014, Art. no. e87357. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0087357
[91] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection," *Statist. Surv.*, vol. 4, pp. 40–79, 2010. [Online]. Available: http://projecteuclid.org/euclid.ssu/1268143839
[92] M. Sugiyama, T. Suzuki, S. Nakajima, H. Kashima, P. von Bünau, and M. Kawanabe, "Direct importance estimation for covariate shift adaptation," *Ann. Inst. Stat. Math.*, vol. 60, no. 4, pp. 699–746, 2008. [Online]. Available: https://link.springer.com/article/10.1007/s10463-008-0197-x
[93] (2019). *DEEP-Hybrid-DataCloud, Deep Open Catalog—Deep Marketplace*. Accessed: Oct. 9, 2019. [Online]. Available: https://marketplace.deep-hybrid-datacloud.eu/
[94] S. Dlugolinsky, V. Tran, and G. Nguyen. (2019). Massive Online Data Streams (MODS). CSIC-UC - Instituto de Fisica de Cantabria. Accessed: Oct. 9, 2019, doi: 10.20350/digitalCSIC/8975.
[95] S. Dlugolinsky and G. Nguyen. (2019). *Deep as a Service: Mods Container*. Accessed: Oct. 19, 2019. [Online]. Available: https://github.com/deephdc/DEEP-OC-mods
[96] Á. L. García, "DEEPaaS API: A REST API for machine learning and deep learning models," *J. Open Source Softw.*, vol. 4, no. 42, p. 1517, Oct. 2019, doi: 10.21105/joss.01517.
[97] S. Dlugolinsky, V. Tran, and G. Nguyen. (2019). Aggregated Network Monitoring Data. CSIC-UC-Instituto de Fisica de Cantabria. Accessed: Oct. 9, 2019, doi: 10.20350/digitalCSIC/8974.



GIANG NGUYEN received the M.Sc. degree in information technology and the Ph.D. degree in applied informatics from the Slovak University of Technology (STU), Bratislava, Slovakia. She is currently a Senior Researcher with the Institute of Informatics, Slovak Academy of Sciences (IISAS). She is a coauthor of scientific articles. She has been working in EU IST RTD, EU H2020 projects, such as DEEP-Hybrid-DataCloud, PROCESS, EGI-InSPIRE, PELLUCID, ANFAS, and national projects. Her research topics focused on soft computing, machine learning, deep learning, cyber security, and high-performance computing. She is a member of program committees. She is the Supervisor of Ph.D. degree students at IISAS, FIIT STU, Bratislava, and FEI TUKE, Košice, Slovakia, and a Reviewer of CC journals and national proposals.

STEFAN DLUGOLINSKY received the Ph.D. degree in applied informatics from the Slovak University of Technology (STU), Bratislava, Slovakia. He is currently a Scientific Researcher with the Institute of Informatics, Slovak Academy of Sciences (IISAS). He is a coauthor of scientific articles. He has been working on EU IST RTD, EU H2020 projects, such as Designing and Enabling E-infrastructures for intensive Processing in a Hybrid DataCloud (DEEP-HybridDataCloud), PROCESS, VENIS, COMMIUS, EUSAS, and national projects. His main research interests include information systems, information extraction, machine learning, and knowledge-oriented technologies.

VIET TRAN received the M.Sc. degree in informatics and information technology and the Ph.D. degree in applied informatics from the Slovak University of Technology (STU), Bratislava, Slovakia. He is currently a Senior Researcher with the Institute of Informatics, Slovak Academy of Sciences (IISAS). He is also actively participates on preparations and solving a number of EU IST RTD 4th, 5th, 6th, 7th FP, and EU H2020 projects. He is also working as the Work Package Leader of EU H2020 projects DEEP-HybridDataCloud, PROCESS and as a Key Person with EOSC-Hub and EOSC-Synergy. His primary research fields are complex distributed information processing, grid and cloud computing, system deployment, and security.



ÁLVARO LÓPEZ GARCÍA received the Ph.D. degree in science, technology, and computing from the University of Cantabria (UC). He was a Visiting Researcher with the IN2P3/CNRS Computing Center, Lyon, France, and a Research Associate with the Italian National Institute for Nuclear Physics (INFN). He is currently a Postdoctoral Researcher with CSIC. He is also an Assistant Professor with UC, teaching several computer's architecture subjects, and a Professor with the official master's degree in data science with the Universidad Internacional Mendendez Pelayo (UIMP). He has taken a part in several national and European projects, such as EGEE-II/III, Int.Eu.Grid, EUFORIA, EGI-InSPIRE, EGI-Engage (task leader of the Federated Cloud JRA), INDIGO-DataCloud (task leader of the Cloud Computing Virtualization JRA), and AARC-II. He is also coordinating the DEEP-Hybrid-DataCloud H2020 project and participating in the EOSC-Hub and EOSC-Synergy. In the last years, he has been working on the adoption of the cloud by scientific datacenters and users.