BACS: A comprehensive tool for deep learning-based anomaly detection in edge-fog-cloud systems

N. Milosevic, D. Jakovetic, S. Skrbic, M. Savic, D. Stamenkovic
Dept. of Mathematics and Informatics
Univ. of Novi Sad, Fac. of Sciences
Novi Sad, Serbia
{nmilosev, dusan.jakovetic, srdjan.skrbic,
svc, dusan.stamenkovic}@dmi.uns.ac.rs

J. Mascolo, D. Masera
Centro Ricerche Fiat
Torino, Italy
julien.mascolo@crf.it, davide.masera@fcagroup.com

***Abstract*—The Internet of Things (IoT) technology has shown to offer promising benefits in various domains, including smart cities, smart factories, and smart manufacturing. However, the large number of heterogeneous devices deployed in such IoT systems increases the attack surface and hence poses additional security challenges. In this context, many of the relevant threat detection tasks can be formulated as deep learning (DL)-based detection of anomalies in the data acquired by the IoT system, e.g., IoT sensory data, network traffic flow data, etc. There is hence a significant need for comprehensive anomaly detection (AD) tools in IoT applications. Indeed, as typical applications usually lack labeled data, multiple models need to be tried and compared one against another in order to get solid results. Moreover, the AD offerings should be adapted to the computational and storage capabilities of the underlying infrastructure for the given application. In this paper, we present BACS (Behavioral Analysis and Cognitive Security), a comprehensive DL-based tool for anomaly detection in IoT systems that has been developed in the context of the Horizon 2020 project C4IIoT. BACS is designed to work in edge-to-fog-to-cloud (E2F2C) environments, adapting its offering of anomaly detection methods to the computational, storage and timing requirements of the three layers. We describe the BACS internal architecture, explain how it is used by the developers, and illustrate its performance on real-world data provided by the Centro Ricerche Fiat (CRF).**

***Index Terms*—Deep learning-based anomaly detection, edge-to-fog-to-cloud system, performance evaluation.**

I. Introduction

Deep learning (DL) approaches are becoming increasingly popular for improving device monitoring and security features of various Internet of Things (IoT) systems [2], [17], [18], [20]. Large scale IoT data is collected and analyzed by DL methods that produce insights about the health status of the IoT infrastructure, emerging threats, etc. Among different DL methods, outlier (anomaly) detection is of significant interest [13]: finding anomalies in the IoT-generated data may correspond to, e.g., detecting faults in device operation, errors in communication, but also to detecting different types of security threats (e.g., device tampering, IoT pivot, botnet, distributed denial of service, etc.) [7]. The DL-based models utilized include, e.g., recurrent neural networks [16], autoencoders [9], [8], [13]; one-class classification models [3], etc.

In IoT applications, many systems follow an edge-to-fog-to-cloud architecture (E2F2C) design, e.g., [11]. Therein, the data generated by IoT devices gets transferred to fog servers (e.g., mobile operator gateways) and is subsequently further communicated to the cloud. In order to shorten response times, data analytics (e.g., anomaly detection) in E2F2C systems takes place not only at the cloud, but also at the edge or fog. As edge devices usually have limited computational and storage capabilities, only low-to-moderate complexity models (with a potentially limited accuracy performance) can be deployed therein.

There is a significant need for comprehensive anomaly detection (AD) tools targeted for IoT applications. Indeed, as usual applications exhibit lack of labels in the data, typically many different AD methods should be tried and compared one against another in order to obtain results with confidence. An additional reason for trying multiple AD models is that different models can perform differently relative to each other for different applications or data sets, i.e., a single method is usually not universally better than others. Comprehensive AD tools with built-in methods hence facilitate and shorten experimentation times for developers. There exist several tools and libraries for anomaly detection either as stand-alone or sub-modules of widely used tools, such as [19], scikit-learn [12], Tensorflow [1], etc. However, the PyOD tool focused on anomaly detection does not incorporate relevant advanced deep learning-based AD methods such as those based on autoencoders. In addition, more work is needed on devising a unifying tool for AD adapted to E2F2C systems.

Contributions. In this paper, we present the BACS (Behavioral Analysis and Cognitive Security) tool for anomaly detection in E2F2C environments that has been developed in the context of the Horizon 2020 project C4IIoT [4]. BACS is designed to adapt to E2F2C environments, i.e., it offers different

The paper is supported by the European Union's Horizon 2020 Research and Innovation program under Grant Agreement No 833828. The paper reflects only the view of the authors and the Commission is not responsible for any use that may be made of the information it contains.

packages that are optimized for edge, for, and cloud operation. BACS offers a comprehensive pool of both supervised and unsupervised anomaly detection methods (See Section III). We present the BACS' internal architecture, describe the available anomaly detection methods, and demonstrate how BACS is used by developers. Finally, we present comprehensive results on the BACS anomaly detection performance on several real-world IoT applications, namely the smart logistics and smart manufacturing applications through the real data provided by the Centro Ricerche Fiat (CRF). While there currently exist several widely used software tools that offer anomaly detection capabilities, such as PyOD, Tensorflow, Scikit-learn, etc., BACS builds upon their implementations to offer a more comprehensive pool than each of these individual frameworks provide, that is furthermore adapted to E2F2C systems. Unlike the works [3], [8], [9], [13], [16] that consider a specific class of DL models and a targeted IoT application, we offer a comprehensive tool that allows to implement a more general class of DL models and target a broader class of IoT anomaly detection applications.

Paper organization. Section II provides preliminaries on a prototypical E2F2C architecture on top of which BACS can be deployed. Section III describes the BACS architecture and the AD models that it offers and explains how BACS is used to implement various AD models. Section IV presents comprehensive results on the performance of BACS on real industrial IoT applications. Finally, Section V concludes the paper. Some additional numerical results and considerations can be found in the Appendix.

II. Background and Preliminaries

We describe a prototypical E2F2C architecture to which the BACS system can be deployed (Figure 1). The E2F2C architecture is a widely adopted architectural pattern for IoT, e.g., [11]. The architecture consists of three layers. The first (bottom) layer consists of various edge devices that acquire measurements about the environment (e.g., various sensors and smart meters in a smart home application). The type of IoT devices can vary (e.g., microcontollers, single board computers, cameras, etc.), but typical applications assume presence of devices with low-to-moderate computational and storage capabilities. The data acquired by the IoT devices is sent via appropriate communication channels to the fog layer. The type of communication may vary depending on the application. For example, in smart home applications, the devices can communicate via narrow-band IoT (NB-IoT) protocol, [10]. The fog layer contains various servers (e.g., mobile operator gateways) that posses intermediate storage and processing capabilities. Finally, the data at the fog layer can be further transferred (e.g., through the core network) to the cloud. The cloud layer assumes highest computational and storage capabilities relative to the edge and fog layers. However, processing at the cloud incurs an increased delay that corresponds to the time required for the data acquired by an IoT edge device to first reach fog and subsequently to arrive to the cloud. The corresponding edge-to-fog and fog-to-cloud delay times are application specific. For illustration purposes, if platform has encryption capabilities which require message encryption and decryption at certain points in the system architecture the delay time can increase by couple of seconds at the minimum, even with high network throughput.

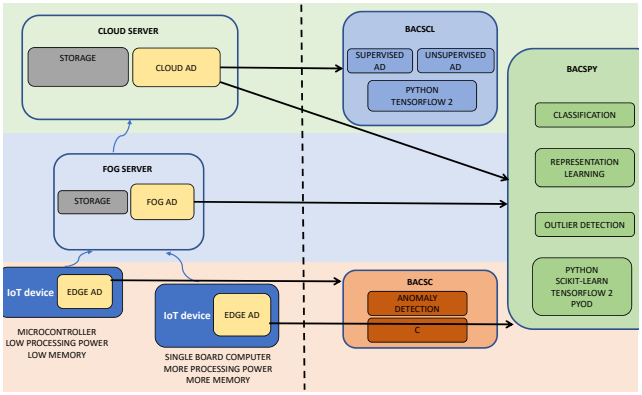

Fig. 1. A prototypical E2F2C system; BACS' mapping to E2F2C.

As described ahead in Section III, BACS allows that each of the three layers to perform anomaly detection. While BACS is equipped with a number of different DL-based anomaly detection models, for a simplified presentation we focus here on autoencoders-based anomaly detection, to set up for future reference relevant metrics and quantities for system operation. Autoencoder is a feed-forward neural network that learns latent low-dimensional data representations, e.g., [13]. An autoencoders operates in two phases: training and inference. Anomaly detection based on autoencoders works as follows, e.g., [13]. First, in the training phase, the autoencoder takes as inputs normal (non-anomalous) data points and learns a latent low-dimensional representation based on the training data. Denote by $\mathcal{A}$ the trained autoencoder model that takes as input a data point X and produces as its output its autoencoder reconstruction $Y = \mathcal{A}(X)$. In the inference phase, a new data point X' is declared as anomaly if the reconstruction error $\mathrm{Err}(X')$ (for example, one can take $\mathrm{Err}(X') = \|X' - \mathcal{A}(X')\|_2^2$) is above a threshold $\tau > 0$. One can take τ as the maximal or high-percentage quantile (e.g., 95% quantile) over the reconstruction errors across the training data points.

We assume that training for anomaly detection in the E2F2C system is done offline, e.g., at the cloud. However, inference can be done either at the edge, fog, or cloud. This means that different instances of trained autoencoders are deployed at the edge, fog, and cloud. We refer to those as $\mathcal{A}_{\mathrm{edge}}$, $\mathcal{A}_{\mathrm{fog}}$, and $\mathcal{A}_{\mathrm{cloud}}$. Each of the three autoencoders takes as inputs either a single data point generated at the edge (e.g., a single sensor reading by an IoT device), or multiple data points across multiple devices and/or multiple time instances (timeseries data).

Due to limited computational and storage resources at the edge (or fog), the autoencoders $\mathcal{A}_{\text{edge}}$ and $\mathcal{A}_{\text{fog}}$ are of a lower "complexity" when compared to $\mathcal{A}_{\text{cloud}}$. The complexity of a model here refers to the size of input the model can accommodate, and the size of storage needed to deploy the trained model (that corresponds, e.g., to the number of layers in the autoencoders neural network). For example, as edge devices have a limited storage, they cannot accommodate autoencoders with large inputs sizes or large number of layers. Clearly, the "size" of models that edge, fog, or cloud layers can host increases as we traverse from edge to cloud. For example, edge devices can usually only accomodate single hidden layer models while fog and cloud models often go much deeper (and wider). Some Cloud BACS neural network models exceed one million parameters, spanning over dozen of large (wide) layers.

With $\mathcal{A}_{\text{edge}}$, $\mathcal{A}_{\text{fog}}$ and $\mathcal{A}_{\text{cloud}}$, we are interested in their anomaly detection performance. While various metrics are possible, we will be mainly interested in precision P, recall R, and F_1 score, defined respectively by: $P = \frac{TP}{TP+FP}$; $R = \frac{TP}{TP+FN}$; $F_1 = \frac{2 \cdot P \cdot R}{P+R}$. Here, TP, FP, and FN are the number of true positive, false positive, and false negative outcomes, respectively. That is, TP is the total number of testing data points for which the autoencoder declared the data point as an anomaly, while the data point is indeed an anomaly. (FP and FN are defined analogously.) In order to evaluate the metrics P, R, and F_1, we need to have the "ground truth" available, i.e., we need to know for a given data point whether it is an anomaly or not. This scenario corresponds to the experimental setting we report on in this paper (Section IV). When ground truth is not available, several methods to evaluate anomaly detection are possible, e.g., [6].

Another metric of interest for us is the *response time*, that we define here as the time required by a trained autoencoder to receive the data input and to process it and produce the decision (anomaly or not). With $\mathcal{A}_{\text{edge}}$, the response time is just the time of inference of the model; with $\mathcal{A}_{\text{fog}}$, the response time is the inference time of the autoencoder plus the time it takes the input to arrive from edge to fog; finally, with $\mathcal{A}_{\text{cloud}}$, the response time is the inference time of the autoencoder plus the time it takes the data input to arrive from edge to cloud.

III. BACS TOOL DESCRIPTION

A. BACS internal architecture and AD models available

BACS internal structure contains the following modules adapted to the E2F2C prototypical architecture (See Figure 1 how BACS modules are mapped to E2F2C).

BACSCL (BACS Cloud Layer) performs anomaly detection based on deep autoencoder forests (unsupervised AD) and deep neural network forests (supervised AD) implemented in Python using the Tensorflow 2 library. The Cloud module also offers support for recurrent neural network models (e.g. LSTM's) and regression threshold-based fully-connected neural networks. Diferentially private methods (modifications of K-Means and PCA) are also available. BACSCL modules can run in a distributed cloud environment and support both data partitioning (training datasets are stored within a distributed file system) and model partitioning (deep autoencoders/neural networks within a forest run on different computational nodes within the cloud).

BACSPY provides anomaly detection based on outlier detection, classification and representation learning algorithms implemented in Python using Tensorflow 2, scikit-learn and PyOD libraries. BACSPY modules also realize AD functionalities running at the cloud, field gateway and edge devices with higher computational power (e.g., Raspberry PI). BACSPY supports both standalone and federated AD models.

BACSC contains lightweight anomaly detection routines implemented in C for constrained microcontroller edge devices.

We outline here a collection of anomaly detection models available in BACSPY. We first introduce naming conventions which allow us to simplify the presentation. Namely, model names starting with "TF" denote TensorFlow2 Deep-Learning models. Model names ending with SKLAD denote scikit-learn models, those ending with PYODAD denote PyOD models, while those ending with DPLAD denote the diffprivlib library-based implementations of differentialy private models.

Edge and Fog (FG) models have designation in their names (Edge and FG respectfully), while for the cloud models the layer designation is omitted. The available (cloud) models are the following:

1. TensorFlow Autoencoder models: TFAutoAD – TensorFlow Autoencoder; TFAutoDeepAD – TensorFlow Autoencoder variation with more layers (deeper); TFAutoDeepVAEAD – TensorFlow Autoencoder variation with more units in layers (wider); TFAutoVAEAD – TensorFlow Variational Autoencoder; TFAutoWideVAEAD – TensorFlow Variational Autoencoder with more units in layers;

2. Scikit-learn models: EE_SKLAD – Elliptic Envelope Model; LOF_SKLAD – Local Outlier Factor Model; IF_SKLAD – Isolation Forest Model; SVM_SKLAD – Support Vector Machine Model (One Class).

3. PyOD models: ABOD_PyODAD – Angle-based Outlier Detector; KNN_PyODAD – K-nearest Neighbours; PCA_PyODAD – Principal Component Analysis; HBO_PyODAD – Histogram-based Outlier Detection; AE_PyODAD – Autoencoder Model (PyOD specific).

4. TensorFlow Fully Connected models: TFAutoFCNAD – Fully connected (Dense) Neural Network; TFAutoDeepFCNAD – Fully connected (Dense) Neural Network Model with more hidden layers (deeper).

5. TensorFlow Recurrent models: TFAutoLSTMAD – Long-Short Term Memory Recurrent Neural Network; TFAutoGRUAD – Gated Recurrent Unit Recurrent Neural Network; TFAutoRNNAD – Recurrent Neural Network.

6. Facebook Prophet Model (TFAutoProphetAD).

7. Diferentially private models (diffprivlib-based): Kmeans_DPLAD – Differentially private K-Means; PCA_DPLAD – Differetially private Principal Component Analysis.

8. Ensemble Models: AutoEnsembleAD – Combination or a subset of all the other models with weighted (model size and

complexity based) voting system.

B. BACS tool usage considerations

We next show the ease-of-use aspect of the BACS tool by explaining how one can use it to obtain trained anomaly detection models for a custom dataset.

When using BACS, the first step is to obtain and ingest data. BACS works with tabular data which needs to follow certain rules in order for BACS to automatically ingest it. The data is provided to BACS in the CSV format, where the first column is expected to be the timestamp as BACS is used for time-series analysis. Other columns will be used as feature columns.

The next step is choosing which models are to be trained by using the included BACS model training script. By default all available models are trained for the entire infrastructure (edge, fog, cloud) but a subset can be selected.

The differences in model types for the different layers (edge, fog, cloud) are in model size and more importantly window lengths on which they operate (i.e., sizes of input at inference–see Section II). Cloud models generally operate on longer window lengths while fog and edge models operate on shorted window lengths because of memory and processing constraints. Data is automatically processed (with a sliding window approach) by BACS data loading modules.

Depending on the model type, the data is adapted to the model automatically. For example, some models (e.g., some neural network models) require data normalization which is performed automatically within BACS. Also, some models expect different formats at the input and at the output, while autoencoders use the same inputs and output vectors, and regression networks split the selected data point windows. BACS handles this model heterogeneity automatically.

After models are trained, training metrics can be obtained. Final loss values and average inference times are provided for all models. If labels (ground truth) are present, BACS can also provide validation metrics such as accuracy, precision, recall and F1-scores.

BACS also offers the ability to synthetically generate anomalous data by in-place modification of normal training data (which usually does not contain anomalies). There are several strategies for generating anomalous data such as randomization of feature values, random increases and decreases, zeroing out of the data, etc. This feature is especially useful if only normal non-anomalous data is available and an anomaly detection model is still necessary in early stages of system implementation. The synthetic anomaly generator can also be used in conjunction with supervised models to perform self-supervised learning without anomalous data.

Regarding model tuning, all BACS models offer a threshold quantile hyperparameter which defines how anomalies are detected. This parameter defines which quantile of the observed training loss values (errors) is to be used for the threshold above which anomalies are detected. By default this value is set to 1.0; this means that the observed error during real-world usage has to be greater than the highest observed error during training. BACS provides sensible default values for all the hyperparameters which can be used as a baseline for tuning more performant models. Further usage considerations, e.g., on connecting with message queue systems, can be found in the Appendix.

IV. Numerical results

We present here numerical results on a comprehensive set of experiments for a real data set provided by CRF in the context of a smart factory application. See also the Appendix for additional numerical results, including those on a logistics application scenario.

The data comes from autonomous ground vehicles on the factory floor. The vehicles are equipped with IoT devices which log various sensory data used for anomaly detection. The dataset contains the following features: 1) timestamp – exact moment of the data log; 2) motor id – source of the data point (vehicle id); 3) acceleration – value from the acceleration sensor (absolute); and 4) velocity – values from the speed sensor (absolute). In this context, anomalous sensory readings (e.g., acceleration) may correspond to errors in the robot trajectory and are thus highly relevant to detect.

We apply various unsupervized BACS models on the data. Table I presents the number of detected anomalies, inference times, and average inference times for various methods. The results also include those for the experimental Facebook Prophet time-series forecasting model adapted for anomaly detection usage.

We performed additional analysis with synthetic anomaly generation. For training, only normal (non-anomalous) real data is used. On the other hand, for testing, we generated synthetic anomalies that accurately emulate realistic anomalies for the considered scenarios. This also allows us to evaluate accuracy of the methods, e.g., to measure the F1 score.

We generate anomalies in two scenarios, i.e., two types of anomalies are created, referred to here as singular and continuous anomalies. Singular anomalies are those where sensor data becomes corrupted at any single point in time. Continuous anomalies are the anomalies that span multiple consecutive data points in time; such anomalies are more likely to occur in streaming, real-time usage. In Table II we present the resulting metrics for the singular anomalies all the BACS models compatible with the considered application scenario. See the Appendix for the results on continuous anomalies. We consider F1-scores as the most important metric, as accuracy can be misleading due to a natural class imbalance of the data (smaller number of anomalies compared to normal data). We can see that almost all BACS models achieve good F1-scores and accuracies.

V. Conclusion

In this paper, we presented BACS, a comprehensive tool for anomaly detection in E2F2C IoT systems. BACS offers to data analysts and developers implementations of a number of different unsupervized and supervized deep learning methods. The different offerings (modules) of BACS are adapted to the deployment requirements and constraints of the underlying

Model	Anomalies	Inf. Time	Avg. Inf. Time
TFAutoAD (Edge)	323	50.57	0.0156
SVM_SKLAD (Edge)	324	0.44	0.0001
TFAutoAD (FG)	323	50.48	0.0156
SVM_SKLAD (FG)	321	0.49	0.0002
TFAutoAD	323	50.46	0.0156
TFAutoDeepAD	323	50.53	0.0157
TFAutoDeepVAEAD	323	49.46	0.0153
TFAutoVAEAD	322	49.26	0.0153
TFAutoWideVAEAD	327	49.43	0.0153
EE_SKLAD	323	0.89	0.0003
LOF_SKLAD	411	2.42	0.0008
IF_SKLAD	484	150.24	0.0466
SVM_SKLAD	321	0.51	0.0002
ABOD_PyODAD	470	2.24	0.0007
KNN_PyODAD	281	0.73	0.0002
PCA_PyODAD	323	0.67	0.0002
HBO_PyODAD	466	0.38	0.0001
AE_PyODAD	323	84.56	0.0262
TFAutoFCNAD	291	49.37	0.0153
TFAutoDeepFCNAD	355	50.01	0.0155
TFAutoLSTMAD	484	50.54	0.0157
TFAutoGRUAD	484	50.56	0.0157
TFAutoRNNAD	645	50.43	0.0156
TFAutoProphetAD	3040	8578.22	2.6525
Kmeans_DPLAD	1991	1.04	0.0003
PCA_DPLAD	3173	0.70	0.0002

TABLE I

THE COLUMN ANOMALIES REPRESENTS THE NUMBER OF DETECTED ANOMALIES ON THE TRAINING DATASET (MODEL SENSITIVITY). INF. TIME SHOWS THE TOTAL MODEL INFERENCE TIME FOR ALL DATASET DERIVED TIME SERIES WINDOWS. AVG. INF. TIME REPRESENTS THE AVERAGE MODEL INFERENCE TIME WHICH IS THE EXPECTED MODEL RESPONSE TIME IN REAL-WORLD USAGE. TIMES ARE IN SECONDS. MODELS OPERATE IN THE CLOUD LAYER (SEE FIGURE 1), UNLESS OTHERWISE SPECIFIED IN THE MODEL NAME.

Model	Accuracy	Precision	Recall	F1
TFAutoAD (Edge)	0.90	0.50	0.94	0.65
SVM_SKLAD (Edge)	0.89	0.45	0.78	0.57
TFAutoAD (FG)	0.93	0.86	0.99	0.92
SVM_SKLAD (FG)	0.89	0.84	0.89	0.86
TFAutoAD	0.92	0.94	0.93	0.94
TFAutoDeepAD	0.49	0.79	0.28	0.42
TFAutoDeepVAEAD	0.52	0.84	0.31	0.45
TFAutoVAEAD	0.51	0.84	0.31	0.45
TFAutoWideVAEAD	0.52	0.83	0.31	0.45
EE_SKLAD	0.65	0.89	0.51	0.65
LOF_SKLAD	0.91	0.93	0.94	0.93
IF_SKLAD	0.52	0.80	0.35	0.49
SVM_SKLAD	0.74	0.91	0.66	0.76
ABOD_PyODAD	0.91	0.92	0.94	0.93
KNN_PyODAD	0.85	0.93	0.82	0.87
PCA_PyODAD	0.48	0.78	0.28	0.41
HBO_PyODAD	0.63	0.85	0.52	0.65
AE_PyODAD	0.48	0.78	0.26	0.39
TFAutoFCNAD	0.87	0.94	0.85	0.90
TFAutoDeepFCNAD	0.87	0.93	0.86	0.89
TFAutoLSTMAD	0.72	0.89	0.64	0.74
TFAutoGRUAD	0.62	0.85	0.50	0.63
TFAutoRNNAD	0.86	0.89	0.90	0.89
TFAutoProphetAD	0.15	0.10	0.95	0.18
Kmeans_DPLAD	0.50	0.63	0.56	0.59
PCA_DPLAD	0.62	0.64	0.94	0.76

TABLE II

RESULTS WHEN TESTING WITH SYNTHETIC SINGULAR ANOMALIES (SELF-SUPERVISED RESULTS). MODELS OPERATE IN THE CLOUD LAYER, UNLESS OTHERWISE SPECIFIED IN THE MODEL NAME.

E2F2C system, offering for example to the edge layer tools that produce lighter trained models. We describe the BACS internal architecture, demonstrate its usage, and present comprehensive experimental results that report BACS performance of real data provided by Centro Ricerche Fiat (CRF).

REFERENCES

[1] M. Abadi et al., "TensorFlow: A system for large-scale machine learning." 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), USENIX Association, pp. 265–283, 2016.

[2] J. Bas, A. Perez-Neira, "On the physical layer security of IoT devices over satellite," in Proc. Europ. Sig. Proces. Conf. (EUSIPCO), A Coruna, Spain, pp. 1–5, Sept. 2019.

[3] V. H. Bezerra, V. da Costa, S.B. Junior, R.S. Miani, B.B. Zarpelão, "IoTDS: A One-Class Classification Approach to Detect Botnets in Internet of Things Devices," Sensors, Vol. 19, No. 14, 2019.

[4] C4IIoT: Cyber security 4.0: protecting the Industrial Internet Of Things, https://www.c4iiot.eu/

[5] L. Da Xu, W. He, S. Li, "Internet of things in industries: A survey," IEEE Trans. Ind. Inform., Vol. 10, No. 4, pp. 2233–2243, 2014.

[6] N. Goix, "How to Evaluate the Quality of Unsupervised Anomaly Detection Algorithms?," 2016, available at: https://arxiv.org/abs/1607.01152

[7] F. Hussain, R. Hussain, S. A. Hassan and E. Hossain, "Machine Learning in IoT Security: Current Solutions and Future Challenges," IEEE Comm. Surveys & Tutorials, Vol. 22, No. 3, pp. 1686–1721, 2020.

[8] Y. Kawachi, Y. Koizumi, S. Murata, and N. Harada, "A twoclass hyperspherical autoencoder for supervised anomaly detection," in Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, pp. 3047–3051, 2019.

[9] Y. Koizumi, S. Murata, N. Harada, S. Saito, and H. Uematsu, "SNIPER: Few-shot learning for anomaly detection to minimize false-negative rate with ensured true-positive rate," in Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, pp. 915–919, 2019.

[10] B. Martinez, F. Adelantado, A. Bartoli and X. Vilajosana, "Exploring the Performance Boundaries of NB-IoT," in IEEE Internet of Things Journal, vol. 6, No. 3, pp. 5702–5712, 2019.

[11] N. Mohan and J. Kangasharju, "Edge-Fog cloud: A distributed cloud for Internet of Things computations," Cloudification of the Internet of Things (CIoT), Paris, France, Nov. 2016., DOI: 10.1109/CIOT.2016.7872914

[12] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 122, no. 1, pp. 2825—2830, 2011.

[13] M. Savic et al., "Deep Learning Anomaly Detection for Cellular IoT with Applications in Smart Logistics," IEEE Access, vol. 9, pp. 59406–59419, Apr. 2021.

[14] I. Stellios, P. Kotzanikolaou, M. Psarakis, C. Alcaraz and J. Lopez, "A Survey of IoT-Enabled Cyberattacks: Assessing Attack Paths to Critical Infrastructures and Services," IEEE Communications Surveys & Tutorials, Vol. 20, No. 4, pp. 3453–3495, 2018.

[15] S. J. Taylor, B. Letham. "Forecasting at scale," The American Statistician 72.1, pp. 37-45, 2018.

[16] Z. Tian, C. Luo, J. Qiu, X. Du, and M. Guizani, "A distributed deep learning system for web attack detection on edge devices," IEEE Trans. Ind. Inform., Vol. 16, No. 3, pp. 1963–1971, 2019.

[17] A. Zanella, N. Bui, A. Castellani, L. Vangelista, M. Zorzi, "Internet of Things for Smart Cities," Internet of Things Journal, Vol. 1, no. 1, pp. 22–32, 2014.

[18] J. Zhang, R. S. Blum, and H. V. Poor, "Approaches to secure inference in the Internet of Things: Performance bounds, algorithms, and effective attacks on IoT sensor networks," IEEE Signal Process. Mag., vol. 35, no. 5, pp. 50–63, Sep. 2018.

[19] Y. Zhao, Z. Nasrullah, and Z. Li, "PyOD: A Python Toolbox for Scalable Outlier Detection." Journal of Machine Learning Research, vol. 20, no. 96, pp. 1–7, 2019.

[20] M. Zolanvari, M. A. Teixeira, L. Gupta, K. M. Khan and R. Jain, "Machine Learning-Based Network Vulnerability Analysis of Industrial Internet of Things," IEEE Internet of Things Journal, Vol. 6, No. 4, pp. 6822–6834, 2019.

APPENDIX

A. Further BACS usage considerations

We describe here further usage considerations for the BACS tool.

The trained models can be exposed in large systems by using BACS components for communication (inference API). The preferred way to do so is to use a message queue system for all components and BACS components are implemented to use the open-source Apache Kafka message queue. All components use the same message queue to communicate which allows for other various components to be used such as anomaly mitigation systems, data storage and encryption services and so on. Usage of message queue also allows BACS components to keep certain parts of the global system state available for usage. For example, some components require historical data (buffers) of events in the system which can be obtained through the message queue.

It is important to mention that BACS models can be also used without a message queue in place. All models offer data storing and loading methods (file-based) so they can be serialized and reused.

An example how a trained BACS model (variational auto-encoder can be used) is shown below:

```
model = BACSModel(
    # next line specifies which model
    # to use and where it is stored
    TFAutoVAEAD(model="BACS_VAEAD_SFUC"),
    # this parameter provides path
    # for model saving
    # (e.g. if tuning is performed)
    "models/BACS_VAEAD_SFUC_tuned",
    # window length parameter,
    # depends on the "position"
    # (edge, fog, cloud) of the model
    10,
)

# is_anomaly returns whether an anomaly
# is detected and a normalized confidence
# (certainty) score
detection, conf = model.is_anomaly(t)
```

B. Further numerical results for the smart factory scenario and continuous synthetic anomalies

Table III presents results for the various BACS models for the continuous synthetic anomalies and the dataset described in Section IV.

Model	Accuracy	Precision	Recall	F1
TFAutoAD (Edge)	0.47	0.14	0.88	0.24
SVM_SKLAD (Edge)	0.53	0.13	0.67	0.21
TFAutoAD (FG)	0.61	0.50	0.99	0.67
SVM_SKLAD (FG)	0.61	0.51	0.86	0.64
TFAutoAD	0.70	0.72	0.87	0.79
TFAutoDeepAD	0.64	0.76	0.64	0.70
TFAutoDeepVAEAD	0.72	0.78	0.80	0.79
TFAutoVAEAD	0.72	0.78	0.79	0.79
TFAutoWideVAEAD	0.72	0.78	0.80	0.79
EE_SKLAD	0.53	0.72	0.44	0.55
LOF_SKLAD	0.77	0.75	0.95	0.84
IF_SKLAD	0.74	0.74	0.93	0.82
SVM_SKLAD	0.72	0.73	0.90	0.81
ABOD_PyODAD	0.74	0.73	0.96	0.83
KNN_PyODAD	0.63	0.70	0.73	0.71
PCA_PyODAD	0.64	0.77	0.64	0.70
HBO_PyODAD	0.55	0.72	0.49	0.58
AE_PyODAD	0.65	0.78	0.63	0.70
TFAutoFCNAD	0.65	0.72	0.73	0.73
TFAutoDeepFCNAD	0.64	0.73	0.70	0.71
TFAutoLSTMAD	0.60	0.74	0.60	0.66
TFAutoGRUAD	0.53	0.68	0.51	0.59
TFAutoRNNAD	0.69	0.72	0.85	0.78
TFAutoProphetAD	0.12	0.10	0.98	0.18
Kmeans_DPLAD	0.54	0.66	0.60	0.63
PCA_DPLAD	0.51	0.60	0.72	0.65

TABLE III
SMART FACTORY SCENARIO: BACS RESULTS WHEN TESTING WITH SYNTHETIC CONTINUOUS ANOMALIES (SELF-SUPERVISED RESULTS). MODELS OPERATE IN THE CLOUD LAYER, UNLESS OTHERWISE SPECIFIED IN THE MODEL NAME.

C. Numerical results for a smart logistics application

We describe here results for the experiment where we have analyzed the CRF data in a logistics scenario. In more detail, we analyze the data logged by IoT devices embedded onto a logistics vehicle; the IoT devices collect information about the vehicle location and various other sensor parameters. The features of the dataset include:

1) timestamp – exact moment of the data log;
2) GPS data (latitude, longitude, altitude, speed) – used to track the vehicle trajectory;
3) Acceleration and magnetic acceleration sensors (x, y and z axis) – used to track fine movements of the vehicle.

The data was organized by a vehicle itinerary, where the average time series length was around 200 data points. No data was labeled, and unsupervised methods were used. The device also has a BACSC autoencoder implementation for edge anomaly detection. Table IV presents inference times, average inference times, and the number of detected anomalies for different BACS models.

Model	Anomalies	Inf. Time	Avg. Inf. Time
TFAutoAD (Edge)	60	3.10	0.0158
TFAutoAD (FG)	6	3.07	0.0159
TFAutoAD	56	3.02	0.0160
TFAutoDeepAD	50	3.02	0.0160
TFAutoDeepVAEAD	175	3.10	0.0165
TFAutoVAEAD	174	2.94	0.0156
TFAutoWideVAEAD	175	2.96	0.0157
EE_SKLAD	127	0.06	0.0003
SVM_SKLAD	180	0.03	0.0002
LOF_SKLAD	154	0.12	0.0006
IF_SKLAD	176	8.63	0.0459
ABOD_PyODAD	156	0.14	0.0008
KNN_PyODAD	150	0.04	0.0002
PCA_PyODAD	175	0.04	0.0002
HBO_PyODAD	164	0.04	0.0002
AE_PyODAD	175	5.02	0.0267
TFAutoFCNAD	0	2.93	0.0156
TFAutoDeepFCNAD	0	2.97	0.0158
TFAutoLSTMAD	8	3.14	0.0167
TFAutoGRUAD	9	2.93	0.0156
TFAutoRNNAD	86	2.96	0.0158
Kmeans_DPLAD	2	0.07	0.0004
AutoEnsembleAD	36	31.17	0.1658

TABLE IV

BACS RESULTS ON THE LOGISTICS SCENARIO: THE COLUMN ANOMALIES SHOWS THE NUMBER OF DETECTED ANOMALIES ON THE TRAINING DATASET (MODEL SENSITIVITY); INF. TIME REPRESENTS THE TOTAL MODEL INFERENCE TIME FOR ALL DATASET DERIVED TIME SERIES WINDOWS. AVG. INF. TIME REPRESENTS THE AVERAGE MODEL INFERENCE TIME WHICH IS THE EXPECTED MODEL RESPONSE TIME IN REAL-WORLD USAGE. TIMES ARE IN SECONDS. MODELS OPERATE IN THE CLOUD LAYER, UNLESS OTHERWISE SPECIFIED IN THE MODEL NAME.