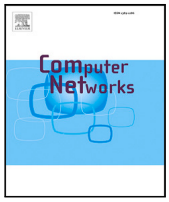
Computer Networks 254 (2024) 110828

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



ASAP: Automatic Synthesis of Attack Prototypes, an online-learning, end-to-end approach

Jesús F. Cevallos M., Alessandra Rizzardi[1], Sabrina Sicari [*,2], Alberto Coen-Porisini

Dipartimento di Scienze Teoriche e Applicate, Università degli Studi dell'Insubria, via O. Rossi 9, Varese (VA), 21100, Italy

ARTICLE INFO

Keywords:
Zero-day attack detection
Out-of-distribution generalization
Collective anomaly detection
Internet of Things

ABSTRACT

Zero-day attack detection and categorization is an open-research field where four main context factors need to be taken into account: novel or zero-day attacks (i) are unlabeled by definition, (ii) may correspond to out-of-distribution data, (iii) can arise concurrently, and (iv) distribution shifts in the feature space need online-learning. Given such constraints, the online detection and categorization of new cyber threats can be modeled as a heterogeneous collective anomaly detection problem, for which no online-learning solutions exist purely based on back-propagation. To this respect, this paper presents an online-learning, end-to-end back-propagation strategy for Automatically Synthesizing the potential signatures or Attack Prototypes of novel cyber threats (ASAP). The presented framework incorporates automatic feature engineering, operating over raw data from the OpenFlow monitoring API and raw bytes of traffic captures. In ASAP, specialized inductive biases enhance the training data efficiency and accommodate the inference machinery to resource-constrained scenarios such as the Internet of Things. Finally, the validity of this framework is demonstrated in a live training experiment comprising IoT traffic emulation [3].

1. Introduction

Anomaly detection has been applied to infer the presence of cyber-attacks among network-related data for decades [2]. Moreover, the Network Intrusion Detection (NID) research community has recently sophisticated vanilla offline statistical analyses to face the challenges presented by distribution shifts among observed data. Approaches based on *online-learning* [3] and *continuous-learning* [4] have been implemented to dynamically adapt the definition of normal and abnormal patterns in network observations. Online-learning models produce inferences each time they process a single observation, while offline-learning models collect observations and perform bulk processing over them to produce a result in a second moment.

Another challenging objective accomplished by state-of-the-art NID approaches is training-data efficiency: recent approaches based on *Few-shot Learning* (FSL) [5] have increased the learning efficiency of neural networks, enabling them to learn from low training samples. In this context, a recent synergic advance to FSL-based NID is *Multi-Modal learning* (MML) [6,7], where cyber-attacks are analyzed through multiple *domains* or feature-sets. If at least one of these feature sets is labeled, on-the-fly categorization of zero-day attacks is possible even on a *Zero-Shot Learning* (ZSL) basis [8].

In general, learning from multiple domains has proven useful to enhance the context-awareness of NIDs based on Machine Learning (ML), giving the inference machinery a 360° visibility of the network state and thus augmenting the quality of predictions [9]. In this context, *Software Defined Networking* (SDN) [10] is a game-changing technology that enables the fine-grain monitoring of the data plane and the efficient management of the control plane [11].

This paper investigates the best combination of the above-mentioned techniques (i.e., online-learning, few-shot learning, multi-modal learning, and SDN-based monitoring) to face a still-to-resolve challenge in the current research landscape of NID: the online categorization of collective anomalies from raw network observations. Moreover, this categorization is made without manual feature selection/pre-processing, but within an end-to-end automatic feature engineering pipeline. To this respect, it is noted that offline analyses of traffic traces can use unsupervised learning techniques to cluster anomalies [12]. However,

* Corresponding author.
E-mail addresses: jf.cevallosmoreno@uninsubria.it (J.F. Cevallos M.), alessandra.rizzardi@uninsubria.it (A. Rizzardi), sabrina.sicari@uninsubria.it (S. Sicari), alberto.coenporisini@uninsubria.it (A. Coen-Porisini).
[1] Member, IEEE.
[2] Senior Member, IEEE.
[3] To foster research in the field, the source-code of ASAP alongside instructions to reproduce the experiments are made publicly available among the set of SmartVille NID models in [1].

https://doi.org/10.1016/j.comnet.2024.110828
Received 4 April 2024; Received in revised form 23 August 2024; Accepted 22 September 2024
Available online 26 September 2024

although semantically rich and impactful, offline analyses usually lack timeliness in the context of attack discovery and mitigation [13].

Moreover, the assumption of having a labeled set of samples for new attacks in any feature domain may not be strictly realistic, limiting the effective usefulness of partially labeled ZSL-based detectors of new attacks [14]. For this reason, this paper presents a candidate modeling strategy for zero-day attack detection and categorization by framing the problem into a heterogeneous *Collective Anomaly Detection* (CAD) problem [13], where not only multiple observations are classified as anomalies, but these might also be clusterized into different classes.

The issue of heterogeneous CAD could be solved by importing recent deep clustering frameworks [15]. Still, two main reasons may impose a different strategy: (i) the Out-Of-Distribution (OOD) character of new attacks may require a meta-learning approach [16], and (ii) the computational resource restriction at the network's edge imposes favoring inductive biases over the depth of neural networks [17]. For these reasons, prototypical networks [18] and manifold learning [19] are imported in ASAP. The first end-to-end backpropagation pipeline for **A**utomatic **S**ynthesis of zero-day **A**ttack **P**rototypes.

Main contribution

This paper presents an end-to-end back-propagation approach for online-learning to concurrently detect multiple Zero-day Attacks (ZdA) while synthesizing their corresponding latent prototypes. Three main characteristics make of ASAP a relevant approach in the field of machine-learning based intrusion detection pipelines:

(1) The presented pipeline classifies both known and unknown traffic and is implemented using end-to-end back-propagation. Consequently, it permits online-learning and does not rely on manual feature selection/engineering.
(2) Prototypical learning is at the core of ASAP, importing the benefits of increased training data efficiency and enhancing compatibility with low-computational resource scenarios such as the IoT.
(3) With respect to canonical anomaly detection techniques, ASAP focuses on learning to simultaneously categorize a heterogeneous out-of-distribution set of anomalies into multiple classes.

Finally, it is worth noting that, to the best of the authors' knowledge, the three concurrent goals of ASAP, namely, multi-class classification, ZdA detection and clustering, constitute an original outcome for an end-to-end DL pipeline.

Outline. After clarifying some preliminary concepts in Section 3, and offering a summary of the related works in Section 2, this paper presents a detailed description of the proposal in Section 4. A use case study is then provided in Section 5, describing and discussing the obtained results. Finally, the work limitations and future research directions are pointed out in Section 6.

2. Related works

The authors of [8] relied on zero-shot learning to achieve ZdA detection. They used supervised learning counting on descriptions of seen classes and with parallel labels of unseen classes on another domain. In this work, autoregressive neural networks are used to translate between the feature domain and the semantic domain. More specifically, the authors of this work used prior knowledge of attacks whose samples were not included in the training dataset to compare their hidden representations with those of testing data at inference time. Similarly, the work in [20] uses manifold alignment to create cluster correspondences between different slices of the same datasets and between different datasets. The alignment between a full-labeled source dataset and a partially labeled target dataset performs zero-day attack inferences on the second. To assess their method, the authors performed experiments using different chunks of the NSL-KDD dataset.[4]

The authors in [21] introduced a hierarchical model with raw traffic passes through two stacked anomaly-detection modules. The first anomaly detection component separates known-benign traffic from potentially anomalous patterns. The former are filtered from the initial input, and the latter enters a second module, which operates ZdA detection. The authors of this work designed two threshold parameters related to the first anomaly criterion and the second confidence criterion that associates an anomaly to a known attack class or a potential ZdA. The work in [22] offers a two-step ZdA attack detection pipeline based on conditional and variational neural networks. The outputs of the first phase are related to a closed-set classification task, while in the second stage, the reconstruction errors of conditional autoencoders are used as a ZdA association score.

The authors of [23] presented a few-shot classification pipeline based on a fusion of the prototypical and graph convolutional networks. The raw bytes of flows' traffic captures were mapped to pixels on learning images. Few-shot learning was then performed on such image samples. The authors of this work also validated their algorithm with a fraction of the IoT-23 [24] dataset. The authors of [25] focused on few-shot intrusion detection framing it as a hyper-parameter optimization problem and using the model-agnostic meta-learning paradigm [26]. The authors of [27], instead, used capsule networks [28] in the feature learning stage to enhance the performance of prototypical FSL in the context of NID. In contrast to the few-shot settings of these works, the ASAP method infers unknown attacks on an open-set basis, i.e., without using labels at test time.

The authors of [29] presented a pipeline for intrusion detection that inverts the approach of [21]. First, a high-dimensional latent space is constructed, and a convolutional neural network divides inputs between benign and known attacks. Then, two anomaly-based classifiers analyze the traffic deemed as benign to identify oversights and novel attacks. The authors differentiate between novelty and OOD attacks and use their approach to detect only the former. Recent surveys on ZdA detection are available in [30–32].

The framework in [14] focuses on the heterogeneous categorization of OOD observations. It presented a hint-labeling strategy to extend the episodic learning in [18] to anomaly detection tasks. The *neural algorithmic reasoning* blueprint [33] is used to ZdAs considering a two-level taxonomy of attacks (macro-attacks and micro-attacks) and low-dimensional tabular data. A successive extension of this work [34] presents a two-step process where the neural processor is offline pre-trained on low-dimensional data to reach convergence on a fine-tuning stage using high-dimensional raw-packet bytes. ASAP represents the first step toward an end-to-end extension of these contributions in which a single-level taxonomy of attacks is considered.

The assumption of having second-domain labels for unknown attacks limits the effective applicability of ZdA detectors. For this reason, with respect to [8,20], ASAP disregards such a hypothesis. Moreover, this work not only seeks to combine closed and open-set classification [21,29,35], but it additionally addresses the categorization of multiple concurrent ZdAs. With respect to multi-step training approaches [22,34], ASAP focuses on end-to-end learning, augmenting training efficiency. In contrast to the few-shot settings of [23,25,27], the presented method is zero-shot because it does not need human-expert labels to categorize multiple possibly unknown attacks at test time.

3. Preliminaries

This section presents some preliminary concepts at the core working principles of ASAP. Table 1 contains a list of the abbreviations used in this paper, apart from the proper names of algorithms.

[4] https://www.unb.ca/cic/datasets/nsl.html

Table 1
Abbreviations used in this paper.

Abbreviation	Meaning
API	Application Programming Interface
ARI	Adjusted Rand-Index
ARP	Address Resolution Protocol
ANN	Artificial Neural Network
BCE	Binary Cross-Entropy
BN	Batch Normalization
CAD	Collective Anomaly Detection
C&C	Command-and-Conquer
DDoS	Distributed-Denial-of-Service
DoS	Denial-of-Service
DL	Deep Learning
FSL	Few-Shot Learning
GAT	Graph Attention Network
HPO	Hyper-Parameter Optimization
HScan	Horizontal Scan
IoT	Internet of Things
ML	Machine Learning
MML	Multi-Modal Learning
MLP	Multi-Layer Perceptron
NID	Network Intrusion Detection
OOD	Out of Distribution
PL	Prototypical Learning
PN	Prototypical Networks
RN	Relation Network
SDN	Software-Defined Networking
TNP	True Negative-Proportion
TPP	True Positive-Proportion
ZdA	Zero-day Attacks
ZSL	Zero-Shot Learning

3.1. Prototypical meta-learning

Prototypical Networks (PN) [18] offer a neural architecture strategy that decouples the classification task from the singular distributions of classes. PNs are biased toward learning a representational space where the Euclidean distances between latent vectors describe the class assignment of each inputs. A detailed explanation follows on how such a representational and discriminative mechanism works.

PNs are trained through *episodic learning*: given in input a set of *query* and *support* latent samples, PNs make a multi-class classification inference for each one of the former as a function of the labels of the latter. Let the input batch be represented by $\mathscr{B} = \{\mathscr{B}_S \cup \mathscr{B}_Q\}$ where $\mathscr{B}_S$ is the set of support latent vectors $\mathbf{z}^s_1, \mathbf{z}^s_2, \ldots, \mathbf{z}^s_{|\mathscr{B}_S|}$ and $\mathscr{B}_Q$ is the set of query latent vectors $\mathbf{z}^q_1, \mathbf{z}^q_2, \ldots, \mathbf{z}^q_{|\mathscr{B}_Q|}$. The class-wise centroids or *prototypes* are computed using the support latents:

$$\mathbf{c}_i = \frac{1}{N_i}\sum_{j=1}^{N_i} \mathbf{z}^s_j, \ \forall \mathbf{c}_i \ \in \mathscr{C}, \ \text{ s.t. } \ \mathbf{z}^s_j \in \mathscr{B}_S, \ \ \forall j \in \{i, \ldots, N_i\} \tag{1}$$

where N_i is the number of support latents in class i and $\mathscr{C}$ is the set of classes included in $\mathscr{B}$.

Successively, PNs build a classification logits vector for each query sample where the vector components are the association scores to each class. These scores are inversely proportional to the Euclidean distances between the latent representation of the query sample and the correspondent class prototype. ASAP uses PNs to perform multi-class classification of attacks. By doing so, the class prototypes learned in the PN framework are mapped to latent attack signatures. To obtain these signatures for unlabeled attacks, two additional steps are required, which are now explained.

3.2. Zero-day attacks

Zero-day attacks are new types of cyber threats that exploit vulnerabilities in software or hardware that have not been patched or fixed by the vendors [30]. Applying ML to detect unknown attacks is an open research area, mainly because statistical-based models lack OOD generalization capabilities [36]. ZdA detection can be seen, in fact, as an OOD anomaly detection task, where anomalies correspond to samples of zero-day attacks. The held-out data in OOD anomaly detection must come from a different set of classes concerning those used for training. For this reason, the attack classes used for training differ from those used for testing. Moreover, in the context of episodic learning, by the definition of ZdAs, the test ZdA input samples do not have a labeled support set [32].

3.3. Manifold learning for clustering-friendly latent spaces

Generalizable tasks, like meta-learning and OOD clustering, might require a strict alignment between the semantic relations in data and the geometrical distribution of the correspondent representations in the latent space. Manifold learning [19] refers to a set of techniques that steer the convergence of the latent-space toward such a goal. Inspired by the deep clustering research community [15], this paper uses both contrastive and associative regularization to induce the convergence of a latent space that permits clustering to OOD samples.

The working principle of this regularization mechanism is as follows. Given an input batch $\mathscr{B}$, the corresponding class one-hot labels $\mathbf{y}_i, \forall i \in |\mathscr{B}|$ are used to build a semantic adjacency kernel $\mathbf{A}$:

$$\mathbf{A} = \left[a_{ij}\right]_{i,j\in|\mathscr{B}|} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_{|\mathscr{B}|} \end{bmatrix} \times \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_{|\mathscr{B}|} \end{bmatrix}^T$$

On the other hand, pair-wise similarities α_{ij} between the latent representations $\mathbf{z}_i, \mathbf{z}_j \forall i, j \in |\mathscr{B}|$ are computed. The similarity distributions are then used to minimize the divergence between the predicted and ground truth similarity and distance distributions using cross-entropy:

$$\mathscr{L}_{AR} = -\frac{1}{|\mathscr{B}|}\sum_{i=1}^{|\mathscr{B}|}\left(\boldsymbol{a}_i^T \cdot \log(\boldsymbol{\alpha}_i) + (1-\boldsymbol{a}_i)^T \cdot \log(1-\boldsymbol{\alpha}_i)\right)$$
$$\text{where} \boldsymbol{a}_i = \begin{bmatrix} a_{i1} \\ a_{i2} \\ a_{i3} \\ \vdots \\ a_{i|\mathscr{B}|} \end{bmatrix} \ \text{ and } \ \boldsymbol{\alpha}_i = \begin{bmatrix} \alpha_{i1} \\ \alpha_{i2} \\ \alpha_{i3} \\ \vdots \\ \alpha_{i|\mathscr{B}|} \end{bmatrix} \tag{2}$$

By minimizing (2), the alignment between the semantic and latent-space geometrical relations of samples is maximized. In other words, the model is biased toward learning the distribution of similarities between the environment observations, which is denoted as $\mathscr{D}$. Being a distribution of a property which is relative to two or more objects, this distribution is said to lay on a higher level of abstraction with respect to the absolute features of inputs. Notice that $\mathscr{D}$ describes the regularities that permit to cluster elements in a way which is consistent with the semantic labels at hand. Note also that the existence and the learnability of $\mathscr{D}$ is a fundamental requirement toward generalizing collective anomaly detection over previously unknown observations. The next section gives a detailed description of the proposed pipeline.

4. The ASAP framework

This section describes the ASAP framework for the online automatic obtainment of ZdA signatures in the latent space through end-to-end back-propagation. The following description follows the schematic representation in Fig. 1.

4.1. State space

The observational or state space of ASAP is that of an IoT network and is represented by point (A) in Fig. 1. More specifically, the main

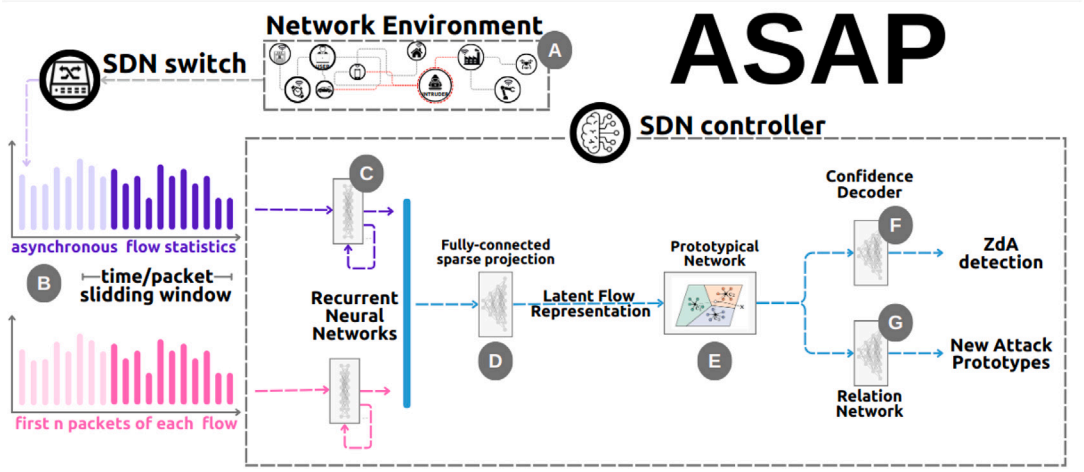


Fig. 1. Schematic representation of the ASAP framework. Features from two-domains are aggregated through time using per-domain recurrent layers. The last outputs of the recurrent layers are sent through sparse linear projections and concatenated to create the latent flow representations. The SDN controller is trained using prototypical classification and adjacency regression toward online detection and clustering of potential attacks out-of-distribution without any further supervisory signal.

components of information considered refer to network level flows, i.e., those having the same source and destination IP addresses, and are of two modalities:

(1) **Anonymized raw packet captures.** Our neural modules ingest the binary octets of network-level packets. An example of such binary octets from an IPv4 packet can be the following:

```
4500 003c 1c46 4000 4006 b1e6 c0a8
c0a8 0002 0050 0050 9fa5 17aa 0000
a002 7210 2d78 0000 0204 05b4 0101
```

Where the binary encoding has been changed to hexadecimal values for readability purposes. The first 64 octets are taken to include the network and transport-layer headers, as well as a part of the payload. Importantly, the IP addresses and transport ports of these packets are masked to prevent the neural modules from using them as discriminators [37]. To characterize each network-level flow, the bytes from the sequence of the last n packets is embedded on a feature tensor. In our experiments, the value of $n = 1$ held the best balance between accuracy and learning efficiency. Appendix A contains more details on how the value of n was chosen.

(2) **Flow statistics from edge forwarders.** ASAP assumes to operate over an SDN-compatible network scenario where an SDN switch sends network observations to an SDN controller. Specifically, a sliding-window sequence of flow statistics is provided by an SDN switch [38] through the *Flowstats* request and responses of the OpenFlow protocol [39], a popular southbound interface for SDN. To build the flow feature tensors, three features extracted from the OpenFlow API were used: byte count, flow duration in seconds, and packet count.

ASAP first processes features of each data-domain separately, as represented by point (B) in Fig. 1. For each data-domain and at any instant τ, a multi-dimensional set of inputs $\mathbf{X}^{\tau} \in \mathbb{R}^{N,d}$, is considered where N is the batch dimension and d is the feature dimension of the considered data modality. After concatenating these input vectors across the time dimension, a three-dimensional input $\mathbf{X} \in \mathbb{R}^{N,d,t}$ is built by stacking the last t matrices $\mathbf{X}^{\tau-t+1}, \ldots, \mathbf{X}^{\tau-1}, \mathbf{X}^{\tau}$. Notice that, if provided, more feature domains with respect to those mentioned above can be easily added in this setting. The interested reader is referred to [1] for experiments with a three-modality setting in which node-status features are incorporated to the input space.

4.2. Automatic and adaptive feature engineering

Online learning implies online and adaptive feature processing. To this end, a learnable Batch Normalization (BN) layer [40] is used in ASAP. During training, BN normalizes inputs using the feature-wise mean and variance computed from the current mini-batch of data. By normalizing the activations, batch normalization helps to reduce the change in the distribution of inputs to subsequent layers during training.

The automatic nature of feature engineering in ASAP derives instead from the fact that raw-bytes of packets are fed to the neural network which extracts suitable representations of these, by using only the gradients of the downstream loss functions. In this sense, the presented framework avoids manual feature selection and engineering, and delivers a neural machinery that can process network traffic of any type in the form of binary raw-bytes, as mentioned in Section 4.1.

4.3. Recurrent time-series processing

After passing the time-series inputs through BN, any type of recurrent module can be used to learn a collapsed representation for each sequence of inputs, leading to a two-dimensional representation $\chi \in \mathbb{R}^{N,h}$, where h is the latent-space dimension. In ASAP, a Gated Recurrent Unit (GRU) layer is used for this purposes. Such a recurrent module uses gating mechanisms to efficiently manage information flow, improving performance on sequential data tasks by mitigating issues like the vanishing gradient problem that were presented by other recurrent modules [41]. More detail on the GRU layer's implementation is provided in Appendix C.

The latent-space dimension is bigger that the dimension of the original feature space ($h \gg d$) to reduce the representational bottleneck and facilitate the separability of different classes in the latent space. Each data modality is processed by its own GRU layer, as schematized in point (C) in Fig. 1. Finally, the rows of every domain-specific matrix $\chi \in \mathbb{R}^{N,h}$ are concatenated to form a combined latent flow representation $\mathbf{z_i}$, $\forall i \in \{0, 1, \ldots, N\}$, as represented by point (D) of the same figure.

4.4. Prototypical learning

ASAP performs Prototypical Learning (PL) [42] over such latent representations. More specifically, for each query latent $\mathbf{z}_i^q$, the inverses of its distances to each prototype are computed to form a logits

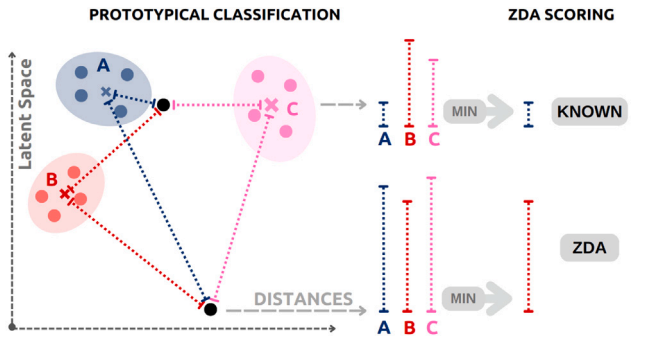


Fig. 2. Left: Prototypical classification uses a set of *support* labeled latent representations (colored dots) for each class to compute the classes' centroids or *prototypes* (colored crosses). For each *query* sample latent (black dots), the class association score is inversely proportional to the distances to each corresponding prototype (colored dashed lines). **Right:** For each query input sample, its ZdA score is proportional to the distance between its latent representation and the closer prototype. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

classification vector $\mathbf{l}_i$:

$$\begin{aligned} \mathbf{l}_i &= 1/([d_i^1, d_i^2, d_i^3, \dots, d_i^{|\mathscr{C}|}] + \epsilon) \quad \text{with:} \\ d_i^j &= \|\mathbf{z}_i^q - \mathbf{c}_j\|_2 \quad \forall j \in \mathscr{C} \end{aligned} \tag{3}$$

where ϵ is a little fixed value added for numerical stability purposes.

Successively, the multi-class cross-entropies between the predicted class assignations and the ground-truth labels of query samples are computed and averaged over the query batch:

$$\mathscr{L}_{cs} = -\frac{1}{|\mathscr{B}_Q|} \sum_{i=1}^{|\mathscr{B}_Q|} \mathbf{y}_i \cdot \log(\mathbf{l}_i) \tag{4}$$

where $\mathbf{y}_i$ is the one-hot label of the ith query sample, and the log operation is performed element-wise. The prototypical classification of flow latents is represented by point (E) in Fig. 1.

During training, the input samples are divided between benign traffic, known attacks, and unknown attacks, also called training ZdAs, which are replaced by a different set of ZdA classes at test time. Thus, to online detect and cluster OOD anomalies, the representational machinery in ASAP accommodates two additional outputs for each query input sample: first, a ZdA score for each query sample, and second, an adjacency matrix that clusters all the input batch into classes. These two additional outputs are now explained.

4.5. ZdA discrimination

Once the multi-class classification vectors $\mathbf{l}_i$, $\forall i \in |\mathscr{B}_Q|$ are obtained, the positions corresponding to training ZdAs are masked to zero, and the one's complement of such masked vectors are computed. The minimum values for each of the resulting complement vectors are then selected and passed through a Sigmoid function. The final values, μ_i $\forall i \in |\mathscr{B}_Q|$, are then used as ZdA scores for the corresponding query samples, as represented in Fig. 2 . Finally, the mean Binary Cross-Entropy (BCE) of the batch is computed and used as a loss signal for the anomaly detection task:

$$\mathscr{L}_{AD} = -\frac{1}{|\mathscr{B}_Q|} \sum_{i=1}^{|\mathscr{B}_Q|} \left(m_i \log(\mu_i) + (1 - m_i) \log(1 - \mu_i)\right) \tag{5}$$

where m_i is a binary label indicating the ZdA condition of the ith sample in $\mathscr{B}$. Note that masking the ZdA positions in the classification vectors is a key step of this training framework: the class labels of ZdAs are not *used* to discriminate them from known attacks during training because they are assumed to be absent at test time. The ZdA classification is also represented in point (F) in Fig. 1.

4.6. Adjacency regression

The final step of the pipeline involves clustering anomalous traffic. To this end, ASAP uses a Relation Network (RN) [43] which operates on the latent input representations $\mathbf{z_i}$, $\forall i \in \{0, 1, \dots, N\}$ in the following manner. First, the absolute value of element-wise subtraction between the latent vectors is computed. By doing so, a distance vector $\mathbf{d}_{ij} \in \mathbb{R}^h$ is obtained for every pair of inputs. These vectors are then passed through a Multi-Layer Perceptron (MLP), specifically implemented as two fully-connected neural maps with a LeakyRelu activation in between that reduces each vector $\mathbf{d}_{ij}$ to be mapped to a scalar value α_{ij}:

$$\tilde{\mathbf{d}}_{ij} = |\mathbf{z_i} - \mathbf{z_j}|, \ \forall i, j \in \{0, 1, \dots, N\} \tag{6}$$

$$\alpha_{ij} = \mathbf{W}_2 \cdot \sigma\left(\mathbf{W}_1 \cdot \tilde{\mathbf{d}}_{ij} + \mathbf{b}_1\right) + \mathbf{b}_2 \tag{7}$$

where $\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2$ are the learnable weights and biases matrices, and their dimension is consistent so as to project $\mathbf{d}_{ij}$ to a one-dimensional space. Overall, the RN ingests pairs of latents and predicts if such a pair belongs or not to the same class. ASAP uses the set of α_{ij} and the set of one-hot labels to perform clustering as explained in Section 3.3. The clustering or relation inference part is represented by point (G) in Fig. 1.

Finally, the three loss terms in (4), (5), and (2) are summed and minimized by performing gradient descent with respect to the whole pipeline parameters (i.e., end-to-end gradient descent). By doing so, the prototypical latent space is aligned with **A** as much as possible, which otherwise would not happen.

5. Experiments

This section provides experimental evidence of the effectiveness of ASAP in the form of a case study. A CPU-only back-end and IoT traffic captures are used to test the validity of the proposed framework in a resource-constrained scenario, with the concurrent presence of known and unknown malicious traffic. The following paragraphs give more detail on the experimental settings and a description of the obtained results.

5.1. Virtual testbed

A virtual testbed was implemented to test the convenience of the proposed neural machinery in an online learning and zero pre-processing setting. The testbed is hosted on a GNS3 v 2.2.46 [44], an open-source robust network emulation software compatible with node containerization. The GNS3 server runs on Ubuntu 22.04.3 LTS. A set of attacker and victim nodes are implemented using Docker containers [45] and connected through an SDN-compatible switch.

The SDN controller is implemented using POX [46], a Python-based open-source library that helps issue OpenFlow commands to an SDN-compatible switch. The "Gar" branch of the official POX repository is used to handle compatibility with Python 3.8.18. The layer-3 forwarding boilerplate code of such a repository is extended. Such a code implements Address Resolution Protocol (ARP) request/response and 2-tuple flow matching rules (a flow comprises a source and destination IP).

5.2. Details on the neural module implementation

A Hyper-Parameter Optimization (HPO) procedure was made to choose the most suitable neural architecture. Also, the time sequence length t was set to 10, the number of raw packets included in each time-instant sample, n, was set to 1, hidden state dimension, h, to 800, and the number of raw packet bytes per packet was set to 64. Note that, rigorously speaking, these hyper-parameters could be conceived

Table 2
Hyper-parameter values and Neural Architecture Specifications.

Hyperparameter	Specification
Learning Rate (for all modules)	0.001
Support samples per class (K-SHOT)	5
Query samples per class	20
GRU Recurrent Layers	1
Hidden space dimension (h)	800
Evaluation Batches	50
Loss Function (Closed set classification)	Multi-class Cross-Entropy
Loss Function (Open set classification)	BCE
Optimizer	Adam (no weight decay)
Experience Replay Max Length	1000
Train-Evaluation split (for known traffic)	80% 20%
Relation network's MLP layer count	2
RN MLP layer dimensions	(h_dim, h_dim // 2, 1)
RN activation function	LeakyRelu(alpha=0.2)
Num. of packets in the feature vector	1
Num. of flow stats timesteps in the feature vector	10
Num. of. bytes for packet features (f)	64
Min. experience tuples for batch training (m)	20

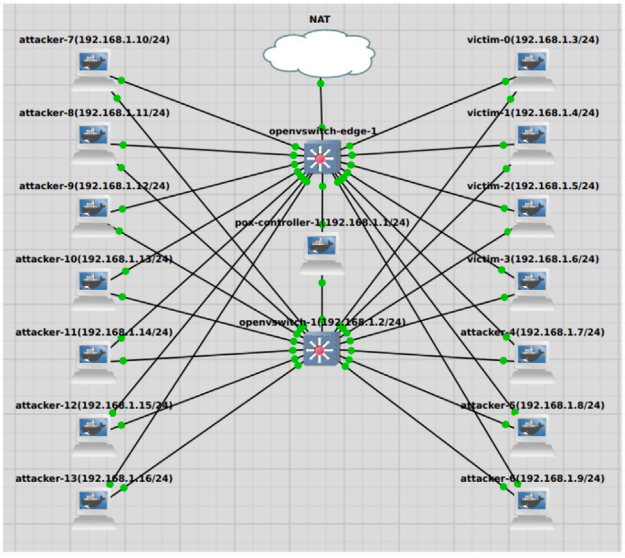


Fig. 3. The network emulation is done over GNS3. An SDN IoT environment is configured using Docker containers that replay attacker and honeypot behavior. The SDN controller online trains the neural modules to perform attack-related inferences. The Network Address Translation (NAT) node is added for train logging purposes.

as manual feature selection. Future work might focus on online adapting these hyper-parameters when needed.[5] Appendix A contains more details on the HPO procedure.

The DL-based inference modules are implemented using the PyTorch deep learning library, version 2.2.0-CPU, and are located inside the SDN controller. In particular, the 1-layered GRU module, the batch normalization layer, and the relation network's MLP were implemented using the corresponding reference implementations in PyTorch. Table 2 contains a full list of the hyper-parameter values and architectural specifications of the implemented neural modules. These values hold on every experiment unless explicitly specified.

5.3. Topology details

The network topology implemented comprises 17 nodes: 10 attackers, 4 victims, 1 SDN controller, and 2 SDN switches. All these nodes concurrently share the 20 × 2.19 GHz virtual cores at the disposition of our hardware back-end. Victims and attackers are based on the Ubuntu Focal Docker image, while the controller is based on the Python 3.8 image. The switches are based on the OpenvSwitch [47] images in the GNS3 Docker Hub. A schematic view of the topology from the GNS3 GUI is available in Fig. 3.

5.4. Realistic IoT traffic and attack patterns

The Aposemat IoT23 traffic captures in [24] were used to reproduce the attacker and honeypot behavior of an IoT network. The IoT23 dataset contains various DDoS attacks such as *Gafgyt*, *Hakai*, and *Mirai*, *Command and Conquer* (C&C) malware such as *Torii* and *Okiru*, and other IoT related malware such as the *Hajime* trojan, the *Muhstik* worm, among other attack traces. In our experiments a total of ten attack types were considered including those previously mentioned. More specifications on the attacks used and the exact number of traces per attack is provided in Appendix B.

The set of used attacks is denoted by $\mathscr{A}$. Before starting any training procedure, an arbitrary partition of $\mathscr{A}$ must be defined as follows:

$$\mathscr{A} = \mathscr{A}_K \cup \mathscr{A}_U^{train} \cup \mathscr{A}_U^{test} \tag{8}$$

where each element of the partition corresponds to a disjoint subset of attacks and has a specific role in the training and evaluation phase of ASAP:

(1) The set of known attacks, $\mathscr{A}_K \subset \mathscr{A}$ is used to train and evaluate known-attack classification. Notice only the labels of the samples in $\mathscr{A}_K$ will be used to compute the multi-class loss function in (4). The traffic samples of each attack in $\mathscr{A}_K$ will be used either for train or evaluation purposes, but not for both.
(2) The ZdA detection task is learned using $\mathscr{A}_K$ and $\mathscr{A}_U^{train}$. Where the latter indicates the set of *training ZdAs*. Each sample in $\mathscr{A}_K$ is given a non-ZdA label while samples in the $\mathscr{A}_U^{train}$ attacks are labeled as ZdAs. Both of these binary labels are used to compute and back-propagate the ZdA discrimination loss in (5) during training.
(3) Finally, the ZdA discrimination is evaluated using $\mathscr{A}_U^{test}$, which contains samples from attacks that were not seen during training, i.e., OOD samples. These samples are also labeled as ZdAs but are used only during evaluation.

Notice that any partition that follows (8) is a potential curriculum for training and evaluating ASAP. Two considerations can be made regarding the significance of different partitions. First, from a field-expert's perspective, different attacks might be more amenable to be assigned to the different subsets of $\mathscr{A}$, depending, e.g., on the multiple levels of knowledge maturity acquired by the community regarding different attacks. Second, from a pure information-theoretical perspective, the partition that maximizes the divergences in the feature distributions between train and test attacks could demonstrate a greater generalization power of the presented pipeline. The experiments made in this paper were done over three randomly created curricula as specified in Appendix B, letting both of the mentioned considerations as future work.

The traffic captures were analyzed using the dataset's provider instructions to filter and extract the malicious traffic. Refer to Appendix B for more details on these analyses. Importantly, to ensure a realistic setting, the malicious flows were reproduced at the same rate they were recorded using the TCPReplay software [48]. By doing so, the potential low-frequency attacks are correctly characterized.

5.5. Online labeling and training strategy

In the experiments, each attacker is assumed to perform only one type of attack. With this assumption, the labeling strategy during training consists of hard-coding the attackers' IP addresses with the

[5] Such a necessity did not arise during the experiments hereby exposed.

corresponding attack label in the controller node. Annotations are used to compute the training losses as explained in Section 5.4.

The presented pipeline performs online-learning of each one of its functional features. Note, however, that the efficiency of this learning paradigm is subjected to the actual throughput of the observed data, and can thus be potentially slow, e.g. in the case of low-frequency attacks. To this respect, adding an Experience Replay (ER) [49] mechanism proves useful to accelerate convergence: at each time-step, ASAP stores the received set of inputs and labels into a memory buffer to accumulate training data. These data are then periodically sampled to perform offline batch-learning[6] aside of the online learning activity.

Notice that instantiating one buffer per class permits the balanced sampling of support and query samples for each known class to implement episodic learning. The set of class buffers in ASAP is dynamic in that it can create new buffers for new classes at runtime, provided that a new labeled IP sends traffic.

ZdA detection requires careful evaluation. Specifically, even if the training accuracy in ZdA detection might be perfect, the model could perform poorly on the evaluation domain given its OOD characteristic. To this respect, an online-evaluation method has been implemented in ASAP: Taking into consideration a fixed partition as defined in (8), two separate replay-buffers are created for each attack in $\mathscr{A}_{\mathscr{K}}$. The first one is used for training purposes, while the second holds test samples. By doing so, test-data batches can be sampled periodically from the test buffers, and from the buffers in $\mathscr{A}_U^{test}$ to compute the accuracy in the evaluation domain. If substantial overfitting is observed, i.e., the evaluation accuracy is worsening as the train accuracy grows, the hyper-parameters of the pipeline can be changed and the training can be restarted.[7]

5.6. Baselines

ASAP is compared with four baselines to put in evidence the role of each one of its design choices. The key properties and the experimental role of such baselines is now explained.

(1) GATV2. This baseline is identical to ASAP except for the architecture of the similarity metric module. More specifically, the MLP of the relation network, which performs the same transformation over all inputs, is substituted by the attention mask of a *dynamic graph attention* layer as defined in [50], which implements different transformations to each sample in the batch. Comparing ASAP with GATV2 aims to answer the following question: *can adjacency regression, as defined in Section 4.6, benefit from using a set-to-set transformation as opposed to a fixed one?* More specific implementation details of the attention network used in this setting is given in Appendix D.

(2) SKR. Standing for *simple kernel regression*, this pipeline does not use the RN's MLP to cluster points but rather operates clustering directly over the classification latent space. More specifically, the only difference between SKR and ASAP is the fact that the former substitutes the RN's logic in (6) and (7) by:

$$\alpha_{ij} = \sigma\left(\frac{1}{\|\mathbf{z_i} - \mathbf{z_j}\|^2 + \epsilon}\right), \ \forall i, j \in \{0, 1, \dots, N\} \tag{9}$$

where $\|\cdot\|^2$ indicates the Euclidean distance, σ is a Sigmoid activation function, and ϵ is an arbitrarily-small fixed real-value added for stability purposes. This baseline is used to test if the transformation made by the RN in ASAP is necessary at all.

(3) NOREG. This baseline is identical to SKR except that it does not back-propagate the gradients of (2). The rationale of this baseline is that the loss term in (2) uses only the labels of known traffic, potentially biasing the neural modules to overfitting the training distribution. By not back-propagating the gradients of (2), this baseline should help understand the potential disadvantages of using (2) on the multi-class classification and ZdA discrimination tasks, as well as the importance of it for the clustering task.

(4) MONO. This baseline also runs the ASAP training and evaluation pipeline but discards multi-learning: it does not use raw-packet bytes, but only the flow-related features. This baseline helps to understand if and how is the second domain of features helpful for learning the three goals of ASAP.

5.7. Metrics

The performance in the DL pipelines in the experiments is measured using the following metrics. For the multi-class classification of currently-known attacks, the **accuracy** measure defined in Eq. (10) is used, where the number of *total predictions* corresponds to the known attacks in each batch/epoch:

$$\text{Acc} = \frac{\text{Correct predictions}}{\text{Total predictions}} \tag{10}$$

For ZdA detection purposes, the **balanced accuracy** is used as defined in Eq. (11):

$$\hat{\text{Acc}} = \frac{\text{TNP} + \text{TPP}}{2} \tag{11}$$

Where TNP is the *true negative proportion* and is defined as the ratio of predicted negatives and the total number of negatives in the batch/episode:

$$\text{TNP} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}} \tag{12}$$

Conversely, TPP is the *true positive proportion* and is defined analogously:

$$\text{TPP} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{13}$$

The reason for using (11) is that ZdAs – the *positive* samples in the open-set classification – represent an unbalanced class. Consequently, using (10) would result in a biased comparison.[8]

For the *anomaly clustering* task, the *Adjusted Rand Index* (ARI)is used:

$$\text{ARI} = \frac{\mathscr{R} - \mathbb{E}[\mathscr{R}]}{\mathscr{R}_{max} - \mathbb{E}[\mathscr{R}]} \tag{14}$$

where $\mathscr{R}$ is the *rand index*, as is defined as:

$$\mathscr{R} = \frac{(a+b)}{\binom{N}{2}} \tag{15}$$

where N is the number of samples in the batch/episode, a is the number of pairs of samples that are correctly assigned to the same cluster, and b is the number of pairs of samples correctly assigned to different ZdAs. Notice that (14) is an adjustment of (15) that considers the potential chance-based agreement between predicted and ground-truth clusters. More information about the ARI metric is in [51].

5.8. Results and discussion

The upper row of Fig. 4 contains Principal Component Analysis (PCA) decompositions of the progressive evolution of the latent space during training. Dots correspond to latents, and their colors correspond

[6] The class-wise experience buffer is sampled after a prefixed number m of experience tuples is collected, where m is a user-defined hyper-parameter. The experiments documented in this paper used $m = 20$, as it resulted on a fair balance between training efficiency and computational overhead.

[7] Future works could implement an online adaptation of the pipeline to different hyper-parameters.

[8] In cases of high unbalance, e.g. 90%/10%, A deterministic negative predictor could achieve better performance than any other baseline.

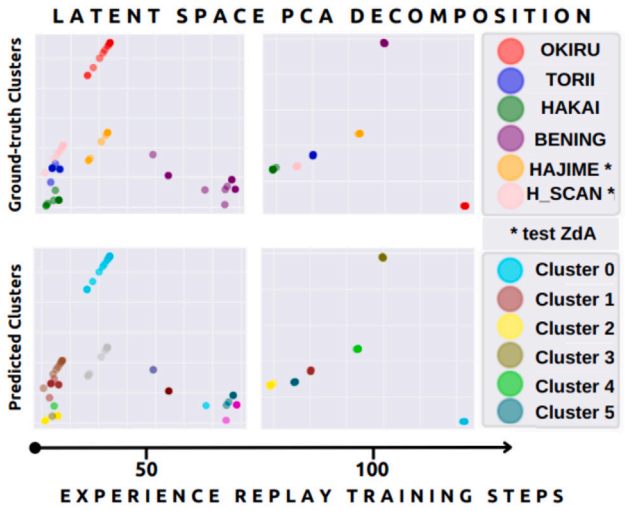


Fig. 4. Two principal component decomposition of the latent space evolution during training. Upper row: colors indicate ground truth labels of different classes. Lower row: Predicted associations for inputs without considering labels. (Colors do not indicate absolute class assignation, but unlabeled cluster assignments). ASAP learns to differentiate among a labeled set of attack classes and to cluster multiple types of unlabeled samples on a forward pass. By doing so, the potential prototypes of new attacks can be automatically obtained. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

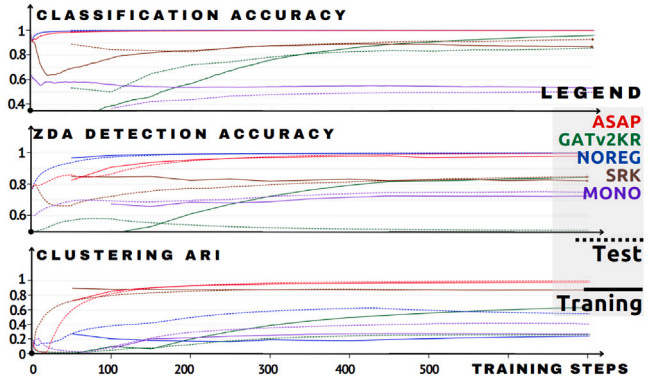


Fig. 5. Experimental results for online learning to classify known attacks (upper line plot), detect Zero-day Attacks (middle), and clustering of known and unknown traffic (lower). Train and test metrics are aggregated per batch. Continuous lines correspond to evaluation metrics, while dashed lines correspond to training metrics. The evaluation data corresponds to out-of-distribution samples.

to the traffic classes. These plots evidence how same-class samples are represented more concisely as training progresses and the inter-cluster distances augment. Such a behavior in the evolution of the latent space can be explained by the minimization of (4) and the prototypical architecture (class assignation is biased toward the nearest class support centroid in the latent space).

Moreover, using a combined state space, the multi-class classification accuracy is generalized OOD, as reflected by the upper line plot of Fig. 5. In this plot, the mean multi-class classification accuracies as defined in (10) are tracked with in-distribution (continuous lines) and out-of-distribution classes (dashed lines) as training progresses. Importantly, the unique baseline not converging to a good classification accuracy in the test (OOD) domain is MONO, which does not use packet features.

The ZdA detection task (middle line-plot in Fig. 5) follows a more *global* alignment between the latent space geometry and the labels of samples. For a ZdA to be detected, its nearest centroid in the latent space must be relatively long with respect to that of known samples, as explained in Section 4. Notice, however, that such a geometrical inductive bias for the ZdA inference is more vulnerable to eventual overfitting of the training domain, i.e., to a latent space which is excessively manipulated to separate classes in the training regime. In fact, though not directly related to ZdA detection, the prototypical-based minimization of (2) could have an adverse effect on the latent space as it is computed only using information of training-time classes.

As can be seen in the middle line plot of Fig. 5, the unique algorithms that preserve a good accuracy in test ZdA detection are those counting on the relation network for adjacency regression. The relation network – and GATV2 kernels – permit in fact to decouple the vector space over which (2) is minimized from the one over which (4) and (5) are minimized. However, even if the GATV2 kernel regressor shows to have a modest ZdA detection accuracy at train time (continuous green line), it is unable to classify test ZdAs (dashed green line), perhaps because of the larger parameter space of this architecture, that might need more data to avoid overfitting the training ZdA setup.

The most difficult task for all the baselines is back-propagation-guided clustering of OOD anomalies. The latent space regularization in (2) benefits precisely this task, as can be seen in the lower line plot of Fig. 5, which tracks the evolution of train and test clustering ARI values as defined in (14) during learning evolution. This plot evidences that ASAP and SKR are the unique algorithms that converge to good clustering scores. Note also that the relation network in ASAP helps outperforming SKR in the overall classification accuracy. Moreover, with the exception of ASAP and SKR, every algorithm worsens the ARI score at test time compared to its train scores.

The worsening of test-time clustering may indicate that a translational distribution shift exists between the global geometry of train and test clusters. Interestingly, the SKR algorithm, which clusters samples using a global learned threshold of the latent Euclidean distances, has lower clustering performance compared to ASAP even during training. This result might suggest that different clusters might have different densities in the latent space of SKR. Again, ASAP decouples the class-assignation latent space from the clustering latent space, making the latter converge to a more clustering-friendly one, i.e. where intra-cluster densities and inter-cluster separations might be more regular. Such a claim is backed up by the scatter plots of the lower row of Fig. 4, where dots correspond to PCA decomposition of latents and colors correspond to the cluster assignation predictions made by the relation network at the head of ASAP.

ASAP performs collective anomaly clustering with high accuracy. In terms of the manifold learning explained in Section 3.3, the experiments show that, in the realistic IoT scenario at hand, $\mathscr{D}$ exists and is learnable. Moreover, it is shown that such a distribution is learnable by having the sole information of known attacks. Equivalently, it is shown that in the latent space learned by ASAP, the densities of unknown clusters are comparable to that of known clusters, and the patterns of inter-clusters distances are stable between known and unknown clusters. To this respect, future work could consider the extent to which $\mathscr{D}$ exists among more heterogeneous sets of attack labels with respect to those used in our experiments.

A final observation is made regarding training data efficiency, defined as convergence speed during training. As can be seen in Fig. 5, NOREG shows the same convergence speed for closed set classification and improved ZdA detection with respect to ASAP. Still, ASAP is the unique algorithm to show a fair balance between OOD generalization and training efficiency in all three tasks (classification, anomaly detection, and clustering), which are taken into account. Considering that the 700 training steps monitored in Fig. 5 correspond to two hours of training, it can be noticed that the convergence plateau for the ZdA detection task, the most challenging task for ASAP, is reached at a maximum of 1 h of exposition to malicious traffic.

Notice the empiric efficiency gains observed in these experiments are also backed up by solid theoretical considerations: by using only the pair-wise similarities α_{ij} between the latent representations, the loss function in (2) biases the action of gradients toward the parameters of the relation network's MLP, i.e., to learn a manifold relative to the similarity distribution among elements. Such a *relational inductive bias* has not only proved important for steering neural networks to reason over a higher level of abstraction [52,53], but has also been recently validated by concurrent work as a sample-efficiency booster for neural modules [54].

6. Conclusive remarks

This paper presented a DL-based recipe for online learning to detect known patterns and reveal concurrent clusters of novel (unlabeled) anomalous observations. Beyond a typical offline trace-driven experiment, the online-learning abilities of ASAP are tested on a carefully designed virtual cyber-ranch that emulates a realistic IoT scenario where known and unknown attacks are launched. Another feature that the design of ASAP concentrates on is zero-preprocessing: the inferences leveraged by the presented pipeline are made over raw traffic bytes and OpenFlow SDN-based monitoring. By doing so, the proposed framework proves to be adaptive by design and portable to real network scenarios.

Future research directions include measuring and, eventually, augmenting the scalability of ASAP to handle multiple forwarding nodes with a single smart controller. Further investigation could also explore how wider definitions of the state space could be used, for example, by using health-status probes from nodes to enrich the information carried by the attack prototypes.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work was supported in part by the SERENA-IIoT project, which has been funded by MUR (Ministero dell'Università e della Ricerca) under the PRIN 2022 program (project code 2022CN4EBH), and in part by project SERICS (project code PE00000014), under the NRRP MUR program funded by the EU - NGEU.

Appendix A. Hyper-parameter optimization

In Fig. 6, the different accuracy metrics (multi-class classification, ZdA detection and clustering) during evaluation episodes are plotted as a function of the hidden space dimension. In these experiments, a hidden dimension of size 400 obtains good results with the best training efficiency.

The observation sequence length ingested by the recurrent modules has also been varied and the inference-time accuracies for each task were registered and plotted in Fig. 7. It was observed that bigger sequences of flow statistics result in better feature engineering. However, adding more than the last ten historical statistics for a flow does not substantially improve accuracy in any task. Importantly, automatic averaging of time-series observations significantly improves the attack classification accuracy, suggesting that better attack prototypes are formed with at least 50 s of flow observation.

Lastly, in Fig. 8, the number of packets that the recurrent model in ASAP ingests to form the latent representation of a flow is varied from one to three. Classification and ZdA detection accuracy benefit from longer sequences of packets to form a more complete attack prototype. Instead, the clustering machinery does not benefit (but rather worsens its ARI) from adding packets to the sequence of ingested features.

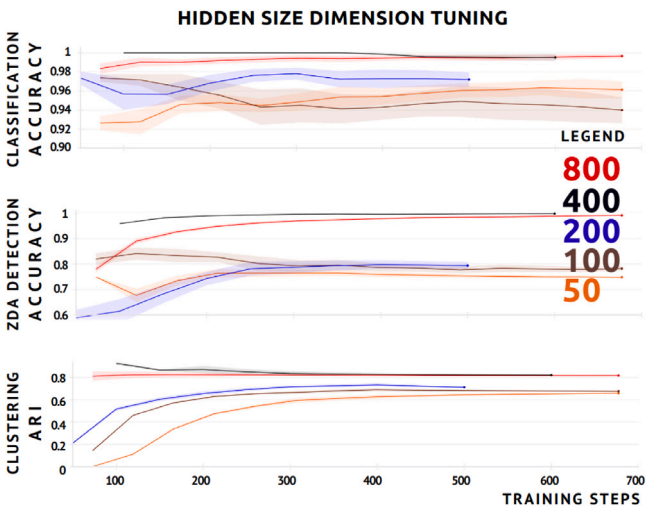


Fig. 6. Latent space dimension optimization. Evaluation accuracy metrics as a function of the latent space dimension. Lineplots correspond to the mean aggregation of accuracies/ARI per evaluation episode. The shadows correspond to the minimum and maximum values of the evaluation batches.

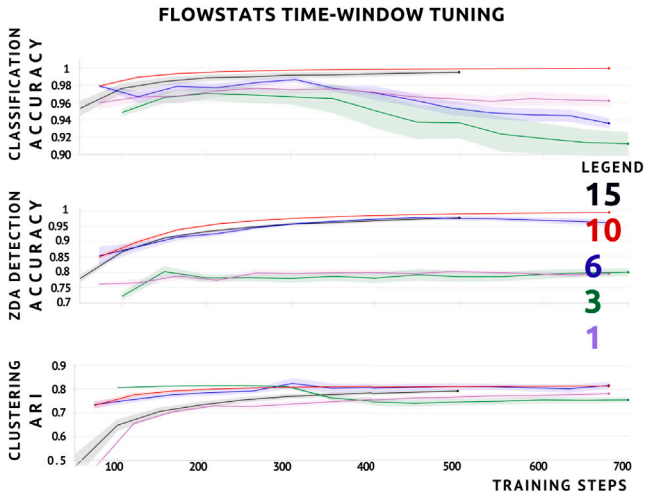


Fig. 7. FlowStats Time-window Optimization. Evaluation accuracy metrics as a function of the number of successive flow statistics (and packet features) ingested by the recurrent encoder. (For a hidden space dimension of 400.

Appendix B. Attacks and bening IoT traffic details

The Aposemat IoT23 dataset [24] was used in this work to reproduce realistic IoT attack and honeypot patterns. The following attacks were extracted from the original Pcap files that are publicly available.[9] Unless explicitly stated diversely, these flows were extracted using the attacker origin IP address as the filtering criterion. Recall that source and destination IP addresses and transport-layer ports were online masked before feeding the neural modules with raw packet bytes.

(1) **Hajime:** This Trojan malware searches to exploit Linux-related vulnerabilities. 5e4 of such flows were extracted from the dataset's capture 9.1.
(2) **Hakai:** This is a distributed denial-of-service (DDoS) botnet, a specialization of the Mirai malware. (Extracted from dataset's capture 8.1.) 1.2e4 flows were extracted.

[9] https://www.stratosphereips.org/datasets-iot23

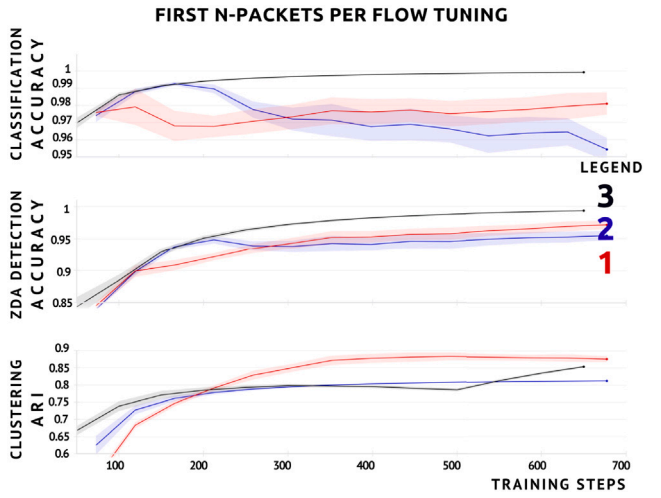


Fig. 8. Optimization of the number of first packets per flow that form the attack prototypes. Evaluation accuracy metrics as a function of the number of raw (anonymized) packets per flow ingested by the recurrent encoder. (For a hidden space dimension of 400).

(3) **Gafgyt:** This is another more general DDoS botnet. (Dataset's capture 60.1.) 5e4 flows extracted.
(4) **Mirai:** This is an open-source DDoS attack specially used over IoT devices. Capture: 34.1. Flows: 2.4e4.
(5) **Torii:** A Command and Conquer (C&C) and Information Gathering malware. Capture: 20.1. Flows extracted: 5e4.
(6) **Muhstik:** A worm based on the Mirai Botnet. Among others, it targets IoT devices. Commonly used to mine cryptocurrency and perform DDoS attacks. Capture: 3.1. Flows extracted: 5e4.
(7) **Okiru:** Another C&C Botnet that targets ARC processors, commonly used in wearables, and medical IoT, among others. Capture 7.1. Flows extracted: 5e4. The criteria for extracting these attack flows included target source and destination transport ports.
(8) **Horizontal Scan:** 5e4 traffic flows related to generic Horizontal Scan (HScan) were extracted from dataset's capture 1.1.
(9) **C&C HeartBeat:** Generic heartbeat-related server-side flows were also extracted in the context of the C&C traffic. Capture 7.1. Flows extracted: 0.15e4.
(10) **Generic DDoS:** Also, a set of 5e4 generic DDoS-related flows were extracted in the context of the C&C traffic from capture 7.1.

The Honeypot devices used by the authors of the IoT23 captures were used in the experiments to emulate honeypot IoT devices used as attack victims. More specifically, these honeypots were the following:

(1) **Somfy door lock device:** All the flows contained in the first three captures of folder Honeypot7.1 were extracted. These flows are related to a smart door lock device. In the implemented topology, two nodes reproduced these flows.
(2) **Philips HUE smart LED lamp:** These flows were extracted from the folder Honeypot4.1, and reproduced by a unique node.
(3) **Amazon Echo home intelligent personal assistant:** These flows were extracted from the folder Honeypot5.1. One node reproduced these flows.

The interested reader is referred to [24] for more details on the dataset. All the flow extraction code used for the experiments is available in the form of a GitHub Gist.[10]

[10] https://gist.github.com/QwertyJacob/e33e68f230d5ecec8b6f2524ab166ebf

Train and test curricula

As explained in Section 5.4, the ten attacks used in our experiments were grouped under the set $\mathscr{A}$, which is then partitioned following (8) to form a training-evaluation curricula for ASAP. In our experiments, three different partitions of $\mathscr{A}$ were used to evaluate our hypotheses:

(1) $\mathscr{A}_K$ = {C&C HB, Gen-DDoS, H-Scan}, $\mathscr{A}_U^{train}$ = {Hakai, Torii, Mirai, Gafgyt}, and $\mathscr{A}_U^{test}$ = {Hajime, Okiru, Muhstik}
(2) $\mathscr{A}_K$ = {Hakai, Torii, Okiru}, $\mathscr{A}_U^{train}$ = {C&C HB, Gen-DDoS, Mirai, Gafgyt}, and $\mathscr{A}_U^{test}$ = {Hajime, Horizontal Scan, Muhstik}
(3) $\mathscr{A}_K$ = {Muhstik, Hajime, Mirai}, $\mathscr{A}_U^{train}$ = {C&C HB, Gen-DDoS, Okiru, H-Scan}, and $\mathscr{A}_U^{test}$ = {Hakai, Gafgyt, Torii}

Appendix C. The Gated Recurrent Unit

A Gated Recurrent Unit is a type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem commonly encountered in traditional RNNs. It achieves this by utilizing gating mechanisms. Specifically, the *update gate* and the *reset gate*, that regulate the flow of information within the network. These gates allow the GRU to maintain and update hidden state representations efficiently, making it particularly well-suited for tasks involving sequential data, such as time-series analysis and natural language processing.

More in detail, the GRU layer operates over a sequence of inputs $x_0, x_1, x_2, \ldots, x_t$ on a sequential manner. As any recurrent model, at any instant t, this module is fed not only with the input observations x_t but also with some recurrent state h_{t-1}, which in the case of the GRU, coincides exactly with the output of the previous instant. The detailed neural architecture of the GRU layer is given below:

(1) **Reset Gate**:

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr})$$

(2) **Update Gate**:

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz})$$

(3) **New Gate**:

$$n_t = \tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{(t-1)} + b_{hn}))$$

(4) **Final Output**:

$$h_t = (1 - z_t) \odot n_t + z_t \odot h_{(t-1)}$$

In the above equations, σ denotes the Sigmoid activation function, $\odot$ represents element-wise multiplication or Hadamard product, and every W, and b correspond to a particular learnable parameter matrix for the gates.

Appendix D. Dynamic graph attention network implementation details

The Relation Network in ASAP can be implemented in lots of ways. Graph Neural Networks (GNN) [55] are a good architectural alternative to the MLP for learning sparse relationships among data. GNNs operate on a graph-format input batch through parametric combinations arising from the topology of the input graph. In this work, such a graph is constructed using the class labels at hand. Specifically, suppose $\mathbf{z}_u \in \mathbb{R}^m$ is the latent representation of an input sample x_u. In that case, a GNN will map it to a representation $h_u \in \mathbb{R}^m$, as a function of the latent representation vectors of the same-class samples or neighbors of x_u.

The Dynamic Graph Attention Networks (GATV2) [50] implement this encoding with an attention mechanism that assigns different weights to the different neighbors of x_u, as a function of their features:

$$h_u = \sum_{v \in \mathcal{N}_u} \text{softmax}_{\mathcal{N}_v}(\mathbf{a}^T \sigma(\mathbf{W} \cdot [\mathbf{z}_v + \mathbf{z}_u])) \cdot \mathbf{W}\mathbf{z}_v \tag{16}$$

where σ is implemented as a LeakyReLU with $\alpha = 0.2$, and $\mathcal{N}_u$ is the set of neighbors of x_u. In contrast to other attention mechanisms, this attention layer $\mathbf{a}^T \in \mathbb{R}^m$ permits the attention scores to vary also as a function of $\mathbf{z}_u$.

In our experiments, the GATV2 baseline is identical to ASAP except for substituting the adjacency regression in (6) and (7), by the attention module in (16). Specifically, the support labels initialize the neighborhoods of elements in the graph, and unlabeled observations are initially set as connected to all the other nodes. Finally, the learned attention kernel $\mathbf{a}$ is taken as the adjacency prediction.

References

[1] J. Cevallos, S. Iannello, G. Grattacaso, A. Rizzardi, S. Sicari, A. Coen-Porisini, Smartville: an open-source sdn online-intrusion detection testbed, 2024, Available: https://github.com/DISTA-IoT/smartville.

[2] R. Samrin, D. Vasumathi, Review on anomaly based network intrusion detection system, in: 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques, ICEECCOT, IEEE, 2017, pp. 141–147.

[3] O.A. Wahab, Intrusion detection in the iot under data and concept drifts: Online deep learning approach, IEEE Internet Things J. 9 (20) (2022) 19 706–19 716.

[4] S.K. Amalapuram, A. Tadwai, R. Vinta, S.S. Channappayya, B.R. Tamma, Continual learning for anomaly based network intrusion detection, in: 2022 14th International Conference on COMmunication Systems & NETworkS, COMSNETS, IEEE, 2022, pp. 497–505.

[5] Y. Wang, Q. Yao, J.T. Kwok, L.M. Ni, Generalizing from a few examples: A survey on few-shot learning, ACM Comput. Surv. (csur) 53 (3) (2020) 1–34.

[6] G. Andresini, A. Appice, N. Di Mauro, C. Loglisci, D. Malerba, Multi-channel deep feature learning for intrusion detection, IEEE Access 8 (2020) 53 346–53 359.

[7] P. Lin, K. Ye, Y. Hu, Y. Lin, C.-Z. Xu, A novel multimodal deep learning framework for encrypted traffic classification, IEEE/ACM Trans. Netw. (2022).

[8] Z. Zhang, Q. Liu, S. Qiu, S. Zhou, C. Zhang, Unknown attack detection based on zero-shot learning, IEEE Access 8 (2020) 193 981–193 991.

[9] L. Yu, L. Xu, X. Jiang, A high-performance multimodal deep learning model for detecting minority class sample attacks, Symmetry 16 (1) (2023) 42.

[10] S.S. Jazaeri, S. Jabbehdari, P. Asghari, H. Haj Seyyed Javadi, Edge computing in sdn-iot networks: a systematic review of issues, challenges and solutions, Cluster Comput. (2021) 1–42.

[11] J.F. Cevallos, A. Rizzardi, S. Sicari, A.C. Porisini, Deep reinforcement learning for intrusion detection in internet of things: Best practices, lessons learnt, and open challenges, Comput. Netw. 3 (3) (2023) 110016.

[12] M. Ahmed, A.N. Mahmood, Novel approach for network traffic pattern analysis using clustering-based collective anomaly detection, Ann. Data Sci. 2 (1) (2015) 111–130.

[13] C. Wang, H. Zhou, Z. Hao, S. Hu, J. Li, X. Zhang, B. Jiang, X. Chen, Network traffic analysis over clustering-based collective anomaly detection, Comput. Netw. 205 (2022) 108760.

[14] Cevallos M. J.F., A. Rizzardi, S. Sicari, A.C. Porisini, Nero: Neural algorithmic reasoning for zero-day attack detection in the iot: A hybrid approach, Comput. Secur. 142 (2024) 103898, [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404824002001.

[15] Y. Ren, J. Pu, Z. Yang, J. Xu, G. Li, X. Pu, S.Y. Philip, L. He, Deep clustering: A comprehensive survey, IEEE Trans. Neural Netw. Learn. Syst. (2024).

[16] C. Lu, X. Wang, A. Yang, Y. Liu, Z. Dong, A few-shot based model-agnostic meta-learning for intrusion detection in security of internet of things, IEEE Internet Things J. (2023).

[17] S. Roy, J. Li, B.-J. Choi, Y. Bai, A lightweight supervised intrusion detection mechanism for iot networks, Future Gener. Comput. Syst. 127 (2022) 276–285.

[18] J. Snell, K. Swersky, R. Zemel, Prototypical networks for few-shot learning, Adv. Neural Inf. Process. Syst. 30 (2017).

[19] M. Meilă, H. Zhang, Manifold learning: what, how, and why, Annu. Rev. Stat. Appl. 11 (2023).

[20] N. Sameera, M. Shashi, Deep transductive transfer learning framework for zero-day attack detection, ICT Express 6 (4) (2020) 361–367.

[21] G. Bovenzi, G. Aceto, D. Ciuonzo, V. Persico, A. Pescapé, A hierarchical hybrid intrusion detection approach in iot scenarios, in: GLOBECOM 2020-2020 IEEE Global Communications Conference, IEEE, 2020, pp. 1–7.

[22] J. Yang, X. Chen, S. Chen, X. Jiang, X. Tan, Conditional variational auto-encoder and extreme value theory aided two-stage learning approach for intelligent fine-grained known/unknown intrusion detection, IEEE Trans. Inf. Forensics Secur. 16 (2021) 3538–3553.

[23] T.T. Thein, Y. Shiraishi, M. Morii, Few-shot learning-based malicious iot traffic detection with prototypical graph neural networks, IEICE Trans. Inf. Syst. 106 (9) (2023) 1480–1489.

[24] S. Garcia, A. Parmisano, M.J. Erquiaga, Iot-23: A labeled dataset with malicious and benign iot network traffic (version 1.0.0), 2020, http://dx.doi.org/10.5281/zenodo.4743746.

[25] Z. Shi, M. Xing, J. Zhang, B.H. Wu, Few-shot network intrusion detection based on model-agnostic meta-learning with l2f method, in: 2023 IEEE Wireless Communications and Networking Conference, WCNC, IEEE, 2023, pp. 1–6.

[26] C. Finn, P. Abbeel, S. Levine, Model-agnostic meta-learning for fast adaptation of deep networks, in: International Conference on Machine Learning, PMLR, 2017, pp. 1126–1135.

[27] H. Sun, L. Wan, M. Liu, B. Wang, Few-shot network intrusion detection based on prototypical capsule network with attention mechanism, PLoS One 18 (4) (2023) e0284632.

[28] M.K. Patrick, A.F. Adekoya, A.A. Mighty, B.Y. Edward, Capsule networks–a survey, J. King Saud Univ.-Comput. Inf. Sci. 34 (1) (2022) 1295–1310.

[29] X.-H. Nguyen, K.-H. Le, Robust detection of unknown dos/ddos attacks in iot networks using a hybrid learning model, Internet Things 23 (2023) 100851.

[30] R. Ahmad, I. Alsmadi, W. Alhamdani, L. Tawalbeh, Zero-day attack detection: a systematic literature review, Artif. Intell. Rev. (2023) 1–79.

[31] N. Mearaj, M.A. Wani, Zero-day attack detection with machine learning and deep learning, in: 2023 10th International Conference on Computing for Sustainable Global Development, INDIACom, IEEE, 2023, pp. 719–725.

[32] Y. Guo, A review of machine learning-based zero-day attack detection: Challenges and future directions, Comput. Commun. 198 (2023) 175–185.

[33] P. Veličković, R. Ying, M. Padovano, R. Hadsell, C. Blundell, Neural execution of graph algorithms, 2019, arXiv preprint arXiv:1910.10593.

[34] J. Cevallos, A. Rizzardi, S. Sicari, A. Coen-Porisini, Zero-Day Attack Detection over High-Dimensional Raw-Iot Traffic Captures, Tech. Rep., 2024, (submitted for publication).

[35] M. Ahmed, A.-S.K. Pathan, Deep learning for collective anomaly detection, Int. J. Comput. Sci. Eng. 21 (1) (2020) 137–145.

[36] Y. Li, F. Gimeno, P. Kohli, O. Vinyals, Strong generalization and efficiency in neural programs, 2020, arXiv preprint arXiv:2007.03629.

[37] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, F. Ahmad, Network intrusion detection system: A systematic study of machine learning and deep learning approaches, Trans. Emerg. Telecommun. Technol. 32 (1) (2021) e4150.

[38] B. Singh, Deepak vand Ng, Y.-C. Lai, Y.-D. Lin, W.K. Seah, Modelling software-defined networking: Software and hardware switches, J. Netw. Comput. Appl. 122 (2018) 24–36.

[39] O.S. Consortium, Openflow switch specification, in OpenFlow switch specification version 1.5.1 (wire protocol 0x06), 2015, [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[40] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: International Conference on Machine Learning, pmlr, 2015, pp. 448–456.

[41] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using rnn encoder–decoder for statistical machine translation, 2014, arXiv preprint arXiv:1406.1078.

[42] D. Li, J. Zhang, Y. Yang, C. Liu, Y.-Z. Song, T.M. Hospedales, Episodic training for domain generalization, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1446–1455.

[43] F. Sung, Y. Yang, L. Zhang, T. Xiang, P.H. Torr, T.M. Hospedales, Learning to compare: Relation network for few-shot learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 1199–1208.

[44] GNS3 Community, GNS3: The software that empowers network professionals, 2024, https://www.gns3.com/.

[45] Docker Contributors, Docker Documentation: An Open Platform for Developing, Shipping, and Running Applications, Docker, Year of the latest version or last update, https://docs.docker.com/.

[46] PoX Community, Pox controller documentation, 2024, GitHub. [Online]. Available: https://noxrepo.github.io/pox-doc/html/.

[47] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, et al., The design and implementation of open {vSwitch}, in: 12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, 2015, pp. 117–130.

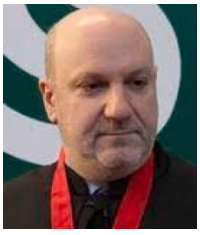
[48] P. Karlsson, TCPReplay: A tool for replay and analysis of tcp traffic, in: Proceedings of the 4th Annual Computer Security Applications Conference, IEEE Computer Society, 1988, pp. 191–197.

[49] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, G. Wayne, Experience replay for continual learning, Adv. Neural Inf. Process. Syst. 32 (2019).
[50] S. Brody, U. Alon, E. Yahav, How attentive are graph attention networks?, 2021, arXiv preprint arXiv:2105.14491.
[51] S. Das, A. Biswas, Analyzing correlation between quality and accuracy of graph clustering, in: Advances in Computers, Elsevier, 2023, pp. 128, 135–163.
[52] P.W. Battaglia, J.B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al., Relational inductive biases, deep learning, and graph networks, 2018, arXiv preprint arXiv:1806.01261.
[53] A. Altabaa, J. Lafferty, Approximation of relation functions and attention mechanisms, 2024, arXiv preprint arXiv:2402.08856.
[54] T.W. Webb, S.M. Frankland, A. Altabaa, S. Segert, K. Krishnamurthy, D. Campbell, J. Russin, T. Giallanza, R. O'Reilly, J. Lafferty others, The relational bottleneck as an inductive bias for efficient abstraction, Trends in Cognitive Sciences (2024).
[55] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, S.Y. Philip, A comprehensive survey on graph neural networks, IEEE Trans. Neural Netw. Learn. Syst. 32 (1) (2020) 4–24.

Jesús F. Cevallos M. received a Ph.D. in Computer Science Engineering from Sapienza University (Rome) in 2022. He now covers a post-doc researcher position at the University of Insubria (Varese). His main research interests are industrial/ethical applications of Deep Learning with a special focus on Natural Language Processing and Neural Algorithmic Reasoning.

Alessandra Rizzardi is Assistant Professor at University of Insubria (Varese), where she received BS/MS degree in Computer Science 110/110 cum laude in 2011 and 2013, respectively. In 2016 she got Ph.D. in Computer Science and Computational Mathematics at the same university. She is the Principal Investigator for the funded national project PRIN 2022, and Editorial Board member for the journals Transactions on Emerging Telecommunications Technologies (Wiley), Internet Technology Letters (Wiley), and Sensors (MDPI). Her research activity is on WSN and IoT security issues.

Sabrina Sicari is a Full Professor at University of Insubria (Varese). She received the M.Sc. degree in Electronic Engineering, 110/110 cum laude, from the University of Catania, in 2002, where in 2006, she got Ph.D. in Computer and Telecommunications Engineering, followed by Prof. Aurelio La Corte. She is a COMNET, IEEE IoT, ETT, and ITL editorial board member. Her research concerns security, privacy, and trust in WSN, WMSN, IoT, and distributed systems. She is an IEEE senior member.



Alberto Coen-Porisini received a Dr. Eng. degree and Ph.D. in Computer Engineering from Politecnico di Milano in 1987 and 1992. He has been a Full Professor of Software Engineering at Università degli Studi dell'Insubria since 2001, Dean of the School of Science from 2006 and Dean from 2012 to 2018. His research regards specification/design of real-time systems, privacy models, and WSN.