



UNIONE EUROPEA
Fondo Sociale Europeo



Ministero dell'Università
e della Ricerca



REACT EU



SCUOLA DI DOTTORATO
UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA

Dipartimento di / Department of
Informatica, Sistemistica e Comunicazione (DISCo)

Dottorato di Ricerca in / PhD program in **Computer Science** Ciclo / Cycle **XXXVII**

Uncertainty Quantification and Distributed Models to Enhance ML-based Network Security

Cognome / Surname **Talpini** Nome / Name **Jacopo**

Matricola / Registration number **848060**

Tutor: prof. **Raffaella Rizzi**

Supervisor: prof. **Fabio Sartori**

Co-Supervisor: dott. **Marco Savi**

Coordinatore / Coordinator: prof. **Leonardo Mariani**

ANNO ACCADEMICO / ACADEMIC YEAR 2023/2024

Abstract

Network intrusion detection stands as a major pillar for ensuring secure network connections. As both networks and network attacks grow in complexity, machine learning models have emerged as promising tools to form the foundation of advanced network intrusion detection systems. This thesis focuses on how the peculiar characteristics of this domain -namely its sensitivity, criticality, and the availability of distributed computational power- should be appropriately considered in the model design of learning algorithms for intrusion detection systems.

The first part of the thesis investigates the problem of developing models whose predictions align with our expectations, especially in scenarios different from the usual assumption that the training and test sets contain independent and identically distributed samples from the same distribution. In particular, we focused on models that are inherently uncertainty-aware, such as those based on Bayesian inference, showing how they can be beneficial to enhance closed-set classification performance, would make it possible to carry out Active Learning, and would help recognize inputs of unknown classes as truly unknowns, unlocking open-set classification capabilities and Out-of-Distribution detection.

The second part proposes federated learning as a framework for training models in a distributed and privacy-preserving way. We introduce an approach to mitigate the negative effects of data heterogeneity among clients, aiming to improve the overall predictive performance of a federated ML-based intrusion detection system.

The third and final part investigates how to train uncertainty-aware models under the federated learning paradigm. We propose a simple approach to calibrate any given pre-trained model under the assumption of the availability of a local calibration set. In addition, we show how Bayesian inference provides a natural approach for designing a federated learning process, especially in one single round, the so-called One-shot federated learning.

Overall, this manuscript aims to shed light on open problems and propose potential solutions in the field of ML-based intrusion detection, with the goal of enhancing the trustworthiness and scalability of machine learning models, particularly in comparison to traditional, centralized approaches.

Contents

List of Acronyms	5
1 Introducing Network Intrusion Detection Systems	6
1.1 Motivation	6
1.2 A Typical ML-based IDS	9
1.3 Challenges and Open Problems in ML-based IDS	11
1.3.1 Trustworthiness	11
1.3.2 Decentralized Training	12
1.3.3 Others Relevant Challenges	13
1.4 Summary of the Main Contributions of the Thesis	14
2 Background on Bayesian Inference and Federated Learning	18
2.1 Introducing Bayesian Inference	18
2.2 Bayesian Neural Network	20
2.2.1 Laplace Approximation	22
2.2.2 Variational Inference	23
2.2.3 Deep Ensembles	24
2.3 Uncertainty Quantification	25
2.3.1 Evaluation of the Uncertainty	26
2.3.2 Examples on a toy dataset	28
2.4 Introducing Federated Learning	30
2.4.1 Problem Formulation	30
2.4.2 A fundamental Result	32
3 Trustworthy ML Models	34
3.1 Trustworthy ML-based IDS and Related Work	34
3.1.1 Related Work	36
3.1.2 Uncertainty Quantification and OoD Detection	39
3.2 Proposed Framework	41
3.3 Problem Statement and Models	41
3.3.1 Problem statement	41
3.3.2 Models	43

3.4	Dataset Description and Preprocessing	49
3.5	Illustrative Numerical Results	51
3.5.1	Models implementation details	51
3.5.2	Experimental Setup and Computational Times	53
3.5.3	Closed-Set Classification	53
3.5.4	Active Learning	57
3.5.5	Out-of-Distribution Detection	61
3.5.6	Discussion	66
3.6	Dynamic Environment	67
3.6.1	Introduction	67
3.6.2	Related Work	68
3.6.3	Problem Formulation and Adopted Models	69
3.6.4	Dataset Description and Preprocessing	73
3.6.5	Illustrative Numerical Results	74
3.6.6	Discussion	76
4	Federated ML-Based Intrusion Detection	77
4.1	FL-based IDS: Related Work	78
4.2	Proposed approach	80
4.2.1	Proposed Architecture	80
4.2.2	Proposed Clustering Strategy	82
4.3	Adopted Classifier and Baseline Approaches	85
4.4	Illustrative numerical results	85
4.5	Discussion	88
5	Towards Trustworthy FL-based Intrusion Detection	90
5.1	On calibrating a FL ML-based IDS	90
5.1.1	Proposed Approach	91
5.1.2	Experimental Setting	93
5.1.3	Illustrative numerical results	94
5.1.4	Discussion	96
5.2	Towards One-Shot FL	98
5.2.1	Introduction and Related Work	98
5.2.2	Problem Setting	99
5.2.3	Proposed Approach	100
5.2.4	Experimental Setting	101
5.2.5	Illustrative Numerical Results	104
5.2.6	Discussion	107
	Bibliography	110

List of Acronyms

AL	Active Learning
BALD	Bayesian Active Learning by Disagreement
BNN	Bayesian Neural Network
CNN	Convolutional Neural Network
DL	Deep Learning
ECE	Expected Calibration Error
ELBO	Evidence Lower Bound
FedAvg	Federated Averaging
FL	Federated Learning
KL	Kullback-Leibler
IDS	Intrusion Detection System
iid	Independent and Identically Distributed
IoT	Internet of Things
MAP	Maximum a Posteriori
MC	Monte Carlo
MCMC	Markov chain Monte Carlo
ML	Machine Learning
MLP	MultiLayer Perceptron
NN	Neural Network
OoD	Out of Distribution
pdf	Probability Density Function
RF	Random Forest
SGD	Stochastic Gradient Descent
SOTA	State of the Art
VI	Variational Inference

Chapter 1

Introducing Network Intrusion Detection Systems

1.1 Motivation

Network intrusion detection is a major component of network security, aimed at protecting computer networks and network-accessible endpoints from malicious activities that compromise confidentiality, integrity, or availability within the network infrastructure. With the constant increase in the number and complexity of occurring incidents [Eur22], it is fundamental to design and implement scrupulous detection strategies and robust counteraction measures to effectively identify and mitigate these threats.

Intrusion Detection Systems (IDSs) are among the primary security measures in communication networks. IDSs are hardware or software systems that analyze network traffic with the aim of identifying attacks, unauthorized intrusions, as well as malicious activities [TLR22a] that traditional in-line devices, such as firewalls, which typically operate only up to the transport layer of the network, may not identify.

The traditional approach for detecting intrusions relies on knowledge-based systems (e.g., expert systems and finite state machine)[HCS⁺19] but as long as networks rise in complexity they become more prone to errors [SNPS18; TJZ⁺21b]. As a consequence, in recent years, data-driven approaches based on *Machine Learning* (ML) have been widely considered for the detection of attacks. In fact, the research field of ML-based IDSs has experienced a remarkable evolution, with an increase of 4 times in new publications incorporating the keywords “machine learning intrusion detection” from 2015 to 2023, as depicted in Figure 1.1.

In this context, it is worth noting that the field of Intrusion Detection presents unique challenges to ML models that differ significantly from those in other domains, such as commercial applications (e.g., recommender systems

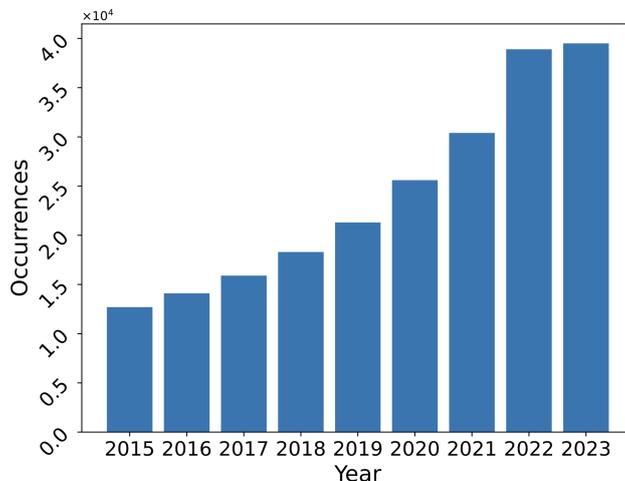


Figure 1.1: Occurrences of the keyword “machine learning intrusion detection” over time in academic papers (from Google Scholar). The results were obtained using the code from <https://github.com/Pold87/academic-keyword-occurrence>

or language translation), where ML models have been highly successful.

Network intrusion detection is a safety-critical domain, where incorrect predictions of an ML model can have severe and costly consequences. A false positive requires spending a considerable amount of analyst time examining the reported incident only to eventually determine that it reflects a benign activity [SP10]. In fact, even a small rate of false positives can quickly make an IDS unusable. In addition, also false negatives can be particularly damaging to an organization, as even a single compromised system can significantly threaten the integrity of the IT infrastructure.

Moreover, an IDS operates in an inherently adversarial and dynamic environment, where attackers and defenders continuously improve their strategies in response to each other’s evolving techniques, with attackers adapting their activities to evade detection [ALMdo⁺23]. Despite that, the vast majority of the proposed methods in the literature is focused on improving predictive performance (e.g., classification accuracy, F1-score) on a test set whose samples are drawn from the same distribution as the training set, the so-called *independent and identically distributed*(iid) assumption [SP10].

However, this setting does not reflect the real-world deployment of an ML-based system, where, in practice, a ML model might be presented with input pertaining to classes never seen at training time [APC22; ALMdo⁺23]. This scenario is especially relevant in intrusion detection [SP10], as networks face increasingly sophisticated attacks, with a rising likelihood of zero-day attacks, particularly in modern network infrastructures [ZCB21].

Given these premises, one of the main contributions of this thesis is to

show how ML models that are inherently uncertainty-aware, i.e., that can provide a meaningful estimation of the uncertainty associated with their predictions, can be useful and play a relevant role in the previously described scenario, providing a foundation for models that perform reliably across different tasks, with the ultimate goal of increasing trust in them.

So far we highlighted how a peculiarity of network intrusion detection domain should be reflected in the design of the model itself. Another orthogonal direction refers to how a networked system can naturally host and deploy an ML-based IDS. As an example, let us consider the emerging paradigm of the Internet of Things (IoT) that facilitates the interconnection of diverse devices and computing capabilities over the Internet or an Edge Computing scenario. In these contexts, data is naturally collected at the edge, making it advantageous to train machine learning models directly at the edge. In fact, the increasing size of data generated by IoT devices represented one of the main motivations for the development of *Federated Learning* (FL) ([MMR⁺17; Kon16]), a framework allowing geographically distributed clients to collaboratively learn machine learning models, without the need to share their own local data.

The key distinguishing property of federated learning is that data remains local, thus avoiding any exchange or transfer of sensitive information bringing forth two compelling potentials, each with its unique set of advantages. Firstly, the emphasis on local data storage increases privacy protection, by keeping data confined to its originating device, federated learning reduces the potential attack surface of the system and hence it minimizes the risk of data breaches and unauthorized access [Eur16]. Secondly, the local data storage approach also leads to notable efficiencies in communication resources. Since data does not need to be replicated in a centralized cloud server, the burden on network bandwidth is substantially reduced.

Another contribution of this thesis is to investigate the feasibility of FL-based IDS, with the ambition of paving the road for the application of the federated learning training paradigm for the development of ML-based IDS. Moreover, we explore the possibility of enhancing the reliability of ML models in a privacy-preserving way. Ultimately, this thesis addresses the challenges of heterogeneity in federated learning and introduces practical algorithms specifically designed for the network domain, intending to enhance the learning process and make FL feasible for intrusion detection.

In the rest of this chapter, we describe a typical ML-based IDS, followed by an overview of the main challenges and open problems for ML-based IDS. Then, we summarize the principal contributions of this manuscript and outline the forthcoming chapters in the last Section.

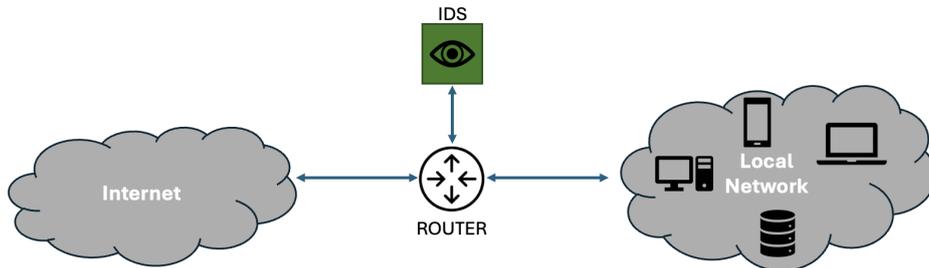


Figure 1.2: Pictorial representation of the deployment of a network intrusion detection system.

1.2 A Typical ML-based IDS

In this section, a high-level overview of ML-based IDS is given, with a focus on methods for the analysis of network flows. We focus on this approach since others, like deep packet inspection or inspecting the complete payload, can be either difficult due to traffic encryption or computationally costly and can become a performance bottleneck in high-speed IP networks [USB17].

A network flow can be defined as a set of IP packets passing through an observation point in the network during a given time interval. All packets belonging to the same flow are characterized through a set of common properties, such as: source and destination IP, port, and IP protocol [SSS⁺10].

Aggregated properties on the network traffic flows can be obtained, e.g. by coarser-grained flow definitions. In particular, in recent years, Netflow [Cla04] has established itself as the main solution for flow-level measurements. It was originally developed by Cisco [Cla04] as a cache to improve IP flow lookups in routers and then it proved itself as an effective tool for network traffic monitoring and reporting, by providing aggregated information on network traffic in the form of flow-records.

Since IDS are used only for detection, they are deployed to collect network traffic by receiving mirror traffic by network devices (switch, router, TAPs, network packet brokers etc.) in the form of Netflow and packet capture. Figure 1.2 shows a typical deployment scheme of an IDS. An edge router mirrors all the network traffic passing through it to an IDS that further analyzes the received data, e.g. by means of an ML model.

Those features are high-level statistics of network flows, such as longest flow packet, shortest flow packet, flow duration, etc, and they are commonly used for the development of data-driven models for the analysis of network traffic [SSS⁺10]. It is worth mentioning that some of these features are in fact dropped for the training of ML-based IDS, in particular those that would be detrimental to the generalization of the model, such as IP addresses and TCP/UDP ports (specific to the end-hosts and user applications), link layer encapsulation type (linked to the network interfaces) and application-layer

attributes (e.g., IRC or HTTP protocol attributes).

Starting from these kinds of data, i.e. network flow features, many solutions have been proposed for network traffic classification, spanning from simple algorithms to Deep Learning models, whose peculiarities give rise to two possible families of IDS, often referred to as *signature-based* or *anomaly-based* [TJZ⁺21b].

- The first category, also known as *misuse-based IDSs*, is based on pattern recognition, with the goal of comparing signatures of well-known attacks and benign traffic to the current network traffic patterns. An example of this approach is represented by the use of pattern-matching algorithms in packet payload analysis, which is adopted by Snort [Roe99]. In this context, supervised ML methods are promising tools to analyze network traffic and classify it as benign or as a particular intrusion [SP10]. The major limitation of this approach is that these IDSs can only recognize known types of intrusions and will fail to identify new kinds of attacks, commonly referred to as zero-day attacks.
- On the other hand, anomaly-based methods rely on a model for the normal (i.e., benign) network traffic so that any pattern that deviates from the usual one is considered an intrusion. In contrast to signature-based IDSs, anomaly-based IDSs are able to detect also new types of attacks since they are based on what is considered to be benign traffic. However, their major limitation is that they typically suffer from a high rate of false positives.[LLLT13].

This thesis is particularly focused on signature-based IDS, whose starting point is the availability of a labelled dataset. This requires network flows to be collected and also labelled by domain experts, to create a dataset of input-output pairs that can be exploited for the training of data-driven models, typically a classifier. A rigorous formulation of this problem will be given in the next section.

In recent years, data-driven approaches for developing signature-based IDSs have been extensively explored (e.g. [SNPS18; TJZ⁺21b; HCS⁺19; BBCC19; VAS⁺19a]) considering different methods such as Random Forests, Support Vector Machines, Neural Networks or Clustering techniques. Various Machine Learning and especially Deep Learning (DL) models have emerged as promising data-driven methods with the capability to learn and extract meaningful patterns from network traffic, which can be beneficial for detecting security threats occurring in networked systems [AMT⁺20; VAS⁺19a]. For instance, [VR20; VAS⁺19b; AYMS21] compare different classification algorithms for developing an IDS and, in general, the best performance is achieved by tree-based classifiers, like Random Forests, and Multi-Layer Perceptrons (MLPs). In addition, [LL19] and exploited DL models for classifying

traffic data collected from the real network environment, experiments show that the MLP model outperforms Convolutional Neural Networks (CNNs) in terms of accuracy, training, and classification times.

1.3 Challenges and Open Problems in ML-based IDS

As mentioned in the previous Section, the standard setting of a signature ML-based IDS can be formalized as a supervised classification problem: given a dataset \mathcal{D} of N input-output pairs $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$, the aim is to define a parametric function (e.g. a Neural Network) that provides the conditional probability distribution $p(\mathbf{y}|\mathbf{x}, \mathbf{w})$ over K classes, for a given input \mathbf{x} and model parameters \mathbf{w} [Mur23]. Then, the predicted class is typically chosen to be the class with the highest probability.

Given the aforementioned setting, the thesis analyzes two open problems regarding the definition of an ML-based IDS, briefly introduced in Section 1.1 :

- The first concerns the trustworthiness of the model itself, by ensuring its predictions align with our expectations, by going beyond the usual focus on the analysis of predictive performance with test inputs close to the data on which the model was trained.
- The second one deals with a distributed scenario, where the dataset \mathcal{D} is split over geographically different clients, giving rise to a decentralized training setting. As we will discuss, this setting is non-trivial in the context of network intrusion detection due to a natural heterogeneity between different clients data.

1.3.1 Trustworthiness

The crucial point regarding the trustworthiness of an ML-signature-based IDS lies in the standard approach to supervised ML assumes the training and test sets both contain independent and identically distributed samples from the same distribution. This assumption often made implicitly, can be a severe limitation for the application of ML models in real-world settings, in which the test distribution may be different from the training distribution. This scenario is particularly compelling in the context of intrusion detection [SP10], since new kinds of attacks, or variations of known attacks emerge over time, and the likelihood of being targeted by zero-day attacks is getting higher and higher, especially in modern network infrastructures [ZCB21]. As a consequence, in practice, a ML model exploited as a basis for an IDS might be presented with input pertaining to classes never seen at training time [APC22]. Moreover,

there is a huge ML literature (e.g. [NYC15][BB16][GPSW17]) showing that ML models, including Deep Learning ones, tend to produce, overconfident, arbitrarily high per-class classification scores not only to misclassified samples from known classes but also to Out-of-Distribution (OoD) instances, which may be related to unknown classes. This somehow unexpected behavior represents a severe issue for the deployment of a *trustworthy* ML-based IDS since we might expect that an IDS will have to face new kinds of attacks or variations of known attacks [SP10]. In fact, it would be desirable to rely on a model that allows network operators or companies offering an IDS service to assess more accurately the uncertainty associated with the chosen model’s predictions, thus raising their awareness and allowing them to perform more informed risk evaluations while taking the corresponding most appropriate countermeasures accordingly. In this context, an uncertainty-aware model can make even low-accuracy prediction useful, by calling the attention of human experts towards those cases where further analysis is necessary.

Here, we refer to the term trustworthiness, or reliability, as the ability of a model to work consistently across different real-world settings, borrowing the terms from reliability engineering [BP75; OK11], in line with the concepts presented in [AOS⁺16; ABB⁺21; TLD⁺22; PSP⁺24].

In the context of intrusion detection, we argue for the following two main desiderata for reliable AI systems: they should represent properly their own uncertainty, and they should be able to efficiently adapt to new data. Moreover, we require that the employed model should perform well in all of these settings simultaneously out-of-the-box, without requiring any customization for each individual task. Then, if a model performs reasonably well across a variety of regimes and tasks, that might increase one’s trust in it, and it may be less relevant to understand how it produced its outputs (i.e. its interpretability).

1.3.2 Decentralized Training

The starting point for a signature-based IDS is the availability of sufficiently large and representative training data. Those data are typically collected from edge nodes and sent to a central server that it is responsible for the training of a ML model. However, as in IoT environments, nodes that collect traffic have also some computational powers, therefore it would be interesting and useful to demand the training of a given ML model directly to these distributed nodes. In fact, a limitation of current ML-based approaches is that model training is based on data and computational power elaborated and owned by a centralized node (e.g., a server). Centralized ML approaches are thus generally associated with different challenges including the need for high computational power and long training time, as well as with the rise of

security and privacy concerning users’ data [MPP⁺20].

In order to address these issues, federated learning was originally proposed in [MMR⁺17] and has recently emerged as an effective model training paradigm to address the issues recalled above. It embodies the principles of *focused collection* and *data minimization*, and can especially mitigate many of the systemic privacy risks and costs resulting from traditional, centralized machine learning, including high communication efficiency and low-latency data processing [KMA⁺21]. Despite IDS seeming a natural scenario where to apply the FL training paradigm, devices typically collect data samples that are not independent and identically distributed [CSGV⁺21]. This scenario, known as *statistical heterogeneity* of the collected clients’ (e.g., IoT devices’) datasets, poses a challenge for a federated learning setting [LSTS20] and is particularly detrimental when developing and deploying an FL-based IDS, since some clients may have traffic associated with several kinds of attacks (e.g. DoS, port scanning, etc.), while other could only have traffic related to their intended operation [CSGV⁺21]. Heterogeneity often reduces the performance of federated learning, it slows convergence down and it may lead to unfair models, making the exploitation of the FL approach non-trivial for training an ML-based IDS [Mar23]. Moreover, the impact of the FL training paradigm on models’ trustworthiness is still uninvestigated.

1.3.3 Others Relevant Challenges

Trustworthiness and decentralized training are the main research directions explored in this thesis; however, it is also worth briefly describing other challenges regarding the methodological aspects of an ML-based IDS, some of which are closely related to these two areas.

- **Adversarial attacks in the Cybersecurity domain.** As we mentioned, intrusion detection is a dynamic environment where new kinds of attacks emerge gradually over time. However, it might happen that an adversary could intentionally select inputs designed to degrade a predictive model’s performance. For example, given an input \mathbf{x} classified as belonging to class c , the adversary may craft a new input \mathbf{x}_{adv} that minimizes the probability of this classification while ensuring that \mathbf{x}_{adv} remains similar to the original input \mathbf{x} [Mur23]. A general discussion on the problem of adversarial attacks in the security domain has been emphasized by a few papers, e.g. [ZLY⁺22; RSER21; MWF24]. This issue closely relates to the previously discussed challenges of OoD detection and distribution shift. However, we have not specifically evaluated the performance of uncertainty-aware models under adversarial attack scenarios, though the same concepts and methodology are expected

to perform similarly [SG18a; CWL⁺20] and we leave this investigation for future works.

- **Interpretability of ML models.** Flexible models like neural networks are usually referred to as ‘black boxes’: although they reach great predictive performance, we do not know why and how they accomplish a given task. On the other hand, interpretable models allow human experts (e.g., security personnel) to inspect them and to provide possible reasoning behind the ML models’ predictions [SMS⁺22; CFLS22]. In this thesis, we focus on a complementary approach: developing a model that performs reliably across various tasks relevant to IDS, even if we may not fully understand its internal workings. It is important to note that the reverse is not necessarily true, a white-box model does not always perform as desired across different relevant tasks, [Mur23; AGM⁺18], these explanations can also be harmful, potentially leading users to place excessive trust in them in unintended ways [KNJ⁺20].
- **Learning from few samples.** ML models are usually “data hungry”, however, this is particularly a problem when a large labeled training set is not available. In the context of IDSs, this can be the case for new kinds of attacks, where only a few examples are available at training time. As a consequence, the concept of few-shot learning - learning and generalizing well starting from just a few examples- is a relevant research direction. In fact, there are many approaches in the ML literature, e.g. [WWHX20], but their application in network IDS is still in its infancy [HTA⁺23; LWY⁺23].

These are only a few of the open problems in the realm of network intrusion detection, mostly focused on the model design principles, given the peculiarities of the domain. For a more general overview of the research directions in the realm of network security, the following reference can be consulted: [TL23; SGJM23; AATS23; HJJ23; ALMdO⁺23]

1.4 Summary of the Main Contributions of the Thesis

This thesis investigates two main problems related to the development and deployment of an ML-based IDS in a realistic scenario: the problem of the reliability of ML models and the opportunity and challenges of a federated training of an IDS.

In Chapter 2, we offer on giving a theoretical overview on the Bayesian learning theory and the problem of model uncertainty quantification. In

particular, we emphasize that Bayesian inference can be seen as a means for developing reliable models with a particular focus on the current state-of-the-art techniques to make Bayesian inference scalable to complex models like deep neural networks. Then, we review some basic results on federated learning.

In Chapter 3, we analyze the trustworthiness of an ML-based IDS in a centralized setting. In particular, we highlight that an ML-based IDSs should always provide accurate uncertainty quantification to avoid overconfident predictions, thus increasing one trust in it. In fact, an uncertainty-aware classification would be beneficial to enhance closed-set classification performance, would make it possible to carry out Active Learning, and would help recognize inputs of unknown classes as truly unknowns, unlocking open-set classification capabilities and Out-of-Distribution detection, thus reducing the gap between Signature and anomaly-based IDSs. Moreover, we propose a method to tackle the need to keep the IDS capability constantly updated, which poses peculiar challenges, especially in resourced-constrained scenarios. To this end, we propose a novel hierarchical model based on a binary classification of benign and malicious traffic performed by a Bayesian Neural Network that is trained continuously and efficiently by exploiting continual learning.

In Chapter 4, we offer a method based on federated learning to train a ML-based IDS in a distributed way with the overall aim of increasing performance by ensuring data privacy and efficiency. We propose an approach to tackle the problem of data unbalance across different clients using a custom clustering strategy.

Last, in Chapter 5 we explore how federated learning affects the calibration of a base model and how to calibrate it in a privacy-preserving way. We then explore the application of Bayesian inference in a distributed manner, demonstrating how it enables one-shot federated learning for model training.

Figure 1.3 shows a pictorial representation of the contributions of this thesis in the realm of a network where there is the need to develop an IDS. The Green arrow indicates that the problem of model trustworthiness is a holistic problem that requires careful choice about the design of the ML model itself. In this context, the main contributions are:

- 1 **J. Talpini**, F. Sartori, M. Savi, *Enhancing Trustworthiness in ML-Based Network Intrusion Detection with Uncertainty Quantification*. Journal of Reliable Intelligent Environments, Aug. 2024
- 2 **J. Talpini**, F. Sartori, M. Savi, *Hierarchical Multiclass Continual Learning for Network Intrusion Detection*, in IEEE Conference on Network Softwarization (NetSoft), Jun. 2024

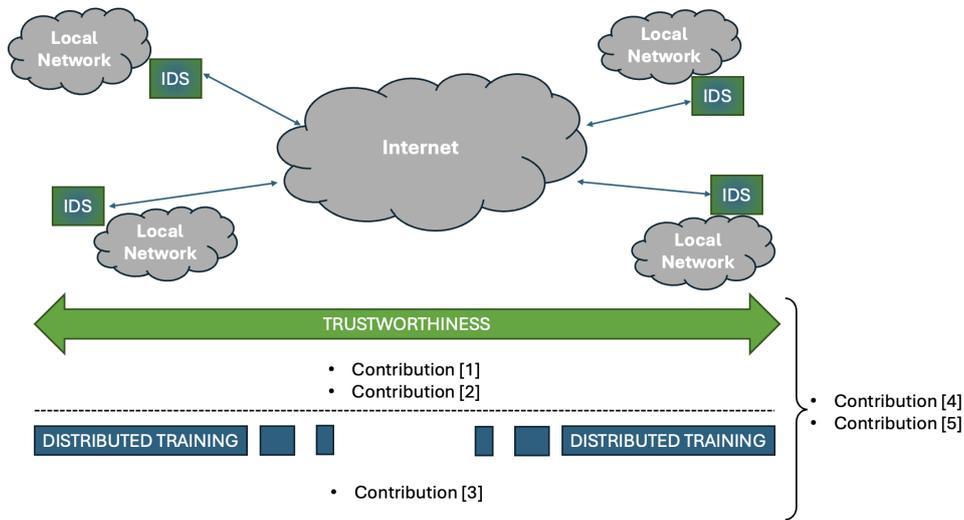


Figure 1.3: Pictorial representation of the overall contribution of the thesis. The trustworthiness problems deal with every region of the network, while federated learning allows local training of an IDS. These will be presented as independent contributions.

The blue band indicates the problem of training an ML-based IDS in a federated way. Within this research direction, the main contribution is:

- 3 **J. Talpini**, F. Sartori, M. Savi, *A Clustering Strategy for Enhanced FL-Based Intrusion Detection in IoT Networks*, in International Conference on Agents and Artificial Intelligence (ICAART), Feb. 2023

After addressing the challenges of trustworthiness and reliability individually, we present our contributions, which aim to explore both fields simultaneously:

- 4 **J. Talpini**, M. di Gennaro, M. Carminati, M. Savi, Francesco Malandrino *SHIELDED: Exploiting Programmable Networks for Secure and Trustworthy Distributed Learning*, submitted to a magazine
- 5 **J. Talpini**, N. Civiero, F. Sartori, M. Savi, *A Federated Approach to Enhance Calibration of Distributed ML-based Intrusion Detection Systems*, submitted to a conference

In addition to the core contributions outlined earlier, which represent the main focus of this manuscript, we also report the following publications that fall outside the direct scope of this thesis:

- 6 F. Sartori, M. Savi, G. Tarrini, **J. Talpini**, *Towards a Knowledge Diversity Notion to Identify Intrusions in Industrial Contexts*, International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Jun. 2024

- 7 N. Di Cicco, **J. Talpini**, M. Ibrahimi, M. Savi, M. Tornatore, *Uncertainty-Aware QoT Forecasting in Optical Networks with Bayesian Recurrent Neural Networks*, in IEEE International Conference on Communications (ICC), May 2023
- 8 F. Sartori, M. Savi, K. Shala, A. Moglia and J. Talpini, *Knowledge Artifacts to Support Dietary: the Diet Module of the PERCIVAL Project*, in International Conference on Metadata and Semantics Research (MTSR), Nov. 2022
- 9 F. Sartori, M. Savi, **J. Talpini**, *Tailoring mHealth Apps on Users to Support Behavior Change Interventions: Conceptual and Computational Considerations*, in Applied Sciences, vol. 12, no. 8, pp. 3782, Apr. 2022

Chapter 2

Background on Bayesian Inference and Federated Learning

The first part of this Chapter is mostly devoted to the basics of Bayesian probability theory and inference, which provide a unifying and principled framework for data modeling. Then the basics of Federated Learning are introduced.

2.1 Introducing Bayesian Inference

Bayesian inference is a framework that represents “degrees of certainty” using probability theory, and which leverages Bayes’ theorem, to update our degree of belief given some observed data [Jay03; Mac95]. In contrast to the classical frequentist approach, probability in the Bayesian framework represents a degree of subjective belief rather than the relative frequencies of events. As a consequence, probability statements can be made about things other than data, including model parameters and models themselves [Mac95; Mur23].

Bayes’ theorem is the core of Bayesian inference, which provides a way of computing the probability distribution over possible values of an unknown (or hidden) quantity given some observed random variables. It can be derived from the two basic rules of probability theories ¹ [Jay03].

The *product rule* allows to express the joint probability density function (pdf) for two random variables $p(x, y)$, as [Bis]:

$$p(x, y) = p(x|y)p(y) = p(y|x)p(x) \quad (2.1)$$

¹the following -informal- derivation is for continuous random variables, the same principles also hold for discrete random variables

Moreover, the *marginal* probability function for x can be obtained, via the *sum rule*, by integrating over y [Bis]:

$$p(x) = \int p(x, y)p(y)dy \quad (2.2)$$

Starting from the previous two equations, it is trivial to derive Bayes' rule:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} \quad (2.3)$$

In the context of Bayesian inference, the previous rules all applied at all levels of the inference [Mac95]. In particular, given a dataset \mathcal{D} , a model M that depends on some parameters θ and some background assumption I , Bayes' theorem can be written as:

$$p(M, \theta | \mathcal{D}, I) = \frac{p(\mathcal{D} | M, \theta, I)p(M, \theta | I)}{p(\mathcal{D} | I)} \quad (2.4)$$

The term on the left-hand side, $p(M, \theta | \mathcal{D}, I)$, is the *posterior* pdf for the model M and parameters θ , given the data \mathcal{D} and background assumptions I . The quantity $p(\mathcal{D} | M, \theta, I)$ is the *likelihood* of the data, given the model, and a fixed setting of the weights θ . The term $p(M, \theta | I)$ is the *prior* pdf of the joint between the model and its parameters in the absence of any data. The last term, $p(\mathcal{D} | I)$, is called marginalized likelihood or *evidence*, it represents the probability for the data, and it acts as a proper normalization factor for the posterior pdf. At a high level, the steps for the Bayesian inference are:

- Formulation of a model for the data, and hence of the likelihood
- Choice of the prior, which encapsulates all the other knowledge that might exist, beyond those explicitly involved in defining the likelihood. A common choice, especially in the context of neural networks, is a Gaussian prior [WI20]. This choice can be justified through the maximum entropy principle [Jay03]. If we only assume that the pdf for the parameters should have finite variance, then the Gaussian is the least informed choice, in the sense that it maximizes the entropy among all pdf having finite variance. Moreover, this choice has a regularizing effect that reflects the preference for smooth functions [RW06].
- Computation of the posterior pdf through Bayes' theorem.
- Depending on the considered problem at hand, one can look for the maximum of the posterior w.r.t the parameter (e.g., Maximum a Posteriori -MAP- inference) or the mean of the posterior, to summarize the information provided by the whole posterior.

2.2 Bayesian Neural Network

This paragraph introduces the Bayesian Neural Networks, by first viewing the usual training of non-Bayesian NN as a probabilistic inference problem and then by generalizing this approach to a fully-Bayesian approach.

Let us consider a supervised learning problem, such as classification, in which a set \mathcal{D} of N input-output pairs $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ is given, and the aim is to define a parametric function (e.g., a Neural Network) that provides the conditional probability distribution $p(\mathbf{y}|\mathbf{x}, \mathbf{w})$ over K classes, for a given input \mathbf{x} and model parameters \mathbf{w} [Mur23].

A neural network can be considered as an implementation of a non-linear function $f(\mathbf{x}, \mathbf{w})$ that approximates the mapping of input-output pairs and it is trained to perform the desired task. For simplicity, we will consider the case of a binary classifier. In this scenario, the output of the network should be in the range of $[0,1]$ and it can be conventionally defined as the probability that the given input belongs to the first class, i.e. $p(y = 1|\mathbf{x}) = f(\mathbf{x}, \mathbf{w})$. This is usually done by employing the sigmoid function as an activation function to guarantee a valid probabilistic interpretation of the output.

In a non-Bayesian approach, a loss function $\mathcal{L}(\mathbf{w})$ is defined and then minimized to find an ‘optimal’ setting of the parameters, which are then used for making predictions. In fact, many common loss functions can be viewed as negative log-likelihood, by considering the data as a fixed quantity and then varying the parameters to find the minimum of such function. In particular, for the considered probabilistic classification model, the log-likelihood can be derived as follows.

$$\mathcal{L}(\mathbf{w}) = \log(p(\mathcal{D}|\mathbf{w})) = \log \prod_{n=1}^N \text{Ber}(y_n | f(\mathbf{x}_n, \mathbf{w})) \quad (2.5)$$

$$= \sum_{n=1}^N \log[f(\mathbf{x}_n, \mathbf{w})^{y_n} \times (1 - f(\mathbf{x}_n, \mathbf{w}))^{1-y_n}] \quad (2.6)$$

$$= \sum_{n=1}^N [y_n f(\mathbf{x}_n, \mathbf{w}) \times (1 - y_n)(1 - f(\mathbf{x}_n, \mathbf{w}))] \quad (2.7)$$

where Ber refers to Bernoulli distribution. The derived expression is also known as the cross-entropy loss function in the ML literature. It is also common to regularize the loss function, to reduce the overfitting. Also, this modification to the loss can be derived through the lens of a probabilistic approach, by working with the log-posterior instead of the log-likelihood.

The posterior over the parameters $p(\mathbf{w}|\mathcal{D})$ can be obtained via Bayes’

theorem (Equation (2.4)), in particular, the log-posterior can be written as:

$$\log(p(\mathbf{w}|\mathcal{D})) = \log(p(\mathcal{D}|\mathbf{w})) + \log(p(\mathbf{w})) - \log(p(\mathcal{D})) \quad (2.8)$$

The log-posterior is decomposed as a linear combination of three terms. The first one is the usual log-likelihood, the second is the log-prior and the last one does not depend on the parameters of the models. If one is interested in picking only the maximum of the posterior (i.e. the MAP estimate of the parameters \mathbf{w}) then minimizing Equation (2.8) is equivalent to minimizing a regularized version of the previously introduced loss function. If one employs a Gaussian distribution as prior, this leads to the usual L2 regularizer, while the choice of a Laplace distribution to the L1 [Mur23].

To sum up, we have shown that common loss function (e.g., the regularized cross-entropy) can be derived through the lens of probabilistic inference so that the training of a NN can be seen as finding the maximum of the posterior pdf of the parameters.

However, as we emphasized at the beginning of this Chapter, a full Bayesian approach aims at inferring the entire probability distribution for the parameters, rather than just its maximum. Bayesian Neural Networks exploit this approach by using the posterior to make predictions for unseen data.

In fact, the most distinguishing property of a BNN is *marginalization*, i.e., rather than using a single set of the weights $\hat{\mathbf{w}}$, determined at the end of the training phase, BNNs rely on the computation of the predictive distribution for a given input \mathbf{x} , as follows [WI20]:

$$p(\mathbf{y}|\mathbf{x}, \mathcal{D}) = \int p(\mathbf{y}|\mathbf{x}, \mathbf{w})p(\mathbf{w}|\mathcal{D})d\mathbf{w} \quad (2.9)$$

Equation (2.9) can also be viewed as a *Bayesian Model Averaging*, where we have an ensemble of models with different parameters settings, and the overall predictions are achieved by an average over the models' ensemble, weighted by their posterior probabilities [WI20]. Again, we emphasize that Equation (2.9) makes sense only in a Bayesian setting, where the posterior $p(\mathbf{w}|\mathcal{D})$ encodes our uncertainty about the model parameters, given the available data, and it allows us to treat the parameters as nuisance parameters and to get rid of them for making predictions by marginalizing them away.

The main drawback of the described approach is its computational cost, especially for complex models like deep neural networks since the posterior is typically highly multi-modal and not available in closed form [Mur23; WI20]. To tackle this problem, there are a large number of different approximate inference approaches that have been applied to BNN, with different strengths and limitations, in order to approximate Equation (2.9) with a finite number

of samples, say N , from the posterior:

$$p(\mathbf{y}|\mathbf{x}, \mathcal{D}) \equiv \mathbb{E}_{\mathbf{w} \sim p(\mathbf{w}|\mathcal{D})} [p(\mathbf{y}|\mathbf{x}, \mathbf{w})] \approx \frac{1}{N} \sum_{i=1}^N p(\mathbf{y}|\mathbf{x}, \mathbf{w}_i) \quad (2.10)$$

where \mathbf{w}_i are samples drawn from the posterior distribution.

In the following, we provide a brief overview of some of these methods, followed by a more detailed discussion of those specifically employed in the rest of the thesis.

- **Markov Chain Monte Carlo (MCMC)**: are considered as the gold standard method for Bayesian inference, since they do not make strong assumptions about the form of the posterior. However, they present scalability issues, especially for complex models. In fact, despite classical MCMC approaches asymptotically recover the true posterior, they are slow to converge and they are considered prohibitively expensive for BNNs. Moreover, they require access to the full training set at each step, in contrast to traditional training based on stochastic gradient descent that exploits mini-batches of data, ensuring scalability.
- **Laplace Approximation** approximates the posterior with a Gaussian distribution constructed at the maximum of the posterior
- **Variational Inference** approximates the posterior by optimizing a lower bound on the evidence with respect to a variational distribution
- **Deep Ensembles** approximate the posterior with an equally weighted mixture of delta functions after the training of multiple models.

2.2.1 Laplace Approximation

Laplace approximation was first proposed for BNN by [Mac92], it is based on a Gaussian approximation to the posterior, $p(\mathbf{w}|\mathcal{D})$, centered at the MAP estimate, $\hat{\mathbf{w}}$. In fact, the log-posterior can be Taylor expanded around the maximum up to second order contribution as:

$$\log(p(\mathbf{w}|\mathcal{D})) \approx \log(p(\hat{\mathbf{w}}|\mathcal{D})) + \frac{1}{2}(\mathbf{w} - \hat{\mathbf{w}})^T H(\mathbf{w} - \hat{\mathbf{w}}) \quad (2.11)$$

where H is the Hessian matrix of the log-posterior, evaluated at the maximum. Therefore, the log-posterior is approximated (locally) as a quadratic function or, equivalently, the posterior is approximated as a Gaussian whose precision matrix Λ can be identified as the negative Hessian matrix at $\hat{\mathbf{w}}$:

$$p(\mathbf{w}|\mathcal{D}) \approx \mathcal{N}(\mathbf{w}|\hat{\mathbf{w}}, \Sigma = \Lambda^{-1}), \quad \Lambda_{ij} = - \left. \frac{\partial^2 \log p(\mathbf{w}|\mathcal{D})}{\partial w_i \partial w_j} \right|_{\mathbf{w}=\hat{\mathbf{w}}} \quad (2.12)$$

It should be noted that the Hessian matrix may not be positive definite, since the log-likelihood of NN is highly non-convex. Therefore, it is common to use a Gauss-Newton approximation to the Hessian, which is guaranteed to be positive definite [Mur23].

The advantages of this approach are that it is simple, and it can be used to derive a Bayesian estimate from a pre-trained model, which makes this approach particularly appealing for deep models [DKI⁺21]. The main disadvantage is that it is a uni-modal approximation of the posterior and that the curvature information is only used after the model has been estimated, and not during the model optimization process. By contrast, variational inference can provide more accurate approximations for comparable cost [Mur23].

2.2.2 Variational Inference

Variational inference aims at approximating the posterior with a tractable distribution, i.e., easy to sample, $q(\mathbf{w}|\boldsymbol{\theta})$ that depends on some adjustable parameters $\boldsymbol{\theta}$. Those parameters are tuned so that q approximates well p , and the objective function for establishing the goodness of the approximation is the Kullback-Leibler (KL) divergence [Mac03], defined as:

$$\text{KL} [q(\mathbf{w}|\boldsymbol{\theta})||p(\mathbf{w}|\mathcal{D})] \equiv \int q(\mathbf{w}|\boldsymbol{\theta}) \log \frac{q(\mathbf{w}|\boldsymbol{\theta})}{p(\mathbf{w}|\mathcal{D})} d\mathbf{w} \quad (2.13)$$

It is possible to rewrite the previous expression through Bayes' theorem:

$$\int q(\mathbf{w}|\boldsymbol{\theta}) \log p(\mathcal{D}|\mathbf{w}) d\mathbf{w} - \text{KL} [q(\mathbf{w}|\boldsymbol{\theta})||p(\mathbf{w})] = \log(p(\mathcal{D})) - \text{KL} [q(\mathbf{w}|\boldsymbol{\theta})||p(\mathbf{w}|\mathcal{D})] \quad (2.14)$$

The last term is always positive and the log-evidence is a constant, given the data. Therefore, the left-hand side is a lower bound on the model evidence, this term will be our objective function to minimize w.r.t. $\boldsymbol{\theta}$ to find the best approximation, and it is known as variational free energy or ELBO (Evidence Lower Bound) $\mathcal{F}(\boldsymbol{\theta})$:

$$\mathcal{F}(\boldsymbol{\theta}) \equiv \text{KL} [q(\mathbf{w}|\boldsymbol{\theta})||p(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})} [\log(p(\mathcal{D}|\mathbf{w}))] \quad (2.15)$$

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta}} \mathcal{F}(\boldsymbol{\theta}) \quad (2.16)$$

The ELBO presents some relevant properties. Firstly it is composed of two terms, the first terms act as a regularizer since it forces the approximant q to be close to the prior $p(\mathbf{w})$ while the second one is the expected log-likelihood of the model w.r.t the approximant q . Moreover, in the ELBO there is not an explicit dependence on the posterior $p(\mathbf{w}|\mathcal{D})$ which makes it feasible to compute.

A common choice for q is a diagonal Gaussian, an approach also known as mean-field approximation. It is worth mentioning that even if the posterior is approximated as a Gaussian, the parameters that minimize the KL objective are often different from what would be found with the Laplace approximation.

In the context of neural networks, the objective function given by the KL divergence can be minimized using standard optimizers typically employed for training NNs (e.g., those based on stochastic gradient descent), making the approximation scalable to complex models. This approach was firstly proposed for BNN in [BCKW15a], where the reparameterization trick is exploited to compute a low variance estimation of the ELBO, addressing the challenge of obtaining an unbiased estimation of $\nabla_{\theta} \mathbb{E}_{q(\mathbf{w}|\theta)}[\log(p(\mathcal{D}|\mathbf{w}))]$; this method is called Bayes by backprop [BCKW15a].

Another particular form of (degenerate) variational inference is Monte-Carlo Dropout, firstly proposed by [GG16; Mur23]. It is based on the usual stochastic dropout layers [SHK⁺14] added as a form of regularization to neural network architectures, and are ‘turned off’ at test time. In Monte Carlo Dropout, the key idea is to perform random sampling during test time as well. Specifically, we randomly drop out each hidden unit by sampling from a Bernoulli distribution. This process is repeated N times, generating N distinct models. The final prediction is obtained by averaging the predictive distributions of these models with equal weighting.

2.2.3 Deep Ensembles

The previous two approaches are based on a unimodal approximation of the posterior. However, modern deep neural networks have highly multi-modal posteriors, with parameters in different modes giving rise to very different functions [WI20]. As a consequence, a local approximation to compute the posterior predictive may underestimate uncertainty and generalize more poorly. A simple alternative method is to train multiple models with different random initialization, and then to approximate the posterior using an equally weighted mixture of delta functions [LPB17; WI20]:

$$p(\mathbf{w}|\mathcal{D}) \approx \frac{1}{N} \sum_{i=1}^N \delta(\mathbf{w} - \hat{\mathbf{w}}_i) \quad (2.17)$$

where: N is the number of models in the ensemble, $\hat{\mathbf{w}}_i$ is the MAP of the i -th model and δ is the Dirac delta distribution.

It is worth mentioning that deep ensembles differ from classical ensemble methods (e.g., bagging and random forests), which obtain a diversity of their predictors by training them on different subsets of the data (e.g., obtained through bootstrap resampling), or on different features. In the case of convex models, such as linear models or shallow decision trees, data perturbation

is necessary to introduce diversity among base learners. However, in the deep ensemble approach, all models are trained on the same dataset with the same input features. Diversity in this case arises from using different initial parameters, random seeds, and the inherent noise in the stochastic gradient descent (SGD) minimizer, which leads to different solutions due to the non-convex nature of the loss function.

2.3 Uncertainty Quantification

The previous sections introduced Bayesian inference and how to apply it to neural networks, as a principled way to perform inference and learn from data. In this Section we emphasize another benefit of the Bayesian approach: uncertainty quantification about models and their predictions. In particular, the concepts of epistemic and aleatoric uncertainty, as two forms of uncertainty in the context of supervised learning, will be introduced. One of the main motivations for the characterization of these two quantities derives from the development of trustworthy ML models: for decision-making, knowing where a method is uncertain because data are limited (high epistemic uncertainty), and knowing where a method is uncertain because of randomness in the data (high aleatoric uncertainty) will be extremely beneficial.

Bayesian inference offers a principled framework for handling uncertainty: the posterior distribution over the weights in Equation (2.9) allows for capturing the model uncertainty, arising from the uncertainty associated with the parameters of the model, given the training dataset.

In addition to properly quantifying the uncertainty associated with each prediction, BNNs can also offer a principled decomposition of aleatoric and epistemic uncertainty. For classification problems², it is possible to estimate the total predictive uncertainty through the *Shannon Entropy* \mathbb{H} of the predictive distribution [Mur23], which is maximized in case of a flat distribution over the classes (i.e., the most uncertain scenario). Moreover, the uncertainty of the predictive distribution can be further decomposed [DHLDVU18], as follows:

$$\underbrace{\mathbb{H}[p(\mathbf{y}|\mathbf{x}, \mathcal{D})]}_{\text{Total Uncertainty}} = \underbrace{\mathbb{I}[\mathbf{y}, \mathbf{w} | \mathbf{x}, \mathcal{D}]}_{\text{Model Uncertainty}} + \underbrace{\mathbb{E}_{p(\mathbf{w}|\mathcal{D})} [\mathbb{H}[p(\mathbf{y}|\mathbf{x}, \mathbf{w})]]}_{\text{Aleatoric Uncertainty}} \quad (2.18)$$

where \mathbb{I} is the *information gain* between parameters and output, and captures the model uncertainty, while the second component is the aleatoric uncertainty, computed as the expected value \mathbb{E} of the entropy of the predictions

²A similar decomposition also holds for regression problems.

obtained by exploiting the models of the ensemble. It is crucial to decouple these two quantities since they behave differently [DHLDVU18]: the former is typically high for previously unseen inputs, while the latter is high for ambiguous or noisy samples and it does not decrease by acquiring more training data. In other words, the latter cannot be exploited to enhance the quality of a model to recognize unseen inputs. Sometimes, aleatoric and epistemic uncertainties are referred to as irreducible and reducible uncertainties respectively, since epistemic uncertainty can be reduced with more data, while aleatoric uncertainty cannot (e.g., the inherent stochasticity of a dice roll cannot be reduced by observing more rolls). Here we avoid such characterization since in principle also aleatoric uncertainty can be decreased, e.g., by changing the underlying system with which we collect the data and by increasing the measurement precision.

The quantity appearing in Equation (2.18) is typically estimated through Monte Carlo (MC) sampling, similar to the predictive distribution. For instance, for estimating the model uncertainty via Variational Inference, it is possible to compute the information gain by considering N forward passes and sampling the weights from the variational distribution q [BCKW15a], as follows:

$$\underbrace{\mathbb{H}[\mathbf{y}, \mathbf{w}|\mathbf{x}, \mathcal{D}]}_{\text{Model Uncertainty}} = \underbrace{\mathbb{H}\left[\frac{1}{N} \sum_{i=1}^N p(\mathbf{y}|\mathbf{x}, \mathbf{w}_i)\right]}_{\text{Total Uncertainty}} - \underbrace{\frac{1}{N} \sum_{i=1}^N \mathbb{H}[p(\mathbf{y}|\mathbf{x}, \mathbf{w}_i)]}_{\text{Aleatoric Uncertainty}} \quad (2.19)$$

where the total uncertainty is computed as the entropy of the predictive distribution (i.e., Equation (2.9)) approximated as an ensemble average with respect to different parameter settings.

It is possible to show [SH20] that a similar decomposition holds also for other kinds of classifiers, based on ensembles, like Random Forests. In that case, the total uncertainty is given by the entropy of the mean predictions and the aleatoric uncertainty by the mean entropy of the predictions of each classifier of the ensemble [SH20].

The rationale for ensemble models is that the epistemic uncertainty is high when members of the ensemble disagree, by assigning different and highly confident predictions to a given input.

2.3.1 Evaluation of the Uncertainty

In this section, we discuss how to evaluate the quality of a trained model, with a particular focus on its uncertainty awareness. Some methods assess the quality of the predicted probabilities, such as calibration, while others

analyze the previously described decomposition of the total uncertainty. In the following, we describe the main strategies used in the literature.

- **Calibration.** Calibration assesses the quality of a model’s predicted confidence over a given population [Daw82]. It quantifies how well model confidence (i.e., the predictive probability of correctness) aligns with model accuracy (i.e., the observed fraction of correctly classified samples). For example, suppose a classifier predicts that the probability of an input belonging to the c -th class is $p(y = c | x) = 0.8$. In that case, we expect the true label to correspond to this class approximately 80% of the time when the classifier is evaluated on a set of samples. Thus, a well-calibrated model is useful to avoid making the wrong decision when the outcome is too uncertain.

For quantitative evaluation of models’ calibration, several calibration measures can be considered. A widely used measure, is called Expected Calibration Error (ECE) [NCH15]. It is based on binning the predicted probabilities in B equal parts and then the overall aim is to compare the discrepancy between the predicted probability and the empirical probability, obtained by counting.

Let $\hat{y}_n = \operatorname{argmax}_c f(\mathbf{x}_n)$ and $\hat{p}_n = \max_c f(\mathbf{x}_n)$ the predicted class (over C classes) and the corresponding probability (or confidence), for a given input belonging to a dataset of size N . Then, the accuracy computed over the samples falling within the bin b is:

$$\operatorname{acc}(\mathcal{B}_b) = \sum_{n \in \mathcal{B}_b} \mathbf{I}(\hat{y}_n = y_n) \quad (2.20)$$

The average confidence within this bin is:

$$\operatorname{conf}(\mathcal{B}_b) = \frac{1}{|\mathcal{B}_b|} \sum_{n \in \mathcal{B}_b} \hat{p}_n \quad (2.21)$$

The gap between these two quantities defines the ECE of the considered classifier f :

$$\operatorname{ECE}(f) = \sum_{b=1}^B \frac{|\mathcal{B}_b|}{B} |\operatorname{conf}(\mathcal{B}_b) - \operatorname{acc}(\mathcal{B}_b)| \quad (2.22)$$

It is worth noting that this metric is frequentist in nature and refers to average behavior; it does not capture the quality of per-instance uncertainty, which a Bayesian approach, for example, can provide [Mur23]. For this reason, it is common to rely on additional tests to assess a model’s uncertainty awareness.

- **Selective prediction.** This method jointly assesses a model’s predictive performance and quality of uncertainty estimates, by abstaining from making predictions when the model is uncertain [EY⁺10]. A common way is to rely on selective prediction: we select a subset of predictions whose confidence is above a varying threshold and we compute the Accuracy only on the most certain outputs, rejecting the others. By increasing the threshold we expect an increase in the number of Rejections as well as an increase in the overall Accuracy.
- **Out-of-Distribution (OoD) detection.** A relevant practical problem, especially for ML-based IDS, is how a predictive model will behave when it is given an input that is Out-of-Distribution. In this scenario, a desired behavior for the model would be to express that it is not confident in its prediction so that the system can abstain from making predictions. This problem is closely related to the Open set recognition that aims at assessing how well a model can detect examples belonging to none of the training classes [GHC20]. This occurs in cases of semantic class shift, where both the input and label distributions change at test time, in a structured way that introduces new output classes not present in the training data. In this scenario, we expect that an uncertainty-aware model should show a high level of uncertainty for these samples. In particular, in the context of BNN we expect a high epistemic uncertainty and thus it is possible to rely on this quantity to distinguish between in-Distribution and OoD samples.

2.3.2 Examples on a toy dataset

In the previous section, many approaches were presented to perform Bayesian inference on Neural Networks. One of the main was to allow proper uncertainty quantification and avoid the problem of over-confidence in traditional methods.

To illustrate and clarify the problem of overconfidence, we show a comparison of different approaches on a toy dataset for a 2D nonlinear binary classification, known in the literature as the “two moons” dataset [PVG⁺11]. The base classifier is a simple NN with one hidden layer of 128 neurons. The plots of Figure 2.1 show the training dataset (blue and red dots) and the decision boundary of the classifier.

As we can see, in the Bayesian approaches the uncertainty (i.e., the predicted probabilities closer to 0.5) spreads out as long as we move far away from the training dataset, as we might expect. On the other hand, the standard NN shows an over-confident behavior providing predictions with a higher level of confidence. It is worth mentioning to remark that by increasing the flexibility of the considered model (e.g., by increasing the number of

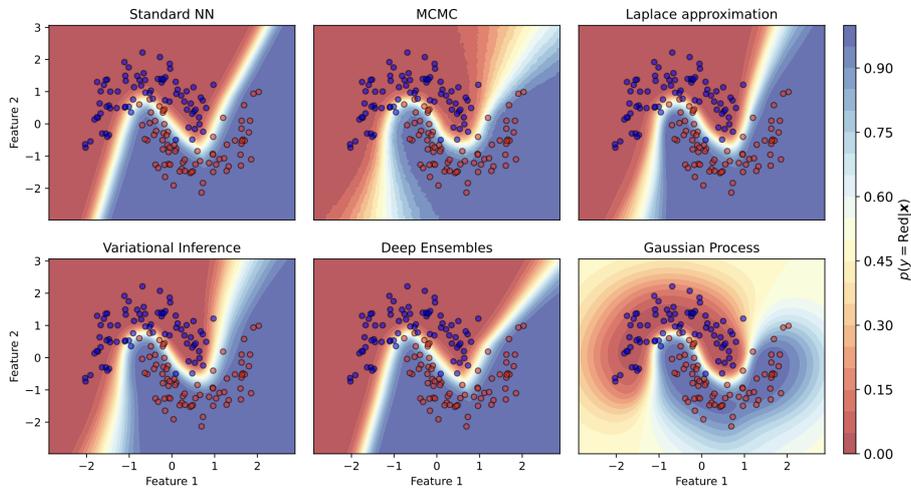


Figure 2.1: Predictions made by different models when presented with the training data shown in blue and red. The baseline is the Standard NN, a simple NN with one hidden layer of 128 neurons, and different approaches to make it Bayesian.

hidden layers and neurons) the uncertainty will increase. A result obtained by Neal [Nea12] shows that in the limit of an infinite number of neurons, a BNN converges to a Gaussian Process, which shows high confidence only for the region of the input space populated by training data points.

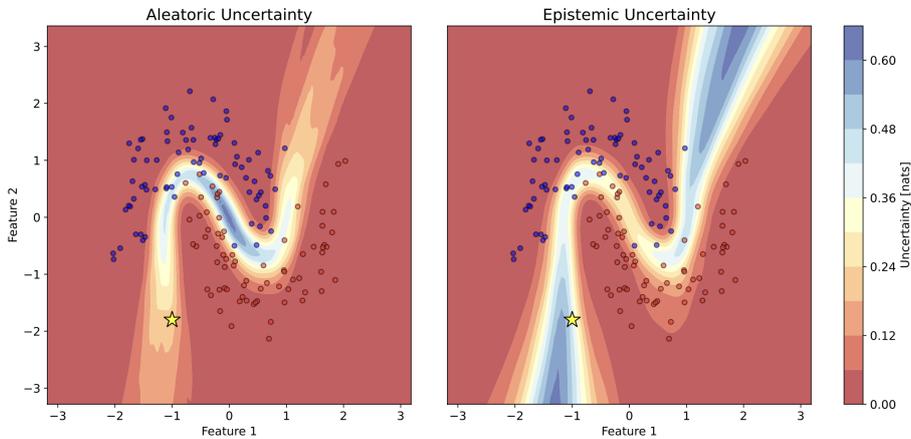


Figure 2.2: Decomposition of Aleatoric and Epistemic uncertainty, obtained through Variational Inference. The yellow star is an example of an OoD input.

Moreover, in Figure 2.2 we illustrate the predicted uncertainty for the previous binary classifier trained using variational inference. In particular, we show the decomposition of the total uncertainty into aleatoric and epistemic components. Aleatoric uncertainty is higher in regions where the two classes overlap, while epistemic uncertainty grows as we move farther from the training data. This decomposition can help identify out-of-distribution

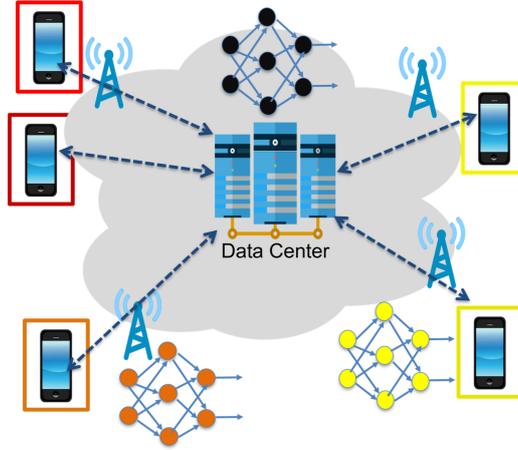


Figure 2.3: Typical setting of a Federated Learning cross-device scenario. Picture taken from [Mar23]

inputs, like the sample represented by the yellow star, for which the classifier exhibits higher epistemic uncertainty.

2.4 Introducing Federated Learning

In this section, we outline a typical federated learning system, introduced in [MMR⁺17], commonly known as cross-device federated learning (illustrated in Figure 2.3). This system involves many devices and a central server, the overall architecture relies on a client-server architecture, where mobile devices communicate solely with the central server.

2.4.1 Problem Formulation

The original federated learning formulation in [MMR⁺17] involves learning a single, global statistical model from data stored on a finite number C of remote clients. The problem is typically analyzed through the lens of statistical learning theory, with the goal of minimizing the following objective function to find an optimal parameter setting $\hat{\mathbf{w}}$ of the global model:

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} F(\mathbf{w}) = \sum_{c=1}^C \frac{N_c}{N} F_c(\mathbf{w}) \quad (2.23)$$

where: F_c is the local loss function for the c -th client, N_c is the number of data-points of the c -th clients and $N = \sum_c N_c$ is the overall number of samples.

The local loss function is generally defined as the empirical risk over the

local dataset $\mathcal{D}_c = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N_c}$:

$$F_c(\mathbf{w}) = \frac{1}{N_c} \sum_{i=1}^{N_c} \ell(\mathbf{w}; \mathbf{x}_i, \mathbf{y}_i) \quad (2.24)$$

where l_c denotes the local loss function (e.g., cross-entropy) for a given set of model parameters \mathbf{w} and an input-output pair (\mathbf{x}, \mathbf{y}) . From the above definition, it follows that the global objective function of Equation (2.23) can be viewed as the empirical loss associated with the aggregated dataset $\mathcal{D} = \bigcup \mathcal{D}_c$.

The most common algorithm to solve Equation (2.23) is FedAvg [MMR⁺17], it is an iterative algorithm that splits the training process into T communication rounds. At the beginning of the t -th communication round, the server selects a set of clients and broadcasts the current model \mathbf{w}_t to the selected clients C_t (Line 4). Upon reception of the model \mathbf{w}_t , each client updates the model, usually through a finite number of local stochastic gradient descent (SGD) updates, using its local dataset \mathcal{D}_c (Line 6). Then, the client sends back the resulting model \mathbf{w}_c to the server (Line 7). Finally, the server aggregates the local update models \mathbf{w}_c at $t + 1$ to produce a new global model \mathbf{w}_{t+1} (Line 9).

Algorithm 1 FedAvg: Federated Averaging [MMR⁺17]

Require: Data $\mathcal{D}_{1:C}$ (data of each client); number of communication rounds T ; number of local epochs E ; learning rate η

- 1: **Server** randomly initializes global model weights \mathbf{w}_1
- 2: **for** each round $t = 1, 2, \dots, T$ **do**
- 3: **Server** selects a subset C_t of clients
- 4: **Server** broadcasts global model weights \mathbf{w}_t to selected clients C_t
- 5: **for** each client c in C_t **in parallel do**
- 6: $\mathbf{w}_{t+1}^{(c)} \leftarrow \text{ClientUpdate}(\mathbf{w}_t, \mathcal{D}_c, E)$
- 7: **Client** c sends updated weights $\mathbf{w}_{t+1}^{(c)}$ to the server
- 8: **end for**
- 9: **Server** aggregates clients' updates:
- 10: $\mathbf{w}_{t+1} \leftarrow \sum_{c \in C_t} \frac{N_c}{N} \mathbf{w}_{t+1}^{(c)}$
- 11: **end for**
- 12: **function** CLIENTUPDATE($\mathbf{w}, \mathcal{D}, E$)
- 13: **for** each local epoch $e = 1, 2, \dots, E$ **do**
- 14: Sample indexes \mathcal{I} uniformly from $\{1, \dots, |\mathcal{S}|\}$
- 15: $\mathbf{w} \leftarrow \mathbf{w} - \eta \sum_{i \in \mathcal{I}} \nabla \ell(\mathbf{w}; \mathbf{x}_i, \mathbf{y}_i)$
- 16: **end for**
- 17: **return** \mathbf{w}
- 18: **end function**

2.4.2 A fundamental Result

The core principle behind federated learning is that, through collaboration, each client can benefit from the data held by all participating clients. This raises a fundamental question of whether participating in global model training is genuinely beneficial for the client [MMRS20; Mar23].

As before, this question has been studied in the literature mostly under the framework of statistical learning theory. The setting is defined as follows. Let \mathcal{H} be a hypothesis class of functions mapping inputs X to their outputs Y and let $d_{\mathcal{H}}$ be the pseudo-dimension of the hypothesis class \mathcal{H} . A given client, say c , has access to a dataset \mathcal{D}_c of N_c samples that are independently and identically distributed according to an (unknown) probability distribution $p_c(X \times Y)$. The client c aims at learning a model (or hypothesis) by minimizing its own population risk, defined as:

$$L_c(h) \equiv L_{P_c}(h) \equiv \mathbb{E}_{(x,y) \sim p_c} [\ell(h(x), y)] \quad (2.25)$$

However, the expectation that appears in the last terms can not be explicitly computed, since it would require the full knowledge of the data distribution p_c , which is usually unknown. Instead, the client can compute an empirical estimation, defined as:

$$L_c(h) \equiv L_{S_c}(h) \equiv \frac{1}{N_c} \sum_{i=1}^{N_c} \ell(h(x_i), y_i) \quad (2.26)$$

It is worth mentioning that a purely local model can show strong generalization capabilities when the size of the training data is sufficiently large [MMRS20]. However, in general, this favorable scenario is not always encountered, therefore, standard FL learning aims to minimize the empirical loss associated with the aggregated dataset, reported in Equation (2.26).

In this setting, it is possible to give an expression for the generalization bound of the global model \bar{h} [MMRS20]. More precisely, let $\delta \in (0, 1)$, then with probability at least $1 - \delta$, the following holds:

$$L_c(\bar{h}) - \min_{h \in \mathcal{H}} L_c(h) = \mathcal{O} \left(\sqrt{\frac{d_{\mathcal{H}} + \log \frac{1}{\delta}}{N}} \right) + \text{disc}_{\mathcal{H}} \left(\mathcal{P}_c, \sum_{c'} \frac{N_{c'}}{N} \mathcal{P}_{c'} \right) \quad (2.27)$$

where: $d_{\mathcal{H}}$ is the pseudo-dimension of the hypothesis class \mathcal{H} and the last term represents the discrepancy associated with the hypothesis class \mathcal{H} defined for two distributions over \mathcal{P}_1 and \mathcal{P}_2 as:

$$\text{disc}_{\mathcal{H}}(\mathcal{P}_1, \mathcal{P}_2) = \max_{h \in \mathcal{H}} |L_{\mathcal{P}_1}(h) - L_{\mathcal{P}_2}(h)| \quad (2.28)$$

The main conclusion of this analysis is the following: since the global model is trained on the combined data from all users, it tends to generalize well overall, since the first terms decrease proportionally to $1/\sqrt{N}$, where N is the size of the dataset. However, the global model may not perform optimally for an individual user due to differences in data distribution across users. Specifically, if the distribution of a given user differs significantly from the ‘mean’ distribution $\bar{p} = \sum_c \frac{N_c}{N} p_c$, the second term on the right-hand side of Equation (2.28) will be large, indicating a potential drop in the predictive performance of the model for that user. This can be the case when the local data of individual clients do not reflect the overall population distribution or the number of data points available on each device can vary widely. This imbalance, along with the broader issue of the data being non-identically and non-independently distributed, has been widely recognized as a fundamental challenge in federated learning.

The first issue related to statistical heterogeneity is the potential slowdown or even the inability of federated learning algorithms to converge [KKM⁺20; LSZ⁺20].

Moreover, statistical heterogeneity makes even the assumption that clients should train a common model questionable. As a consequence, personalized FL has emerged as a promising alternative to learning a general global model [MNB⁺21].

Chapter 3

Trustworthy ML Models

This Chapter demonstrates how ML-based IDS can benefit from uncertainty-aware models and, more broadly, from Bayesian inference. The material covered in this Chapter is adapted and restructured from the following papers:

Contributions

- **J. Talpini**, F. Sartori, M. Savi, *Enhancing Trustworthiness in ML-Based Network Intrusion Detection with Uncertainty Quantification*. Journal of Reliable Intelligent Environments, Aug. 2024
- **J. Talpini**, F. Sartori, M. Savi, *Hierarchical Multiclass Continual Learning for Network Intrusion Detection*, in IEEE Conference on Network Softwarization (NetSoft), Jun. 2024

3.1 Trustworthy ML-based IDS and Related Work

As we discussed in Chapter 1, ML-based IDSs have been extensively investigated in the literature particularly, as said before, in the context of signature-based intrusion detection. These systems have demonstrated great performance in terms of classification scores [LLLT13]. However, the vast majority of the proposed methods in the literature for signature-based intrusion detection rely on the implicit assumption that all class labels are a-priori known. This means that they are designated to perform the classification in a *closed-set* setting where each input is assigned to a category belonging to the set of labels provided during the training phase. However, in practice, a ML model might be presented with input pertaining to classes never seen at training time [APC22]. This scenario is particularly compelling in the context of intrusion detection [SP10], since networks are subject to new

attacks and the likelihood of being targeted by zero-day attacks is getting higher and higher, especially in modern network infrastructures [ZCB21].

Moreover, several studies (e.g. [NYC15; BB16; GPSW17]) have shown that ML models, including Deep Learning ones, tend to produce, overconfident, arbitrarily high per-class classification scores not only to misclassified samples from known classes but also to *Out-of-Distribution* instances, which may be related to unknown classes. This behavior represents a severe issue for the deployment of a *trustworthy* ML-based IDS since we might expect that an IDS will have to face new kinds of attacks or variations of known attacks [SP10]. In fact, it would be desirable to rely on a model that allows network operators or companies offering an IDS service to assess more accurately the *uncertainty* associated with the chosen model’s predictions, thus raising their *awareness* and allowing them to perform more informed risk evaluations while taking the corresponding most appropriate countermeasures accordingly.

We thus argue that for safety-critical applications, such as intrusion detection, the adopted ML model should be characterized not only through the lens of classification performance (accuracy, precision, recall, etc.) but it should also:

1. Provide *truthful uncertainty quantification* on the predictions for *closed-set classification*, a crucial property to avoid making wrong and overconfident decisions when the outcome is too uncertain, to help in the context of risk decision making. Having reliable uncertainty estimates is valuable also for performing *Active Learning* [CGJ96], i.e., a process where a ML-based system could learn from small amounts of data, and choose by itself what data should be labelled by a domain expert. This is a crucial aspect for the training and deployment of ML models in storage and memory-constrained scenarios (e.g. Edge Computing [VWB⁺16]) or when the labelling process of the data is demanding as in network intrusion detection [APC22], where huge amounts of data can be extracted in real-time by exchanged traffic flows.
2. Be able to *recognize as “truly unknowns” inputs belonging to unknown categories*. This can be done by adopting uncertainty-aware classifiers, which can perform at the same time the usual closed-set classification and also be able to detect OoD samples in an *open-set classification* setting (i.e., with also unknown classes at test time)¹.

The goal of this Section is to perform a critical comparison of *uncertainty-aware ML models* and *open-set classifiers* that could be used in the context

¹Here, when referring to *OoD Detection*, we will always implicitly refer to the context of *open-set classification*. This means that the two terms can be used interchangeably, with the assumption that OoD samples belong to unknown classes.

of network intrusion detection, considering an uncertainty-unaware ML-based IDS as a baseline. Moreover, we propose a custom model, based on Bayesian Neural Networks (BNNs), which is designed to offer well-founded uncertainty estimates for the classification of known network intrusions while enhancing the detection of unknown traffic patterns, by reducing the number of false alarms. To this end, we designed a method to recalibrate the uncertainty predicted by a given trained BNN to enforce high uncertainty for inputs far away from the training data, without adding substantial computational overhead. Our illustrative experimental results are obtained on two open-source datasets [AMT⁺20; SLG⁺18], and show that the adoption of uncertainty-aware models based on Neural Networks (NNs), Bayesian Neural Networks and Random Forests (RF) is very beneficial in the considered context, as they are able *(i)* to perform truthful uncertainty classification in a closed-set scenario, *(ii)* while also supporting Active Learning for efficient data labelling, and *(iii)* to enhance OoD Detection with respect to existing ad-hoc open-set classifiers. Moreover, we show how the proposed model stands out for its ability to detect OoD samples, by reducing the false positives, and by showing higher robustness across different OoD experiments, in comparison to other state-of-the-art methods. With our work, we therefore pave the way towards the adoption of uncertainty-aware ML models in risk-sensitive applications, such as intrusion detection, where the problem of uncertainty quantification and OoD Detection is crucial and still in its infancy.

3.1.1 Related Work

In recent years, data-driven approaches for developing signature-based IDSs have been extensively explored (e.g. [SNPS18; TJZ⁺21b; HCS⁺19]) considering different methods such as Random Forests, Support Vector Machines, Neural Networks or Clustering techniques. Various Machine Learning and especially Deep Learning models have emerged as promising data-driven methods with the capability to learn and extract meaningful patterns from network traffic, which can be beneficial for detecting security threats occurring in networked systems [AMT⁺20]. However, it is important to stress that the vast majority of these ML-based IDSs are tested in a *closed-set* scenario. For instance, [VR20; VAS⁺19b] compare different classification algorithms for developing an IDS and, in general, the best performance is achieved by tree-based classifiers, like Random Forests and Multi-Layer Perceptrons (MLPs).

To the best of our knowledge, in the field of ML-based intrusion detection, only a few works have addressed the specific problem of open-set classification to enhance signature-based approaches, while the more general problem of

uncertainty quantification (also beneficial for enhanced closed-set classification and for Active Learning) is still unexplored. In the following, we thus discuss relevant related works (i) on Active Learning based on uncertainty quantification and (ii) on open-set classification in the considered domain.

In the realm of uncertainty quantification in support to *Active Learning* for intrusion detection only a few works can be found in literature: [CSLB20; HBG23; BMGP23] exploit the total uncertainty (more details in Section 3.1.2) of ML models, mainly Neural Networks, to acquire samples to label. We will show why this approach is sub-optimal and how an IDS may take advantage of a more appropriate uncertainty quantification, enhancing the trustworthiness and efficiency of such a process in a closed-set classification scenario.

On the other hand, the literature on the *open-set classification* problem for network intrusion detection is richer. As an early contribution, [KKK19] proposed a hybrid IDS, which combined an anomaly detection module based on Spark ML and a signature-based detection module based on a Convolutional-LSTM network classifier. In this way, it is possible to improve the scalability of intrusion detection by combining an anomaly detection method with a closed-set classifier.

More recent and competitive works based on a single model rather than a hybrid system for tackling the open-set classification problem are [ZZGS21] and [SPG⁺22]: this is the approach investigated here. In [ZZGS21] the authors propose the “Open Set Classification Network” (OCD), a Convolutional Neural Network trained using both fisher loss [YZY19] and MMD loss [GBR⁺12]. The rationale is trying to learn an optimal feature representation in the hidden layers of the network so that feature representations within the same known class are close together, while the feature representations of the unknown class and the known class are as far apart as possible. For that purpose, the authors propose to synthesize samples of possible unknowns to ensure the second phenomenon during the training phase. While this approach may be intriguing, its drawback lies in the necessity for ad-hoc synthetic training data to simulate unknowns. In contrast, here, we focus on methods trained only on known kinds of attacks, as a common supervised classification task, without making any explicit assumption on the possible OoD inputs. We argue that, in general, it will not be possible or practical to make a model aware of all of the possible unknowns. As an alternative, it may be sufficient for the model to detect that an input is ambiguous or novel, and then to react in an appropriate way, or require the intervention of a human expert for taking a decision.

More recently, [SPG⁺22] proposed EFC, an Energy-based Flow Classifier able to tackle the open-set classification problem. It is a statistical model for finding a probability distribution characterizing the per-class flows, and

Table 3.1: Related work on ML-based IDS and their peculiarities.

Paper	Uncertainty Aware	Hybrid Model	Open Set Classification
[SNPS18][TJZ ⁺ 21b][HCS ⁺ 19]	✗	✗	✗
[CSLB20][HBG23][BMGP23]	✗	✗	✗
[KKK19]	✗	✓	✓
[ZZGS21]	✗	✗	✓
[SPG ⁺ 22]	✗	✗	✓
[Guo22][ARI ⁺ 22][HAT ⁺ 20]	✗	✗	✗
Our Work	✓	✗	✓

then it calculates the flow energy to quantify how likely a flow belongs to a given probability distribution. So, if the energy is low for a given flow, it is more likely that it belongs to the set of flows that generated the posterior distribution (see Section 3.1.2) for that class, while if the energy is above a certain threshold, it can be considered as an unknown. We decided to incorporate this method as a reference for our study due to its promising performance in terms of OoD detection with respect to previous models. Moreover, this approach stands out because it can achieve reliable results by utilizing only known classes as training data, eliminating the need to synthesize unknown samples.

Last, it is possible to find a vast literature regarding the problem of *zero-day attack detection* (e.g., [Guo22; ARI⁺22; HAT⁺20]). However, it should be noted that most of these works tackle the problem of detecting unknown attacks as a pure anomaly-detection problem or as a binary classification problem (benign traffic vs. attack). On the other hand, here, we propose the adoption of an end-to-end approach, with a *single model* that can retain the classification performance (in terms of high detection accuracy and recall) of a pure signature-based IDS on known kinds of attacks, while adding the capability of detecting unknowns, an aspect typical of pure anomaly-based solutions. In essence, our analysis differs as we are addressing an open-set classification task: we argue that the development of an uncertainty-aware IDS is a means for reaching this goal, in addition to the previously described advantages compared to usual classification methods.

Table 3.1 summarizes the discussed related works, highlighting their peculiarities: if they are uncertainty-aware, if the proposed approach is a hybrid model, and if the authors tackle the open set classification problem. The symbol ✓ indicates that a model possesses the considered ability, the green color means that it is a positive aspect, red a negative.

3.1.2 Uncertainty Quantification and OoD Detection

In this Section, we begin by presenting the concept of uncertainty quantification and Bayesian Neural Networks. Subsequently, we focus on the problem of Out-of-Distribution detection, discussing specialized methods tailored to address this specific problem and how uncertainty-aware models can effectively address this challenge.

Uncertainty Quantification

A crucial aspect of a model trustworthiness is the quantitative assessment of the model’s uncertainty about its own predictions. We discussed it at the beginning of this Chapter as well as in Chapter 2.

In this Section, we will exploit variational inference for BNN. A general discussion can be found in Section 2.2.2, here we just recall that the Variational Inference approach aims at approximating the posterior $p(\mathbf{w}|\mathcal{D})$ with a tractable distribution $q(\mathbf{w}|\boldsymbol{\theta})$, and adjusting the parameters $\boldsymbol{\theta}$ to get the best approximation by minimizing the ELBO loss [Mac95]: a common assumption for q is a diagonal Gaussian distribution. In [BCKW15a], the authors proposed a principled and backpropagation-compatible algorithm for minimizing the ELBO loss, which makes Variational Inference scalable to complex Neural Networks. In particular, for estimating the model uncertainty via Variational Inference, it is possible to compute the information gain by considering N forward passes and sampling the weights from the variational distribution q [BCKW15a], obtaining the result of Equation 2.19.

It is possible to show [SH20] that a similar decomposition holds also for other kinds of classifiers, based on ensembles, like Random Forests. In that case, the total uncertainty is given by the entropy of the mean predictions and the aleatoric uncertainty by the mean entropy of the predictions of each classifier of the ensemble [SH20]. The rationale is that the epistemic uncertainty is high when members of the ensemble disagree, by assigning different and highly confident predictions to a given input. This is something that we exploited to build an uncertainty-aware RF model (see Section 3.3).

The main drawback of traditional Bayesian Neural Networks is that they typically require multiple forward passes at test time to estimate the mean predictive distribution. As a result, there has been a growing interest in developing methods for uncertainty quantification by employing deterministic single forward-pass neural networks to provide a reduced latency estimation. Among them, it is worth mentioning Deep Deterministic Uncertainty (DDU) [MKvA⁺21], an NN-based approach that employs a feature-space density model as a proxy for the epistemic uncertainty and the entropy of the softmax outputs as a measure of the aleatoric uncertainty. We considered also DDU in our critical comparison of uncertainty-aware models, as we will show later.

Out-of-Distribution detection

As already mentioned, most of ML models are trained based on the closed-world (or closed-set) assumption, where the test data is assumed to be drawn from the same distribution as the training data. However, when models are deployed in an open-world scenario, test samples can be Out-of-Distribution and therefore should be handled with caution, by rejecting them or handing them over to human domain experts [YZLL21].

OoD Detection is a broad topic, and here we concentrate only on the sub-field of *open-set classification* where a model must not only be able to distinguish between the training classes, but also indicate if an input comes from a *semantically* new class it has not yet encountered. Moreover, in our critical comparison, we will consider methods for OoD Detection that do not need training or fine-tuning on OoD data, since these samples may not be available in practical applications. For a survey on OoD Detection the reader should refer to [YZLL21].

Early works observe the overconfidence of Neural Networks and therefore focus on redistributing the logits (i.e., unnormalized Softmax values). In particular, one of the first methods was OpenMax [BB16], which replaces the softmax layer with an OpenMax layer and calibrate the logits with a per-class probabilistic model based on the activation patterns in the penultimate layer of the Neural Network. A more recent and competitive approach in terms of OoD Detection was proposed by [LWOL20]. They suggest an Energy-Based OoD Detection method that uses a scalar energy score, which is lower for observed data and higher for unobserved ones. Moreover, this approach can be applied to any pre-trained neural classifier, without the need of re-training it or to modify its architecture. Given its advantages and peculiarities, we will consider such an Energy-Based model in our critical evaluation.

OoD Detection can be also seen as an application of epistemic uncertainty quantification: since we do not train on OoD data, we expect OoD data points to have higher epistemic uncertainty than in-Distribution (iD) data. Several works [MKvA⁺21; VASTG20a; LLP⁺20a] explicitly leveraged this observation for effective uncertainty quantification and competitive OoD Detection.

In general, the Out-of-Distribution detection problem can be formulated as a *binary classification problem*: for a given input \mathbf{x} and a given (close-set) classifier f the following decision rule g is applied:

$$g(\mathbf{x}, f) = \begin{cases} f(\mathbf{x}), & \text{if } S(\mathbf{x}, f) < \tau \\ \text{unknown}, & \text{if } S(\mathbf{x}, f) > \tau \end{cases} \quad (3.1)$$

where: S is a score associated to a certain input for distinguishing between known and unknown inputs (e.g. epistemic uncertainty), and τ is a threshold

that has to be carefully defined.

3.2 Proposed Framework

3.3 Problem Statement and Models

In this Section, we formalize the three closely-related problems that we investigate and then we briefly describe the models used to address those problems in the context of network intrusion detection.

3.3.1 Problem statement

We consider the following three problems, that ideally should *all* be addressed by a ML-based model adopted for network intrusion detection.

Problem 1 (Closed-Set Classification). The first problem deals with uncertainty quantification on a common (closed-set) multi-class classification problem. A desirable property of an algorithm employed for IDS would be to assign a high degree of uncertainty to its erroneous predictions so that a system administrator may be able to make better decisions and likely avoid severe issues. In fact, a classifier can leverage accurate uncertainty estimates to refrain from making predictions if the uncertainty degree associated to a certain fraction of samples, let's say p , is excessively high, and may ask for the intervention of a human expert. In such case, the classifier will only make predictions on the remaining $(1 - p)$ fraction of samples, i.e., the ones whose prediction is more certain. By relying on its capability of assessing uncertainty effectively, the classifier will prioritize making predictions on instances where it feels more confident. Taking into account two sources of uncertainty (i.e., aleatoric and epistemic) can help well assess uncertainty in closed-set scenarios.

Problem 2 (Active Learning). The second problem we focus on is related to acquiring and labelling large volumes of network traffic for training a ML classifier. In the domain of IDS, the process of acquiring data and label them is complex and demanding, and there is the need of continuously acquire new labeled data [APC22; MLL⁺18; JSD⁺17]. In this context Active Learning [CGJ96] aims at training a ML model in a data-efficient manner by iteratively acquiring only the most relevant samples from a large pool of unlabelled data, rather than annotating all samples, and labelling them with the help of an expert. In particular, in the *active-learning loop*, everything starts by training a model on a small random batch of labeled data. Then new, informative samples are added to the original training set and the model is trained on the updated training set. This procedure is repeated until the

model achieves a desirable classification performance. The ultimate goal is to make the model ensure optimal performance while using the minimum amount of training data.

Equation (2.18) offers a natural and principled way for measuring how a sample is informative for a given model, since the mutual information $\mathbb{I}[\mathbf{y}, \mathbf{w} \mid \mathbf{x}, \mathcal{D}]$ expresses how much knowing the label \mathbf{y} of a given sample \mathbf{x} will reduce our uncertainty about the model parameters \mathbf{w} , and thus can be used as an acquisition strategy for selecting points to label. This approach, known in the literature as Bayesian Active Learning by Disagreement (BALD) [HHGL11], has proven to be highly effective in the context of Deep Learning (e.g. [GIG17; SYL⁺17]). Active Learning effectiveness can also be seen as an additional evaluation of the ability of a model to estimate the epistemic uncertainty and disentangle different sources of uncertainty [MKvA⁺21].

Problem 3 (OoD Detection). The third problem that we tackle can be summarized as follows: given a classifier f trained using a dataset consisting of K known classes (e.g., known kinds of attacks), is it able to recognize as *unknown*, and thus abstain from performing the classification, inputs which belong to new, *semantically* different, classes?

While many ML-based IDS have been proposed in the literature, the vast majority of them are focused on enhancing closed-set classification performance, while not analyzing their predictive uncertainty and the OoD Detection problem, as highlighted in Section 3.1.1 and 3.1.2. In the context of intrusion detection, truthful predictions are crucial for early identification of potentially anomalous network traffic, e.g. new kinds of attacks or variations of known attacks, allowing a network operator to proactively take risk-informed countermeasures.

For instance, an alarm can be triggered if the observed uncertainty or a custom score for OoD Detection exceeds a predefined threshold set by the network operator upon experience. This threshold may be established by considering known traffic patterns and serves as a basis for identifying novel potential intrusions. By comparing the score against the threshold, it is possible to identify suspicious activities that deviate from normal behavior, alerting network operators to potential security threats.

The overall picture of the considered approach for developing an IDS is summarized in Fig. 3.1. The uncertainty-aware model is adopted by an IDS, which can be either a software artifact running in a host or in an Edge Computing node, or a hardware appliance placed at the edge. It is locally trained and is able to acquire in real time packet-related data by means of *packet mirroring*, executed by a border switch, or by exploiting efficient feature extraction capabilities of innovative *programmable data planes* of networking devices [DCKM⁺23]. The model acts on a labeled dataset of

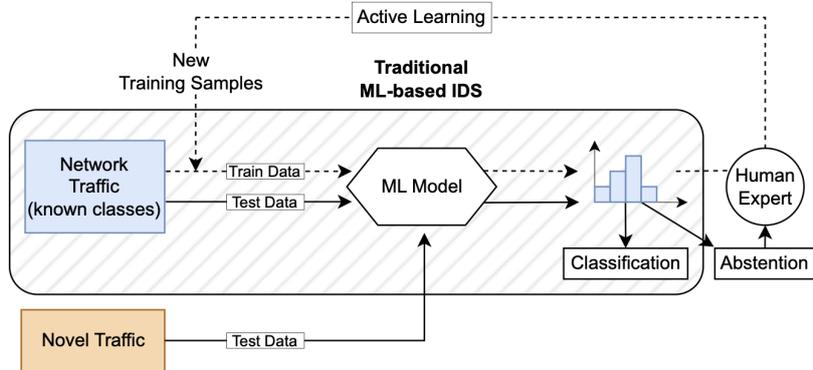


Figure 3.1: Pictorial representation of the proposed approach: an uncertainty-aware ML model is able to recognize unknowns or ambiguous inputs and requires the intervention of a human expert. Moreover, by estimating the epistemic uncertainty, it is possible to perform the Active Learning loop, thus training the model with the smallest possible amount of data.

network traffic, possibly involving the Active Learning loop, and it can provide truthful uncertainty estimates of incoming traffic, possibly asking an expert (e.g. a security engineer) for intervention: this may happen for ambiguous iD inputs or in the case of OoD samples. In the latter situation, unknowns can then be added to the training data, upon appropriate labelling by the human expert, so that the model can be retrained also on new kinds of intrusions.

Once again, we would like to stress that uncertainty-aware models are an appealing foundation for implementing the described pipeline and tackling the three main problems previously described in this Section. In the end, the ambition is to develop a model that behaves as expected across different tasks (e.g. proper uncertainty quantification and OoD detection) and ultimately increase one’s trust in it.

3.3.2 Models

In this Section, we briefly describe the main peculiarities of the models compared and evaluated in this Section. The detailed implementation of each of them is instead described in Section 3.5.1. Table 3.2 summarizes the considered approaches and their ability to perform proper *uncertainty quantification* (by decomposing aleatoric and model uncertainty), *OoD Detection* and *Active Learning*. The symbol ✓ indicates that a model possesses the considered ability. In addition, the Table also specifies whether the approach is based on a NN model or not.

We considered four different types of NN-based models:

- **Neural Network (NN):** This is our *uncertainty-unaware baseline* model. We consider a Multi-Layer Perceptron (MLP), which is a

Table 3.2: Summary of the considered models and their peculiarities

Model	NN-Based	Uncertainty Decomposition	Active Learning	OoD Detection
NN	✓	✗	✗	✗
Energy-Based [LWOL20]	✓	✗	✗	✓
DDU [MKvA ⁺ 21]	✓	✓	✓	✓
BNN	✓	✓	✓	✓
RF	✗	✓	✓	✓
EFC [SPG ⁺ 22]	✗	✗	✗	✓
UC-BNN (Ours)	✓	✓	✓	✓

deterministic model able to provide the conditional probability distribution $p(\mathbf{y}|\mathbf{x}, \mathcal{D})$ over K classes through the softmax activation function [Mur23]. It is common to use the entropy of the softmax distribution as a measure of uncertainty about the classification of a given input. However, different studies [HG18; SG18b] have shown that this quantity presents issues in uncertainty quantification, especially considering OoD data, due to the fact that softmax entropy is inherently not able to capture epistemic uncertainty, as previously discussed. This architecture will be exploited as a basis for other models and methods discussed in the following.

- **Energy-Based** [LWOL20]: This approach is specifically designed for OoD Detection without leveraging on uncertainty quantification. The authors propose an *energy score* to differentiate between OoD and iD samples and mitigate the critical problem of softmax’s low confidence with arbitrarily high values for OoD examples. Specifically, given a trained neural network, denoting the logits corresponding to the y -th class label as $f_y(\mathbf{x})$, they define the energy score as $E(\mathbf{x}, f) = -T \log \sum_{i=1}^K e^{f_i/T}$, where K is the number of classes and T is a free parameter, typically equal to 1 [LWOL20]. During the test phase, inputs with higher energies are considered as OoD samples and vice versa. The main advantage of this method is that it can be applied to a given NN, without the need of modifying the architecture or retrain it.
- **Deep Deterministic Uncertainty (DDU)** [MKvA⁺21]: It is based on approximating the *feature space distribution* as a Gaussian mixture model, through a Gaussian Discriminant Analysis (GDA). More specifically, let \mathbf{z} be the feature representation of an input \mathbf{x} . DDU involves computing the feature density $p(\mathbf{z})$, as a proxy for the epistemic uncertainty, by marginalizing over the classes’ distributions: $p(\mathbf{z}) = \sum_i p(\mathbf{z}|c_i)p(c_i)$, where $p(\mathbf{z}|c_i)$ is the conditional probability

distribution (modelled as a Gaussian) for each class c_i , and $p(c_i)$ is the per-class prior. As a consequence, for a given input, it is possible to avoid the computation of multiple forward passes through the NN at test time, since a proxy of the epistemic uncertainty is estimated by evaluating the density of the feature representation given by the Gaussian mixture density model. The key observation of [MKvA⁺21] is that density-based or distance-based models in the hidden layers of NNs may fail due to the feature-collapse problem [VASTG20b]: feature extractors might map the features of OoD inputs to iD regions in the feature space, without a suitable inductive bias. As a consequence, DDU proposes to rely on spectral normalization [MKKY18] in models with residual connections [HZRS15], in order to encourage a distance-preserving hidden representation of the inputs and hence enhance the OoD detection.

- **Bayesian Neural Network (BNN)**: Also in this case, the architecture is the same as that of NN (e.g. same number of layers, neurons, etc.). The key difference is that in a Bayesian setting, the goal is to learn from a training dataset an ensemble of plausible parameters in the form of a posterior probability density, rather than a point estimate. To get the posterior it must be specified a prior over the weights (see Section 3.1.2). Here we employ the common *zero-mean, isotropic Gaussian distribution* as a prior over the weights: $p(\mathbf{w}) = \mathcal{N}(0, \alpha^2 \mathbf{I})$, where the variance α^2 was chosen to recover the weight decay β of the *L2 regularizer* employed for the NN in the following way: $\alpha^2 = 1/2\beta$ [WI20][Mur23]. Bayesian predictions involve marginalization over the parameters: to evaluate Equation (2.9), we leveraged the Variational Inference approach described in Section 2.2.2 which makes Bayesian inference scalable to complex networks [BCKW15a].

Finally, it is important to observe that BNNs may not consistently exhibit elevated uncertainty for OoD inputs, particularly in the case of simple models, like Multi-Layer Perceptron [Mur23][WI20]. In this case, there might be regions of the input space where simple BNNs can exhibit overconfident predictions, even for inputs far away from the training data (i.e., OoD samples). This observation serves as the starting point for developing our custom model to address this undesired behavior.

- **Proposed approach (UC-BNN)**: Our main goal is to propose a model that shows all the benefits of a BNN model in closed-set classification settings (i.e., meaningful uncertainty estimates and the ability to perform Active Learning) and boost its capabilities of detecting OoD inputs, thus reducing false alarms and without significantly in-

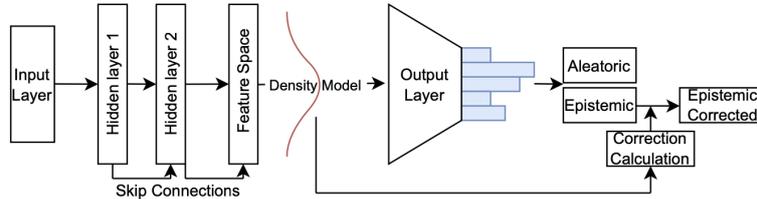


Figure 3.2: Pictorial representation of the proposed model showing the architectural changes with respect to a state-of-the-art BNN and the recalibration process of the uncertainty.

creasing the computational overhead. To achieve this we propose a method for gauging the predicted total and epistemic uncertainty of a given trained BNN model to improve the OoD detection without affecting the iD predictions. We call this model Uncertainty-Corrected BNN (UC-BNN). The core ideas behind the definition of this model are: *i*) Building a meaningful and distance-aware feature extractor, by avoiding the feature collapse phenomenon, as previously described in the NN-based DDU model [MKvA⁺21]. *ii*) After training the BNN, fitting a density model in the feature space of the training data. *iii*) Exploiting the density model to compute the probability density function associated with each input, and use this quantity to re-calibrate both the total and epistemic uncertainty, to reduce overconfident predictions for inputs that are far from the training set. A visual representation of the proposed approach is depicted in Figure 3.2.

To reduce the feature collapse in the feature space and improve sensitivity (step *i*) we add a skip connection to each layer [MKvA⁺21][LLP⁺20b] of the previously described BNN model, so that the output of each layer is now computed as $z_{\text{OUT}} = x_{\text{IN}} + f(x_{\text{IN}})$, where f denotes the activation function, instead of the usual $z_{\text{OUT}} = f(x_{\text{IN}})$. Moreover, as activation functions, we employ LeakyReLU [CUH15] in the first hidden layer and ELU [CUH15] in the second, to further improve sensitivity, since also negative activations can be propagated through the network, and to provide more Gaussian-like activations with respect to the ReLU activation function [CUH15]. After implementing such a distance-aware feature extractor, we model the hidden (or latent) distributions of the training data in the penultimate layer of the network as a multivariate Gaussian (step *ii*)), parameterized by a mean \bar{z} and a covariance matrix Σ . Under this assumption, the probability density associated with the hidden representation z of a given input is uniquely determined by the so-called Mahalanobis distance [Bis] between the mean of the feature distribution \bar{z} and z itself:

$$d(z, \bar{z}) = \sqrt{(z - \bar{z})^T \Sigma^{-1} (z - \bar{z})}. \quad (3.2)$$

The next step involves recalibrating the uncertainty for each input (step *iii*). This process relies on the intuition that we expect a high epistemic uncertainty for the farthest inputs (with reference to the Mahalanobis distance), or, equivalently, for those inputs whose hidden representation falls in the tails of the latent distribution of the training data. The proposed procedure can also be heuristically understood as placing a prior on the expected uncertainty predicted by a model, which should be high for inputs far away from the training data. Considering that the entropy is an additive quantity [Bis], we propose to gauge the total and epistemic uncertainties provided by the BNN model by adding to them a contribution δ that depends on the distance $d(\mathbf{z}, \bar{\mathbf{z}})$ as follow:

$$\delta(d) = \begin{cases} 0 & \text{if } d \leq \tilde{d} \\ 2\epsilon(\text{sigmoid}(\frac{d}{\tilde{d}} - 1) - \frac{1}{2}) & \text{if } d > \tilde{d} \end{cases} \quad (3.3)$$

where: ϵ is the maximum entropy achievable by a flat distribution over N (known) classes and \tilde{d} is a threshold over the distances in the features space. We treat the threshold as an hyperparameter of this model and we set its value as the 99.5% percentile of the distances calculated on the validation set, with the goal of not affecting the predictions on iD data and mitigating the influence of potential outlier values. The particular functional form of Equation (3.3) was chosen to be a continuous function that maps distances to a bounded set, in such a way to not correct the predictions over the iD data and smoothly increases the uncertainty prediction far away from the training data, up to the maximum achievable entropy ϵ . The overall procedure is summarized by the pseudocode in the Algorithm 2. It should be noted that the proposed correction applies to both the total and epistemic uncertainty (lines 9 and 12 of Algorithm 2) so that the aleatoric uncertainty is unchanged, as well as the uncertainty associated to predictions that are already maximally uncertain. In fact, lines 10-12 ensure that both the total entropy and the epistemic uncertainty are at most equal to the maximum entropy ϵ , when $\mathbf{T} + \delta > \epsilon$. In this case, the calibration is determined by the difference between the maximum entropy and the total uncertainty predicted by the model.

For a visual representation of the predictions made by our model, we plot the epistemic uncertainty on a toy dataset in Figure 3.3. These predictions should be compared with those reported in Section 2.3.2: UC-BNN exhibits high epistemic uncertainty far away from the training data, a behavior similar to that of a Gaussian Process, but with the added advantage of being based on a neural network architecture (e.g.,

Algorithm 2 Uncertainty correction (UC-BNN)

Require: Trained BNN

Require: Mean feature vector $\bar{\mathbf{z}}$ and covariance matrix Σ

Require: Threshold \tilde{d}

```
1: function CALIBRATION FUNCTION(input  $\mathbf{x}$ ):
2:   Let  $\mathbf{z}$  be the hidden representation of an input  $\mathbf{x}$ 
3:   Let  $\epsilon$  be the maximum entropy achievable
4:   Predict the class-probabilities  $\mathbf{y}$  of  $\mathbf{x}$ 
5:   Compute Epistemic  $\mathbf{E}$  and Total  $\mathbf{T}$  uncertainty
6:   Compute  $d(\mathbf{z}, \bar{\mathbf{z}})$  ▷ as in Eq.(3.2)
7:   Compute correction  $\delta(d)$  ▷ as in Eq.(3.3)
8:   if  $\mathbf{T} + \delta \leq \epsilon$  then
9:     return  $\mathbf{T} + \delta, \mathbf{E} + \delta$ 
10:  else
11:     $\text{gap} = \epsilon - \mathbf{T}$ 
12:    return  $\mathbf{T} + \text{gap}, \mathbf{E} + \text{gap}$ 
13:  end if
14: end function
```

scalability to large dataset and number of features).

Last, we emphasize that the recalibration procedure applies after the training of the model, which is when the hidden feature distribution of the training data can be considered fairly representative of the iD data distribution. In the Active Learning process, this assumption might not be valid, especially during the first iterations. For this reason, UC-BNN relies on the epistemic uncertainty directly offered by the BNN model to perform Active Learning.

In addition, we considered two different promising models, not based on NNs:

- **Random Forest (RF)**: It is a classification method based on an ensemble of decision trees where each member of the ensemble is trained with bootstrapped samples of a given training set. Each tree can predict the class probability for a given input, which is estimated as the fraction of samples of the same class in a leaf. Then, the overall prediction of the forest is achieved by averaging the predicted class probabilities over different trees, in contrast to the usual approach of majoring vote. Since this model is an ensemble of different predictors, following [SH20] it is possible to exploit the same decomposition of total uncertainty reported in Equation (2.18) (in this case, the sum runs over the number of trees of the forest), to estimate the aleatoric and epistemic uncertainty, and to make RF inherently uncertainty-aware.

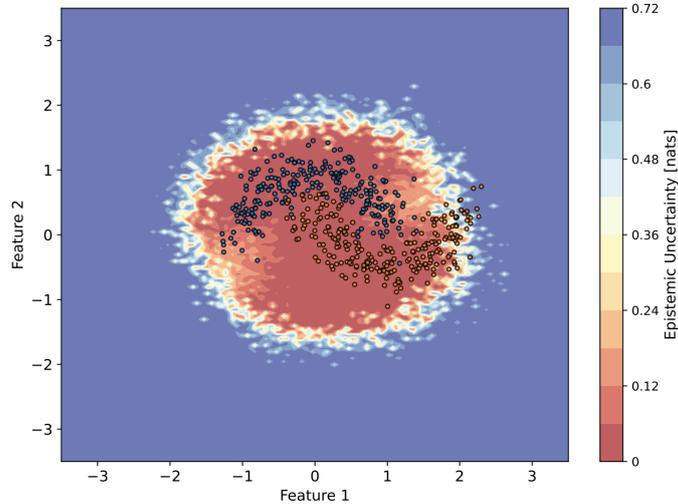


Figure 3.3: Epistemic Uncertainty predicted by the proposed model for the “Two Moons” (see Section 2.3.2) toy-dataset, using a simple MLP with one hidden layer of 128 neurons.

- **Multi-class Energy-based Flow Classifier (EFC) [SPG+22]:** This model is specifically designed for OoD Detection in an intrusion detection scenario without leveraging on uncertainty quantification: it is a statistical model designed specifically for classifying network flows. With EFC multiple distributions are inferred, with each distribution representing a distinct flow class. Subsequently, flow energies are computed within each distribution, and these values are compared to determine the final classification outcome (i.e., known class or unknown).

3.4 Dataset Description and Preprocessing

We consider two open-source datasets: “NF-ToN-IoT-v2”²[SLP21b] (here referred as ToN-IoT) and “CICIDS2017”³ [SLG+18] (here, as CIC-IDS) which are widely used in the literature as a benchmark (e.g., [CSGV+21; TLR22b; TJZ+21a]). Both are organized *per-flow*: each row represents a flow (i.e., source/destination IP pair), for which some additional statistics are computed and also considered as features (e.g. longest flow packet, shortest flow packet, flow duration, etc.) and is annotated as belonging benign traffic or attacks. NF-ToN-IoT includes real collected data and is based on 43 NetFlow [Cla04] features extracted from the network packets (e.g. L4 source port, L4 destination port, TCP flags, etc.), describing the network traffic between different sources and destinations identified by their IP addresses

²Accessible at: https://staff.itee.uq.edu.au/marius/NIDS_datasets/#RA7

³Accessible at: <https://www.unb.ca/cic/datasets/ids-2017.html>

Table 3.3: Samples distribution

Class	Number of Samples	Scenario		
		3U	6U	8U
ToN-IoT				
Benign	6099469	K	K	K
Scanning	3781419	K	K	K
XSS	2455020	K	K	U
DDoS	2026234	K	K	U
Password	1153323	K	U	U
DoS	712609	K	U	U
Injection	684465	K	U	U
Backdoor	16809	U	U	U
MITM	7723	U	U	U
Ransomware	3425	U	U	U
CIC-IDS				
Benign	2095057	K	K	K
DoS Hulk	172846	K	K	K
DDoS	128014	K	K	K
PortScan	90694	K	K	U
DoS GoldenEye	10286	K	K	U
FTP-Patator	5931	K	U	U
DoS slowloris	5385	K	U	U
DoS Slowhttptest	5228	K	U	U
SSH-Patator	3219	U	U	U
Bot	1948	U	U	U
Web Attack-Brute Force	1470	U	U	U
Web Attack-XSS	652	U	U	U

[SLP21b] and presents 10 different classes. On the other hand, the traffic present in CIC-IDS is captured using the CICFlowMeter resulting in 80 different features and 14 different classes of attacks, three of them contain only few tens of samples and therefore have been omitted in the following analysis. For both datasets, among the overall features, we removed from the dataset the timestamp and the source/destination IP addresses in order to only look for intrinsic patterns in the network traffic flows. Moreover, we removed features presenting a constant value (i.e. with a zero variance) across all the samples.

Table 3.3 summarizes the different types of attacks, as long as the distribution of samples (i.e., flows) for each class. In general, we consider two testing scenarios: the usual *closed-set classification* (exploited also for *Active Learning* experiments) and *OoD Detection*. To simulate OoD inputs appearing at run time, we entirely remove multiple classes from the original dataset and consider them as unknowns. We treat the remaining samples,

from the remaining known classes, as a training set to develop a usual multi-class classifier. Datasets with the known classes are partitioned into 60% training, 20% validation, and 20% testing set, and standardized using the training data of known classes, so that each feature distribution presents a zero mean and unit variance. Also, the unknowns are standardized by exploiting the features means and variances computed on the training data. Only with the Random Forest classifier [SH20] we did not standardize the data, since it is not required.

We consider different partitions of the dataset by varying the number of classes moved in the OoD dataset (i.e., whose samples are unknowns) to test the models with different distributions of knowns/unknowns. In particular, we considered as knowns the classes with more samples, in order to mimic a realistic case where an IDS is trained on the well-known and most representative types of network traffic (i.e., the benign traffic and the most common attacks). The considered scenarios are called 3U, 6U and 8U in Table 3.3, where the number in the abbreviation indicates the percentage (30%, 60%, and 80%, respectively) of unknowns among the overall classes, and what classes are unknowns (U) and knowns (K).

As a reference setup for the closed-set classification and Active Learning experiments, we considered the 3U scenario. For further analysis of the OoD Detection setting, we then varied the number of known (K) and unknown (U) classes to check the robustness of the models with respect to changes in the known/unknown distributions. It should be specified that in all the cases we randomly sub-sampled the unknown classes to obtain the same number of per-class unknowns. The reason is that in this way we test models treating the possible unknown distribution equally, without favoring any particular region of the input space.

3.5 Illustrative Numerical Results

3.5.1 Models implementation details

Before getting into the numerical results, we report some details on how we implemented the considered models:

- **NN:** We considered a Neural Network with 2 fully connected hidden layers with 64 neurons each and ReLU as activation function [Mur23]. To reduce overfitting we added 2 layers of batch normalization [IS15] after each single hidden layer. Moreover, for each layer, we employed the L2 regularizer with a weight decay $\beta = 0.1$ [Mur23]. The training process is performed using the Adam optimizer [KB17] with an initial learning rate of 10^{-2} and the cross-entropy as loss function [Mur23]. The training data is presented to the NN in batches of 128 samples for

up to 10 epochs. All the parameters and hyperparameters were chosen to minimize the loss on the validation set: we explored different settings leveraging the Tree Parzen Estimator Bayesian optimization algorithm implemented in the Optuna library [ASY⁺19]. We implemented the classifier using Google’s TensorFlow platform [ABC⁺16] version 2.12.

- **Energy-Based** [LLP⁺20a]: We considered the same implementation of NN with the only difference that we added, in the form of a Python script, the energy-score computation from the logits.
- **DDU** [MKvA⁺21]: Starting from the previously-described NN architecture, we added in the form of a Python script a residual connection to each layer and we estimated the per-class density distribution in the last hidden layer of 64 neurons, following [MKvA⁺21]. For training this model we selected a batch size of 256 samples, which provides a more stable training on the validation set.
- **BNN**: The architecture is the same as the previous NN but without the batch normalization layers. In order to implement the Variational Inference approximation we relied on the TensorflowProbability library [DLT⁺17a] version 0.20, exploiting DenseFlipout layers [WVB⁺18]. We trained the network for 10 epochs with batches of 128 samples using Adam optimizer with a learning rate of 10^{-2} and the ELBO as loss function [BCKW15a]. At test time, in order to obtain the predictive distribution for a given input, we computed 10 forward passes through the network, each time by randomly sampling the weights from the inferred posterior distribution.
- **UC-BNN**: The implementation follows the architecture of previously described BNN architecture, both in terms of parameters and hyperparameters. The only difference, according to the description of the approach reported in Section 3.3.2, is that here we added a first linear layer necessary for implementing the skip connections and we changed the activations function for the two hidden layers from ReLU to LeakyReLU and ELU, respectively. Moreover, we added a skip connection to each layer and we trained the network for 10 epochs with batches of 128 samples, as for the previous BNN model. Also this neural network has been implemented through Tensorflow Probability platform. Lastly, the mean feature vector $\bar{\mathbf{z}}$ and covariance matrix Σ are estimated through the Minimum Covariance Determinant covariance estimator, implemented in Scikit-learn [PVG⁺11]. For making predictions regarding a given input, we computed 10 forward passes to obtain the predictive distribution and the mean feature vector \mathbf{z} , which is then exploited to gauge the predicted uncertainty.

- **RF**: We used the Random Forest Classifier from Scikit-learn [PVG⁺11]. The number of trees within the forest is set to 25, while the other hyperparameters are standard from [PVG⁺11], and we use bootstrapping to induce diversity between the trees of the forest. The described setup has been found to maximize the validation accuracy at a reasonable computational cost.
- **EFC** [SPG⁺22]: All the experiments related to this model are based on the public repository made available by the authors considering its standard parameters [SPG⁺22].

3.5.2 Experimental Setup and Computational Times

All the experiments were performed on a workstation with Ubuntu-Linux operating system, 64 GB of RAM, and equipped with an Intel Core Xeon 8-Core CPU. The software exploited is model-specific and it has been detailed in the previous sub-section for each model.

From the training time perspective, given our experimental setup, Random Forest is the fastest algorithm (~ 40 s on CIC-IDS), while standard MLP-based models (i.e., NN, Energy-Based and DDU) and BNN-based models require roughly one order of magnitude more time (~ 200 s for MLP-based and ~ 250 s for BNN-based on CIC-IDS), while EFC [SPG⁺22] requires another one order of magnitude more time (~ 1500 s on CIC-IDS). On the other hand, at test time, BNN-based models are the slowest ($\sim 20\mu$ s per sample), roughly an order of magnitude more than standard MLP-based models ($\sim 2\mu$ s per sample) and Random Forest ($\sim 1\mu$ s per sample). As said, the difference at test time between BNN-based and MLP-based models is given by the need, for the former models, of computing 10 forward passes instead of just 1, which makes their test time 10x higher.

3.5.3 Closed-Set Classification

In this Section we compare the models through the lens of the usual closed-set classification for the 3U scenario (see Table 3.3) by considering the overall *Accuracy*, the macro F1-Score (*F1-Macro*, i.e., an arithmetic mean of the F1-Score of each class) and the weighted F1-Score (*F1-Weighted*, i.e., a mean of the F1-Score of each class weighted by the number of per-class samples). Moreover, we reported also two common metrics for measuring the *calibration* of a classifier, i.e., the Expected Calibration Error (*ECE*), and the Maximum Calibration Error (*MCE*) [GPSW17][NDZ⁺19].

A model is said to be calibrated if its predicted probabilities match the empirical frequencies: for instance, if a classifier predicts $p(y = k | \mathbf{x}) = 0.8$ for a certain set of inputs, then we expect the class k to be the true label of the

Table 3.4: Test set performance metrics

Closed-set classification						
Model	Accuracy	F1-Weighted	F1-Macro	ECE	MCE	
ToN-IoT						
NN	91.8 ± 0.6	91.7 ± 0.6	86.1 ± 0.7	0.777 ± 0.006	0.839 ± 0.002	
Energy-Based [LLP ⁺ 20a]	91.5 ± 0.6	91.5 ± 0.6	86.5 ± 0.7	0.775 ± 0.006	0.838 ± 0.002	
DDU [MKvA ⁺ 21]	92.6 ± 0.4	92.4 ± 0.3	88.0 ± 0.6	0.785 ± 0.006	0.842 ± 0.002	
BNN	96.55 ± 0.01	96.52 ± 0.01	93.85 ± 0.01	0.810 ± 0.002	0.841 ± 0.002	
RF	98.291 ± 0.001	98.314 ± 0.001	96.652 ± 0.002	0.817 ± 0.0001	0.854 ± 0.001	
EFC [SPG ⁺ 22]	85.11	86.1	72.3	-	-	
UC-BNN (Ours)	96.50 ± 0.01	96.48 ± 0.01	93.52 ± 0.01	0.809 ± 0.002	0.839 ± 0.002	
CIC-IDS						
NN	96.4 ± 0.2	96.5 ± 0.3	81.1 ± 0.4	0.877 ± 0.001	0.869 ± 0.001	
Energy-Based [LLP ⁺ 20a]	96.5 ± 0.2	96.4 ± 0.2	82.3 ± 0.4	0.876 ± 0.001	0.868 ± 0.001	
DDU [MKvA ⁺ 21]	97.1 ± 0.2	97.2 ± 0.2	85.3 ± 0.3	0.878 ± 0.001	0.869 ± 0.001	
BNN	98.95 ± 0.01	98.89 ± 0.01	96.97 ± 0.02	0.860 ± 0.001	0.835 ± 0.001	
RF	98.99 ± 0.001	98.91 ± 0.001	97.252 ± 0.001	0.875 ± 0.0001	0.874 ± 0.001	
EFC [SPG ⁺ 22]	98.85	98.74	94.75	-	-	
UC-BNN (Ours)	98.91 ± 0.01	98.48 ± 0.01	97.13 ± 0.01	0.846 ± 0.001	0.841 ± 0.001	

related inputs for about 80% of the time. It is possible to assess calibration by dividing the predicted probabilities into a finite set of M bins and then, for each bin, assess the discrepancy between confidence and accuracy of samples whose prediction confidence falls into the considered bin. More precisely, the Expected Calibration Error is defined as $ECE = \sum_{i=1}^M \frac{B_i}{n} |\text{acc}(B_i) - \text{conf}(B_i)|$, where n is the number of samples and B_i is the set of indices of samples whose prediction confidence falls into the i -th bin [GPSW17]. For risk-sensitive applications, where reliable confidence measures are necessary, it is possible to rely on the MCE, computed as the maximum discrepancy between confidence and accuracy over the bins [GPSW17]. The lower ECE and MCE, the better.

In Table 3.4 we report the mean value for each metric, along with its standard error. These quantities are computed by repeating the train-test loop 16 times with different random initializations of the training algorithm, to check the robustness of the classifiers with respect to the training procedure. ECE and MCE are computed by partitioning the prediction interval into 10 bins, as in [GPSW17].

On both datasets, Random Forest achieves the highest performance, while NN-based approaches (i.e., NN, Energy-Based, DDU, BNN and UC-BNN) are less competitive. However, it is interesting to note that BNN-based models (i.e., BNN and UC-BNN) perform significantly better than the standard NN (and related methods), despite they basically share the same architecture, on both datasets. The reason for that lies in *how* the two approaches (NN-based and BNN-based) make their prediction and in particular in the Bayesian model averaging of Equation (2.9), which is beneficial also for improving classification performance (in terms of Accuracy and F1-Score), and not only for uncertainty quantification [WI20][SZGL⁺23]. Moreover, it should be noted that BNN-based models, present a more stable performance than traditional NNs, represented by a smaller standard error on the different classification metrics. Note that for EFC we do not report any standard error since this quantity is negligible for this model. Finally, it is worth mentioning that NN, Energy-Based and DDU achieve comparable performance, since the first two models share exactly the same architecture, while the last one only adds residual connections and spectral normalization. Additionally, it can be concluded that the architectural changes introduced by the proposed approach, UC-BNN, have only a marginal impact on the performance of BNN in the usual classification setting. Nevertheless, the proposed model continues to exhibit superior performance compared to NN-based models.

From the point of view of the calibration, all models show similar performance, except Random Forest which shows slightly higher values. We did not report ECE and MCE for EFC as it does not predict a probability distribution as output.

In order to further investigate how well models provide truthful uncer-

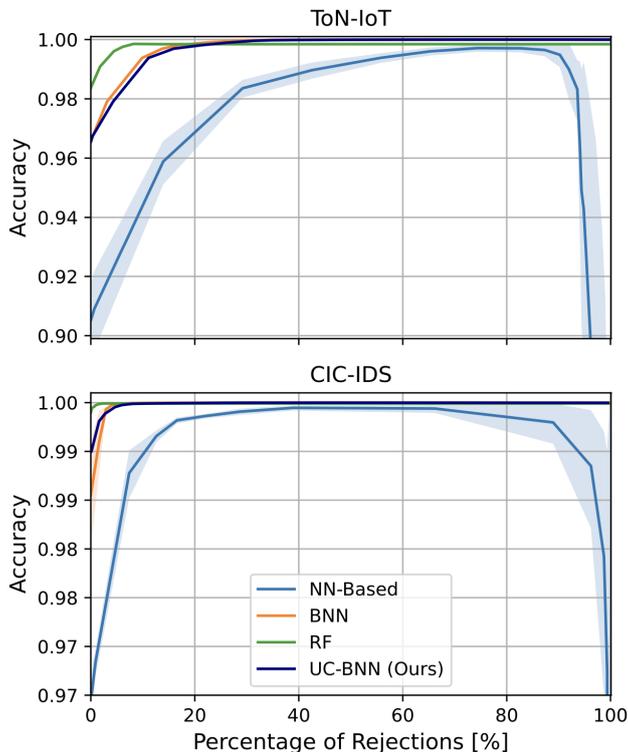


Figure 3.4: Accuracy-Rejections plot for different models and datasets evaluated on the test sets. The accuracy is computed only on a subset of test samples, by rejecting instances classified with a decreasing degree of uncertainty. Solid curves represent the mean accuracy, while bands represent one standard deviation (negligible for RF, BNN and UC-BNN) over 16 repeated experiments. Non-Bayesian NN-based models show overconfident predictions by wrongly classifying the most certain samples.

tainty estimation on their prediction, we select a subset of predictions whose confidence is above a varying threshold and, by applying the decision rule in Equation (3.1), we compute the Accuracy only on the most certain outputs, rejecting the others. By increasing the threshold we expect an increase in the number of Rejections as well as an increase in the overall Accuracy. In Figure 3.4 we report the plots of the Accuracy-Rejections curve for non-Bayesian NN-based models (i.e., NN, Energy-Based and DDU, which perform the same), BNN models (including our proposal UC-BNN) and Random Forest. EFC cannot be evaluated from this point of view since it does not provide uncertainty on its predictions. More specifically, we report the mean accuracy and the standard deviation computed over 16 experiments.

It is possible to notice that in the range of low Rejections the Accuracy increases monotonically for all models, but for high Rejection percentages (i.e., by keeping only the most certain samples), the Accuracy of non-Bayesian NN-based models drops. This reflects the fact that standard NNs tend to give misleading confidence predictions, as reported also in [LPB17][GG16] and

detailed in Section 3.1.2, leading those models to assign high confidence to samples that are, in fact, wrongly classified. On the other hand, BNN-based models, including UC-BNN, show a monotonic increase in accuracy, which reflects the expected behavior of proper and meaningful uncertainty estimates, and the same phenomenon is experienced also by RF. It is also worth noting that the proposed approach, UC-BNN, does not alter the performance of BNN in terms of uncertainty estimates on iD data, as expected during the design of this method. It is worth mentioning that we also found similar behaviors to those appearing in Figure 3.4 for the F1-Score, but we do not report the plot for the sake of conciseness.

In conclusion, this analysis emphasizes the importance of having proper uncertainty estimates: if the confidence estimates are meaningful, one can trust the model’s predictions when the reported uncertainty is low and rely on a different strategy (e.g. the intervention of a human expert) when the model is not confident, especially if the cost of a wrong prediction may be severe.

3.5.4 Active Learning

As we already mentioned, the real-time collection and labelling of large volumes of network traffic present several issues. Active Learning can exploit the epistemic uncertainty estimation of a given model to acquire (and hence label) the smallest amount of data that is relevant for the training process. We evaluate different models exploiting different acquisition functions: *BALD* (i.e., the epistemic uncertainty), the *total uncertainty* (exploited for instance in [SYL⁺17]), and a *random* acquisition function as a baseline.

We begin with a detailed analysis of the numerical results obtained for the ToN-IoT dataset, followed by those for the CIC-IDS dataset.

In particular, for ToN-IoT, we started the Active Learning loop with 10^5 samples (i.e., $\sim 1\%$ of the overall dataset) and added samples until the model reached the desired classification score, with an acquisition size of 10^5 samples at each step. The acquisition size was chosen to have a reasonable trade-off between collected size and computational cost while performing the experiments. At each step, the model is trained on the test set and evaluated in terms of the Macro F1-Score, to give the same weight to each class, and these experiments have been repeated 5 times for each model and acquisition function. To summarize the results we report the mean Macro F1-Score and the standard deviation, computed on the test set, as a function of the acquired training dataset size, expressed as a percentage of the full training set.

Figure 3.5 shows the results for the BNN-based models, including UC-BNN: BALD acquisition function performs better than acquiring data at

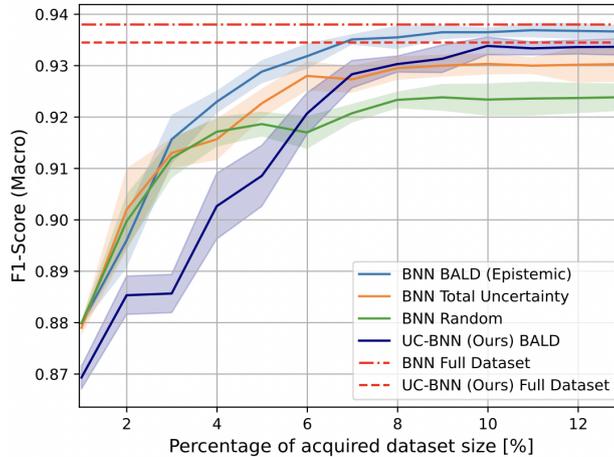


Figure 3.5: Active Learning experiments with BNN-based models for the ToN-IoT dataset. Solid curves represent the mean Macro F1-Score for each acquired batch of data, while bands represent one standard deviation.

random and selecting points exploiting only the total predictive uncertainty. By exploiting BALD as an Active Learning acquisition strategy it is possible to reach the expected F1-Score (i.e., that obtained by considering the full training dataset) with just $\sim 11\%$ of the samples, which may be seen as a significant reduction in the effort of storing data, especially in storage and memory-constrained scenarios (e.g. Edge Computing), and in human intervention for labelling them. The results once again confirm the significant benefits brought by a decomposition of aleatoric and epistemic uncertainty. Finally, it is noticeable that the architectural differences between UC-BNN and BNN initially affect performance within the Active Learning loop. However, after a few iterations, even UC-BNN is capable of converging to its optimal performance by leveraging the model’s predicted epistemic uncertainty, without any correction, as a sampling strategy. We can thus conclude that, in the long-term, UC-BNN does not significantly affect Active Learning capabilities of BNN despite its architectural changes, although the F1-score stabilized to a slightly smaller value.

Figure 3.6 shows the Active Learning experiments results for non-Bayesian NN-Based models. In particular, we tested NN by acquiring data exploiting a random acquisition function and the total uncertainty, given by the entropy of the softmax layers output. Energy-based is not considered as its obtained results are the same as for NN. Then, we employed DDU to sample point according to their density in the features space, so that a decomposition of epistemic and aleatoric uncertainty can be performed and the former can be used by the Active Learning loop. In this case, it was possible to obtain a faster convergence to the performance obtained with the overall dataset. However, it should be noted that traditional NN-based models (including

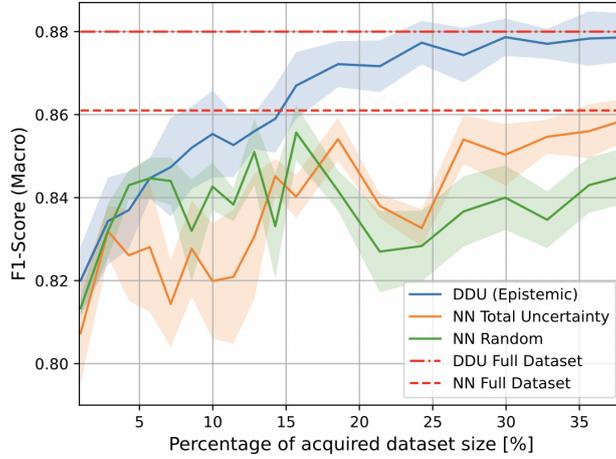


Figure 3.6: Active Learning experiments with non-Bayesian NN models (i.e., standard NN and DDU) on ToN-IoT. Solid curves represent the mean Macro F1-Score for each acquired batch of data, while bands represent one standard deviation.

DDU) require much more data than BNN-based ones ($\sim 35\%$ of the dataset samples) to reach the desired performance, and thus are in general less competitive for Active Learning.

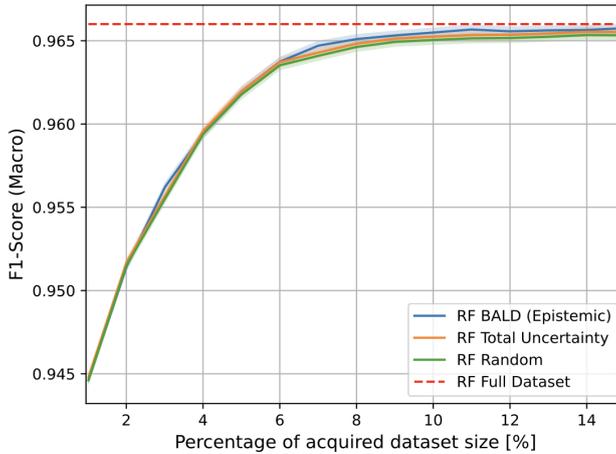


Figure 3.7: Active Learning experiments with Random Forest on ToN-IoT. Solid curves represent the mean Macro F1-Score for each acquired batch of data, while bands represent one standard deviation.

Figure 3.7 shows the experiments for the Random Forest. In this case sampling with BALD and the total uncertainty gives almost comparable performance with respect to a random acquisition function. The fact that batch-based Active Learning does not significantly improve Active Learning for Random Forest was already noted in the literature in different contexts [NYKW12][SKF⁺23], not related to network traffic classification. However,

the obtained results make it possible to conclude that Random Forest requires a much smaller amount of training data with respect to NN-based approaches, as with a random acquisition it is possible to reach the desired performance with only $\sim 10\%$ of samples of the full training dataset.

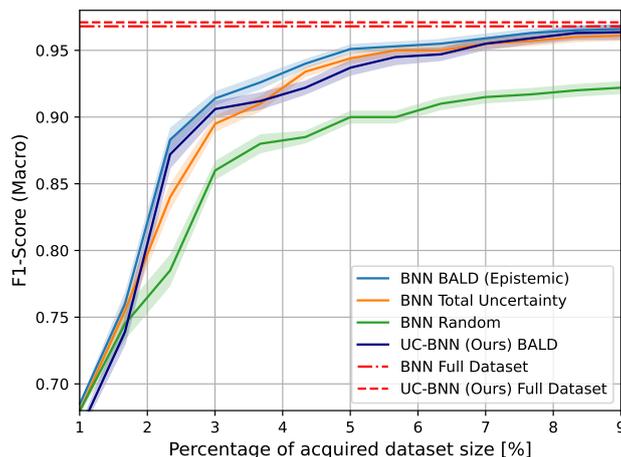


Figure 3.8: Active Learning experiments with BNN-based models on CIC-IDS. Solid curves represent the mean Macro F1-Score for each acquired batch of data, while bands represent one standard deviation.

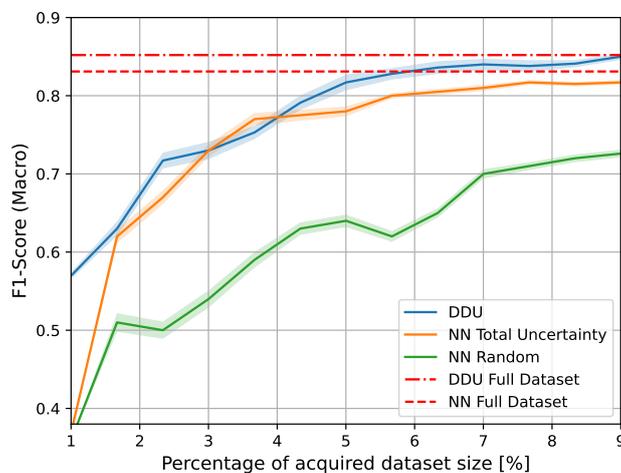


Figure 3.9: Active Learning experiments with non-Bayesian NN models on CIC-IDS. Solid curves represent the mean Macro F1-Score for each acquired batch of data, while bands represent one standard deviation.

We conducted the same experiments on CIC-IDS, beginning with an initial training dataset that consisted of 1% of the total training samples, and utilizing acquisition steps that corresponded to 1% of the entire dataset. Figure 3.8 shows the Active Learning experiments regarding the BNN-based models: once again, BALD acquisition strategy outperforms the other

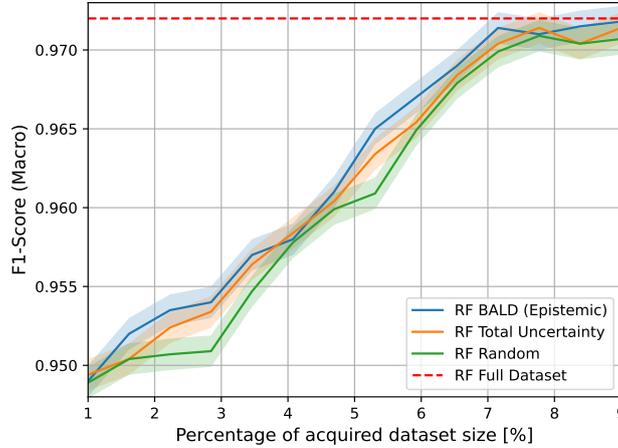


Figure 3.10: Active Learning experiments with Random Forest on CIC-IDS. Solid curves represent the mean Macro F1-Score for each acquired batch of data, while bands represent one standard deviation.

acquisition strategies with UC-BNN converging to its optimal performance using only approximately $\sim 9\%$ of the entire dataset. Figure 3.9 and Figure 3.10 report the results for non-Bayesian NN models and RF, respectively, with similar trends obtained on ToN-IoT.

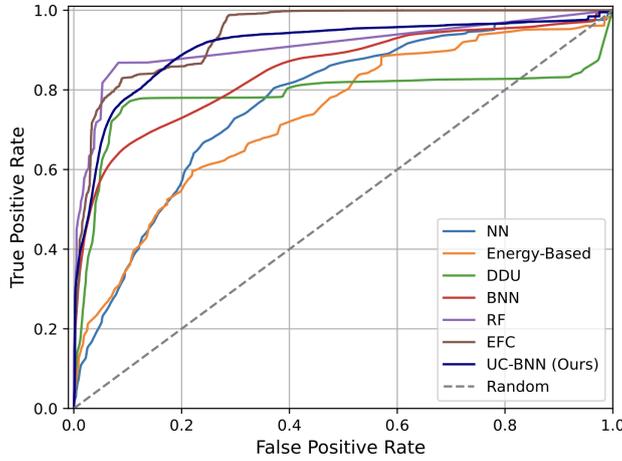
3.5.5 Out-of-Distribution Detection

In this Section we discuss the numerical results for OoD Detection. In order to evaluate the performance of different models, we formulate OoD Detection as a *binary classification* problem (known vs. unknown) so that the comparison may be performed in terms of *ROC Curve* and Area Under the ROC (*AUROC*) [Bra97] for each model (the higher AUROC, the better). In this way it is possible to fairly compare the classifiers in a threshold-independent manner and to evaluate their discriminative performance in terms of OoD Detection. Moreover, AUROC has the advantage of being insensitive with respect to the imbalance of the classes (i.e., knowns/unknowns) [Bra97], which makes this metric particularly well-suited in our scenario.

It should be emphasized that a good IDS should operate in a regime of low False Positive Rate (*FPR*), i.e., it should provide a low rate of false unknowns. For this reason, we report also the AUROC in the region below 20% of false positives, to compare the different methods in a realistic region of deployment and assuming that 20% of false positives is the maximum acceptable value. The AUROC below the 20% of FPR has been normalized (denoted with $*$) so that a perfect classifier with $TPR = 1$ for each value of FPR will have an *AUROC20** of 1. Moreover, as a reference, we report also the AUROC for a random classifier.

Table 3.5: Test set performance metrics for ToN-IoT with three unknown classes

OoD Detection 3U (ToN-IoT)		
Model	AUROC20*	AUROC
NN	0.34 ± 0.02	0.75 ± 0.01
Energy-Based [LLP ⁺ 20a]	0.38 ± 0.02	0.76 ± 0.02
DDU [MKvA ⁺ 21]	0.65 ± 0.01	0.805 ± 0.007
BNN	0.61 ± 0.02	0.84 ± 0.01
RF	0.789 ± 0.001	0.898 ± 0.001
EFC [SPG ⁺ 22]	0.74	0.94
UC-BNN (Ours)	0.69 ± 0.02	0.91 ± 0.02
Random	0.1	0.5

**Figure 3.11:** Mean ROCs for the 3U scenario of ToN-IoT.

As described in Section 3.6.4 we partitioned both datasets by varying the fraction of known/unknown classes. For the sake of conciseness, we report a detailed analysis of the results for the ToN-IoT dataset, while for CIC-IDS we report the average results across the different scenarios.

In the first scenario (*3U scenario*), we considered the models trained on 7 known classes whose closed-set classification performance is summarized in Table 3.4. After such training, we fed the models with input belonging to the three remaining (novel) classes. In Table 3.5, the results are summarized with mean values and corresponding standard errors. By looking at AUROC20*, in this setting, RF seems to be the most promising method, followed by EFC, and UC-BNN among the NN-based models. Figure 3.11 shows the ROC curves for the considered models. It can be observed that certain models exhibit higher ROC curves in specific regions of False Positive Rate (FPR) and lower curves in other regions (e.g., DDU performs better at low FPR than BNN, but worse at high FPR). As a result, this behavior may lead to

Table 3.6: Test set performance metrics for ToN-IoT, with six unknown classes

OoD Detection 6U (ToN-IoT)		
Model	AUROC20*	AUROC
NN	0.39 ± 0.02	0.77 ± 0.02
Energy-Based [LLP ⁺ 20a]	0.50 ± 0.02	0.815 ± 0.02
DDU [MKvA ⁺ 21]	0.63 ± 0.01	0.85 ± 0.01
BNN	0.65 ± 0.02	0.89 ± 0.04
RF	0.633 ± 0.001	0.804 ± 0.001
EFC [SPG ⁺ 22]	0.58	0.88
UC-BNN (Ours)	0.72 ± 0.02	0.92 ± 0.02
Random	0.1	0.5

Table 3.7: Test set performance metrics for ToN-IoT, with eight unknown classes

OoD Detection 8U (ToN-IoT)		
Model	AUROC20*	AUROC
NN	0.40 ± 0.02	0.75 ± 0.02
Energy-Based [LLP ⁺ 20a]	0.51 ± 0.02	0.84 ± 0.02
DDU [MKvA ⁺ 21]	0.56 ± 0.01	0.89 ± 0.01
BNN	0.72 ± 0.02	0.91 ± 0.02
RF	0.780 ± 0.002	0.797 ± 0.002
EFC [SPG ⁺ 22]	0.23	0.74
UC-BNN (Ours)	0.74 ± 0.02	0.94 ± 0.02
Random	0.1	0.5

AUROC values that are misleading when compared and justifies our decision to focus on AUROC at low FPR (AUROC20*) for a fair comparison between methods. It is possible to notice that UC-BNN significantly improves upon BNN, especially at low false positives.

To further investigate the behaviour of those models, we considered two additional scenarios by reducing the number of known classes and increasing the unknowns.

We started by considering 40% of classes as known (i.e., Benign, DDoS, Scanning and XSS) and the remaining 60% as unknowns (*6U scenario*). Since here we are interested in OoD Detection, we did not reported the closed-set classification performance. However, it is worth mentioning that the closed-set performance increased by almost 2% of F1 Score and Accuracy for each classifier, with respect to the 3U scenario. Table 3.6 summarizes the metric values for novelty detection: in this scenario, UC-BNN exhibits the highest values for AUROC and AUROC20*, while BNN performs slightly better than Random Forest and DDU. For the sake of conciseness, we did not reported the plot of ROC curve for this intermediate scenario.

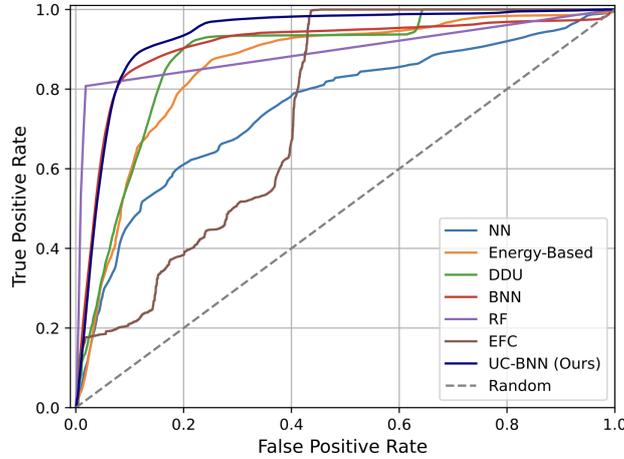


Figure 3.12: Mean ROCs for the 8U scenario of ToN-IoT.

The last experiments were performed on the *8U Scenario*, which is represented by only 20% of known classes (i.e., Benign and Scanning) and 80% of unknown classes. This is an extreme case, where malicious traffic is mainly unknown. In this setting, the three best models are represented by RF, UC-BNN and BNN, while EFC performance drops (see Table 3.7). Figure 3.12 shows the ROC curves for the models in this scenario. In this scenario it is possible to notice that UC-BNN outperforms other NN-based models, and it exhibits the highest True Positive Rate across a wide region of interest of low False Positive Rate, between ~ 0.07 and ~ 0.4 .

We tested the models also on the CIC-IDS dataset, by exploiting the same aforementioned scenarios of known/unknowns ratios. Table 3.8 reports the average results, along with the standard deviation (*STD*) over the three different scenarios, for both datasets. In general, it is possible to state that some methods (i.e., EFC, RF and DDU) are very sensitive to the particular knowns-unknowns setting: this is reflected on high values of the standard deviation of the AUROC, especially the AUROC20*. Moreover, it is possible to notice that RF, BNN-based models and DDU, i.e., the uncertainty-aware methods able to decouple epistemic and aleatoric uncertainty, lead to decent results in terms of OoD Detection capabilities and outperform the other approaches, including the ad-hoc ones such as Energy-Based and EFC. Overall, the best-performing models in terms of AUROC20* are RF and UC-BNN, however UC-BNN exhibits more stable and consistent performance across different scenarios, as indicated by its low value of standard deviation (*STD*), more than almost three times lower than RF. Moreover, it is possible to notice that regarding the overall AUROC metric, UC-BNN presents the highest value on ToN-IoT, while on CIC-IDS it is at the same level of DDU. It is worth mentioning that the OoD detection performance of the considered

Table 3.8: Overall test set performance metrics

OoD Detection (Average)					
Model	AUROC20*		AUROC		
	Mean	STD	Mean	STD	
ToN-IoT					
NN	0.38	0.03	0.76	0.01	
Energy-Based [LLP ⁺ 20a]	0.46	0.06	0.80	0.03	
DDU [MKvA ⁺ 21]	0.63	0.05	0.85	0.03	
BNN	0.66	0.05	0.88	0.03	
RF	0.73	0.07	0.83	0.05	
EFC [SPG ⁺ 22]	0.55	0.23	0.85	0.08	
UC-BNN (Ours)	0.72	0.02	0.92	0.014	
CIC-IDS					
NN	0.15	0.03	0.68	0.04	
Energy-Based [LLP ⁺ 20a]	0.16	0.04	0.68	0.05	
DDU [MKvA ⁺ 21]	0.39	0.08	0.73	0.07	
BNN	0.30	0.03	0.70	0.03	
RF	0.41	0.06	0.69	0.05	
EFC [SPG ⁺ 22]	0.36	0.03	0.69	0.05	
UC-BNN (Ours)	0.39	0.02	0.73	0.03	

models is lower on the CIC-IDS dataset, but shows consistent behavior with that observed on ToN-IoT.

In particular, it is possible to state that uncertainty-aware models are particularly interesting, since they give also good results in the usual closed-set classification and when adopted in the Active Learning loop, being thus the most promising methods to address *all* the three problems related to intrusion detection introduced in Section 3.3.

Overall, it is possible to notice that *uncertainty-aware RF* [SH20] and UC-BNN should be considered as a strong baseline in the field of network intrusion detection, although RF presents a higher variance in the results with respect to BNN and especially UC-BNN. Furthermore, we want to highlight that the UC-BNN method demonstrates superior performance compared to other NN-based models. This underscores the effectiveness of the proposed approach in enhancing OoD detection capabilities without adversely affecting closed-set classification performance. Last, it should be noted that standard NNs give poor performance in OoD Detection: this highlights the importance of enhancing NNs with uncertainty awareness (e.g. by adopting BNN-based models or DDU) for developing a NN-based trustworthy IDS.

3.5.6 Discussion

The previous Sections focus on three fundamental and closely-related problems in the field of trustworthy ML-based network intrusion detection: *(i)* avoiding dangerous overconfident predictions in typical closed-set classification settings, *(ii)* performing efficient Active Learning on incoming network flows, and *(iii)* enabling Out-of-Distribution Detection to effectively identify unknowns. We argue that these problems should *all* be addressed, and the goal of our research has been assessing various ML models to solve them, with particular interest on methods able to guarantee proper uncertainty quantification.

Our research reveals that conventional Neural Network-based approaches are inadequate for trustworthy network intrusion detection, due to their limited capability of quantifying predictions' uncertainty and detecting Out-of-Distribution samples. In contrast, *uncertainty-aware* methods such as BNN-based models and Deep Deterministic Uncertainty demonstrate promising potential to solve all the three problems.

Specifically, we propose a custom BNN-based model, called UC-BNN, which improves OoD detection capabilities with respect to standard BNN and stands out for its robustness, providing the most consistent results in the different OoD detection experiments, without significantly reducing the closed-set classification performance. In addition and not surprisingly, our analysis brings out uncertainty-aware Random Forests as another strong baseline for building a trustworthy ML-based IDS.

As a future work, we would like to leverage *continual learning* and *class-incremental learning* to update the uncertainty-aware ML model at run time with novelties, without forgetting the previously learned classes. Moreover, it would be interesting to investigate the relations between feature importance and uncertainty quantification. Last, we would like to explore distributed learning approaches (e.g. *Federated Learning*) to build uncertainty-aware models in a collaborative way and make them fully suitable for Edge Computing operations.

3.6 Dynamic Environment

At the end of the previous Section, we posed the question: *What can we do with the unknown samples?*

In this Section, we address this point by leveraging Bayesian inference to enable a model to learn continuously from an evolving data stream.

3.6.1 Introduction

As we said in the first chapter, traditional approaches to IDSs rely on *knowledge-based* systems [HCS⁺19]: however, the increasing complexity of networks makes those systems more prone to errors [TJZ⁺21b; SNPS18]. As a consequence, data-driven approaches based on ML have been widely considered in recent years for detecting attacks, demonstrating great performance in terms of classification scores [LLLT13].

However, the vast majority of the proposed methods in the literature on ML-based IDSs are based on a *static* framework, where a model is trained once on a given training set, which is supposed to be representative of all possible network patterns encountered during the deployment. On the other hand, we may expect that in a real-world scenario, intrusions do not emerge at once but gradually over time, so that the original training dataset will eventually become outdated. Thus, there is the need for a *continuous* training of an IDS to accommodate new intrusion variants, i.e., we require that a model should be able to learn from a subsequent collection of potentially different data distributions, referred to as *tasks*, and revealing new attacks. This process becomes particularly challenging when it is not feasible to store all potential historical data, and instead, older data are permanently discarded. This is the typical case occurring in *resource-constrained* scenarios such as when intrusion detection capabilities are deployed at the edge in shared virtualized environments, or in the far-edge (e.g. in IoT gateways) [ZZDZ23]. In this setting, the main issue is represented by the so-called *catastrophic forgetting* [KPR⁺17], i.e., the phenomenon according to which the performance on older data drops when the model is re-trained on the newest ones.

Figure 3.13 illustrates the impact of a distribution shift resulting from the emergence of novel attack types. The scatter-plots are generated using manifold learning through t-Distributed Stochastic Neighbor Embedding (t-SNE) [VdMH08] to reduce the dataset dimensionality and allow a 2D representation. This is performed for two distinct task categories of the considered dataset (see Section 3.6.4), providing an intuitive insight into the necessity of retraining an IDS to maintain high predictive capabilities for both new and older data distributions.

Here, we exploit *Continual Learning* (CL) [PKP⁺19] capabilities to

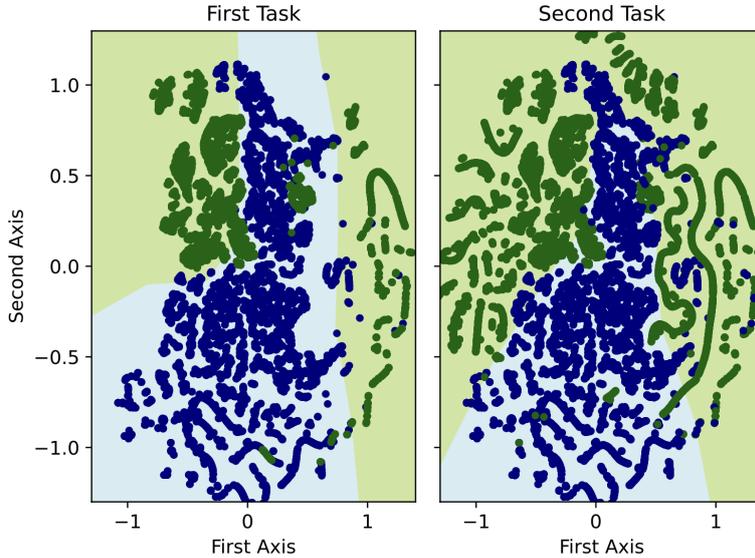


Figure 3.13: A pictorial representation of Continual Learning on a sequence of two tasks. Blue dots represent benign traffic while dark green ones are attacks, light blue and green areas represent the decision contour of a binary classifier, namely an IDS, that should adapt to the appearance of new attacks in the second task, while keeping good performance on the older one.

achieve this goal. The need for CL is particularly relevant when it may not be feasible to store large volumes of old data, which constitutes the primary setting explored here. To this end, we suggest leveraging the inherent hierarchy in the intrusion detection problem (i.e., first predicting whether a flow is an attack and then determining its type) to create a model that can be updated continuously as new attacks emerge. In particular, we propose to rely on a parametric discriminative model (e.g. a Neural Network) as a binary classifier for distinguishing between benign and attack flows. This model is based on Bayesian Inference [NLB⁺18] so that it can be continuously and effectively trained on new tasks. Additionally, we propose incorporating a straightforward generative classifier (e.g. Quadratic Discriminant Analysis) for modeling the residual classification problem of different attacks, as it occurs in our considered class-incremental setting.

Our illustrative results show that the proposed model is able to learn new data patterns, while preserving good classification abilities on the older ones, without storing historical data.

3.6.2 Related Work

The application of ML to network intrusion detection has been receiving considerable attention as it enables automated analysis of network traffic by leveraging past data [TJZ⁺21b; SP10]. For instance, [VR20; VAS⁺19b] compare different classification algorithms to develop an IDS and, in general,

the best performance is achieved by tree-based classifiers, like Random Forests, and Multi-Layer Perceptrons (MLPs). However, as already said, most existing works adopt a *static* approach, where training occurs once on a specific dataset, assuming that test data originates from the same distribution. This limitation is significant given the *dynamic* nature of communication networks, where new network traffic and attacks may emerge during deployment. As a consequence, the field of CL has gained more and more attention, also in the context of intrusion detection, for implementing both anomaly and signature-based IDSs. In the following, we analyze the most relevant work tackling this problem in the context of signature-based IDSs, as it is strictly coupled with multiclass classification.

In [PSM⁺22] the authors propose a comparison of different strategies for addressing the CL problem in the field of intrusion detection. The primary discovery indicates that models for CL based on *replay* surpass conventional statistical methods, as well as cutting-edge Boosted Decision Trees and Deep Neural Network (DNN) models, in effectively addressing the distribution shift problem. A similar conclusion was reached in [OIIT23]. In both these works the experiments are based on a setting where all the classes are a-priori known and then the model should adapt only to the covariate shift happening for each known class. In contrast, in our work, we tackle the problem from a class-incremental perspective where the number of classes is not a-priori known, which we believe better sticks to the real world.

On the other hand, [WLHS21] analyzes the problem of adapting a signature-based IDS to new kinds of attacks in a few-shot regime. However, the solution proposed by the authors relies on having access also to a certain number of historical data to adapt the classifier. In this work, we instead analyze a scenario where historical data may no longer be available while executing the current task, due to privacy reasons or due to stringent resource constraints. Another contribution that analyzes a similar setting as the one considered in this manuscript is [DA22] which, however, exploits different models, not based on NNs. The authors propose an ensemble Incremental Learning algorithm based on Hoeffding Tree, Adaptive Boosting and Hard Sampling, which is capable of learning new attacks without forgetting the previously learned traffic patterns. This approach shows encouraging results but it still requires a list of samples from older tasks that are hard to classify.

3.6.3 Problem Formulation and Adopted Models

The field of Continual Learning is getting more and more attention since for several applications data comes in a sequence of datasets that are used for training and then permanently discarded, as it should occur for network traffic. More formally, the problem statement may be formulated as follows: given

T subsequent and disjoint datasets \mathcal{D}_t , called *tasks*, i.e., $\mathcal{D}_t = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N_t}$ each with N_t input-output pairs, we want to train a parametric model f , characterized by a set of weights \mathbf{w} , sequentially on each task (i.e., $\mathbf{y}_i = f_{\mathcal{D}_i}(\mathbf{x}_i, \mathbf{w})$) without forgetting the knowledge gained from the previously encountered tasks. In the context of this study, our objective is to develop a classifier that can categorize network traffic for each task, defined by the presence of benign traffic and only a specific type of attack. Since tasks vary with the introduction of new attack types, it is essential to consistently retrain the IDS to effectively identify diverse intrusion patterns.

At a high level, CL methods can be broadly classified into three main families [PKP⁺19]: *Replay* methods, *Parameter Isolation* methods, *Regularization*-based methods. Since a complete overview of the CL field is beyond the scope of this work, we limit to describe the last approach which is the most suited for resourced-constrained scenarios. More specifically, the file-rouge of the models of this family is to rely on an extra regularization term introduced in the loss function of each task, in order to consolidate previous knowledge when learning on new data. Among the most commonly-employed methods it is worth mentioning *Elastic Weight Consolidation* [KPR⁺17], which imposes a quadratic penalty to regularise the update of model parameters that were important to previous tasks. The importance of parameters is approximated by the diagonal of the Fisher Information Matrix [KPR⁺17]. More recent and competitive approaches rely on the *Bayesian Inference* framework for mitigating forgetting. In fact, Bayesian Inference offers a straightforward and principled way of implementing CL, as shown in the following.

Bayesian Neural Networks and Continual Learning

In Section 2.2 we discussed several approximate inference methods to approximate the posterior distribution of a NN. One of the most common is represented by *Variational Inference*, which aims to find a tractable approximation to the Bayesian posterior distribution of the weights [BCKW15b].

Here we exploit variational inference to perform Continual Learning, alleviating the catastrophic forgetting. For instance, in [NLB⁺18] the authors propose VCL, a method based on a Bayesian approach to learn parameters of a model, using the previous task’s posterior as the new task’s prior when new data are seen. More specifically, it is possible to approximate each task’s posterior by a variational distribution, $q_t(\mathbf{w}) \sim p_t(\mathbf{w}|\mathcal{D}_{1:t})$ by maximizing the following objective function for every task t [NLB⁺18]:

$$\mathcal{L}_t(q_t(\mathbf{w})) = \sum_{n=1}^{N_t} \mathbb{E}_{q_t(\mathbf{w})}[\log p(\mathbf{y}_n|\mathbf{x}_n, \mathbf{w})] - \text{KL}(q_t(\mathbf{w})||q_{t-1}(\mathbf{w})). \quad (3.4)$$

The first term in Equation (3.4) is the expected log-likelihood of the current model over the data in the current dataset, and it contributes to updating

the model knowledge for the current task. The second term is the Kullback-Leibler (KL)-divergence between the actual and the old variational posterior and penalizes the difference between the current model and the approximate posterior of the previous task. The variational distribution $q_t(\mathbf{w})$ should be chosen to be easy to sample and is typically a diagonal multivariate Gaussian, as used here, so that the loss in Equation (3.4) is minimized with respect to the mean and standard deviations. It is worth noting that this approach has the advantage of not requiring free parameters to be tuned on a validation set (w.r.t. [KPR⁺17] for instance) which can be non-trivial, especially in the considered setting. Last, we emphasized that in contrast to [NLB⁺18] we do not implement the fine-tuning phase for each task with a certain amount of data retained from the previous tasks, the so-called “coresets”, and we solely rely on Equation (3.4), aligning with the considered resource-constrained scenario.

Proposed model architecture

The proposed approach relies on a hierarchical model: first, we use a Bayesian Neural Network (BNN) trained continually as a binary classifier (benign/attack), and then we employ a second generative classifier, trained in a class-incremental manner, to distinguish between known types of attacks. This enables a network administrator to identify the specific type of attack occurring and choose an effective countermeasure, rather than just knowing that an attack is happening.

More specifically, a generative classifier [Bis] aims, firstly, at approximating the class-conditional pdf of the data $p(\mathbf{x}|\mathbf{y})$ for each class by exploiting the training data, and then to rely on Bayes theorem for inferring the conditional distribution $p(\mathbf{y}|\mathbf{x})$ over the classes. The advantage of an even simple generative classifier, like a Quadratic Discriminant Analysis (QDA), is that the parameters of each class-conditional density are estimated independently, meaning that there is no need to retrain the model from scratch when more classes are added. In contrast, in discriminative models (like NNs), all the parameters are entangled, so the whole model must be retrained if new classes are added. Here, we exploit a further simplified version of QDA, by approximating the class-conditioned distributions of the data as diagonal multivariate Gaussian; we refer to this model as Diagonal QDA (DQDA). Under this assumption, we gave that $p(\mathbf{x}|y = c) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_c, \Sigma_c)$ and the posterior over the classes is given by $p(y = c|\mathbf{x}) \propto \pi_c \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_c, \Sigma_c)$, where Σ_c is the diagonal covariance matrix of the data belonging to the class c , and π_c is the prior probability for the c -th class.

It should be noted that a generative classifier may perform poorly if applied to the whole dataset (see Section 3.6.5) but it may provide decent predictive performance on a subset of the data (e.g. represented by only the attack

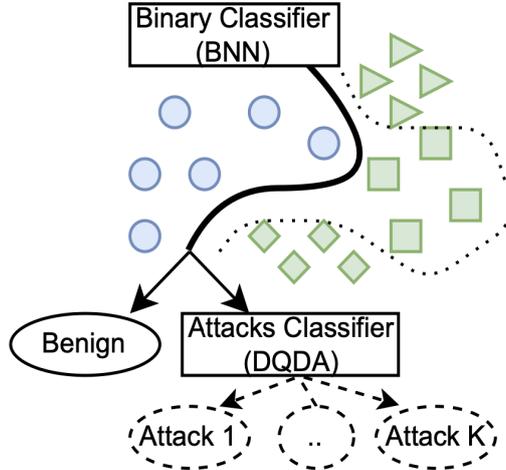


Figure 3.14: A pictorial representation of the proposed hierarchical system. The binary classifier is constituted by a BNN while the multiclass attacks classifier is based on a simple generative classifier.

classes). Moreover, the adopted DQDA classifier has the advantage of being a lightweight model with only $\mathcal{O}(CD)$ parameters, given C classes and D features

Figure 3.14 summarizes the overall workflow at a high level, and in the following we describe the two models in more detail.

The base model adopted here is an MLP composed of two hidden layers of 32 and 16 neurons respectively employing ReLU as activation function. The last layer employs the sigmoid function so that the outputs of the model, given an input, can be interpreted as the probability of being an attack/benign traffic. This particular architecture was chosen to maximize the validation accuracy at a reasonable computational cost, where the training and validation were performed in the standard way, with all the training data present at training time. Then the question is if and how much Continual Learning impacts the overall classification performance. In order to exploit Bayesian Inference in the considered model and employ the previously-described approach for Continual Learning, we relied on TensorFlow probability infrastructure [DLT⁺17b]. More specifically, we exploited Variational Inference with a multivariate diagonal Gaussian distribution as variational distribution. The model is trained for 50 epochs on each task with batches of 128 samples. In this way, it is possible to continuously train a binary classifier for benign/malicious flows, once new attacks are discovered, as previously described.

Regarding the generative classifier we exploited the implementation provided by Scikit-learn [PVG⁺11].

Table 3.9: Samples distribution

Class	Number of samples
Benign	2095057
DoS Hulk	172846
DDoS	128014
PortScan	90694
DoS GoldenEye	10286
FTP-Patator	5931
DoS slowloris	5385
DoS Slowhttptest	5228
SSH-Patator	3219
Bot	1948
Web Attack-Brute Force	1470
Web Attack-XSS	652
Infiltration	36
Web Attack-Sql Injection	21
Heartbleed	11

3.6.4 Dataset Description and Preprocessing

We consider an open-source dataset, i.e., CICIDS2017⁴ [SLG⁺18], to evaluate our proposed system. CICIDS2017 is a labeled per-flow dataset that covers both benign and intrusion traffic, and it is widely used in the literature (e.g. [TLR22a][TJZ⁺21b]). Table 3.9 shows the per-class distribution of samples.

It is clear from Table 3.9 that the dataset exhibits a significant imbalance, with some classes including only a few tens or hundreds of samples. This imbalance poses challenges for both model training and evaluation, particularly for classes that are underrepresented. Consequently, we decided to evaluate our method on the most represented, i.e., the first eight in the Table. This means that we consider seven different tasks, being benign traffic always present. However, we intentionally utilized the underrepresented classes for testing the generalization capabilities of the proposed approach, i.e., in a scenario involving new types of attacks.

The dataset including the eight classes is partitioned into 60% training, 20%, validation, and 20% testing sets. To reduce the unbalance across tasks and classes we randomly subsampled the training data according to the minority class. As a consequence, each task is composed of ~ 3100 benign flows and ~ 3100 samples belonging to an attack class. Finally, data are standardized per-task, ensuring that each feature distribution has a zero mean and unit variance. The parameters for standardization (mean and standard deviation of each feature) are continuously updated on a task basis.

⁴<https://www.unb.ca/cic/datasets/ids-2017.html>

3.6.5 Illustrative Numerical Results

We initially analyze the performance of the proposed approach in binary classification (i.e., benign traffic vs. attack) and later we analyze the performance of the multiclass classifier in recognizing various types of attacks.

Binary classification

We compare our CL-based BNN classifier updated through Variational Inference with two baselines:

- **Full Dataset:** a traditional BNN classifier is trained from scratch on the full dataset including both current and historical data from past tasks.
- **Naïve:** a traditional BNN classifier is re-trained from scratch solely on the data of each task in the standard way. Basically, this training differs from our CL-based approach by neglecting the second term of Equation (3.4). This baseline is useful for assessing the impact of distribution shifts on the traditional BNN classifier.

The main metric to assess the capabilities of the classifiers is the Area Under the ROC curve (*AUROC*) [Bra97] for each model (the higher, the better). In this way it is possible to fairly compare the classifiers in a threshold-independent manner. *AUROC* has also the advantage of being insensitive to the imbalance of classes [Bra97], which makes this metric particularly well-suited to our scenario. We also exploited other common metrics, like the overall accuracy and the F1-Score. All the experiments were repeated 15 times by randomly permuting the order of the task, i.e., by changing the order of appearance of new attacks. Figure 3.15 reports the average evolution of the *AUROC* and Accuracy over different tasks for the considered models. The test set exploited to compute the *AUROC* for each task is composed of the union of all previous tasks' test sets, to check the ability of retaining past knowledge. In general, it is possible to state that the problem of distribution shifts badly affects the performance of an IDS: a classifier trained on a specific task is not able to generalize well to unseen kinds of attacks, despite they belong to the same macro-category (i.e., attack). On the other hand, these plots demonstrate that the proposed strategy based on Bayesian Inference allows the model to retain a certain amount of knowledge related to the older tasks, even though there is a performance gap compared to the model trained with the full dataset. This is also confirmed by the results of Table 3.10, where the performance computed after the last task on the overall test-set is reported. Such a performance gap is expected; however, keeping training data related to older tasks is not feasible in many resource-constrained scenarios, as already thoroughly stressed.

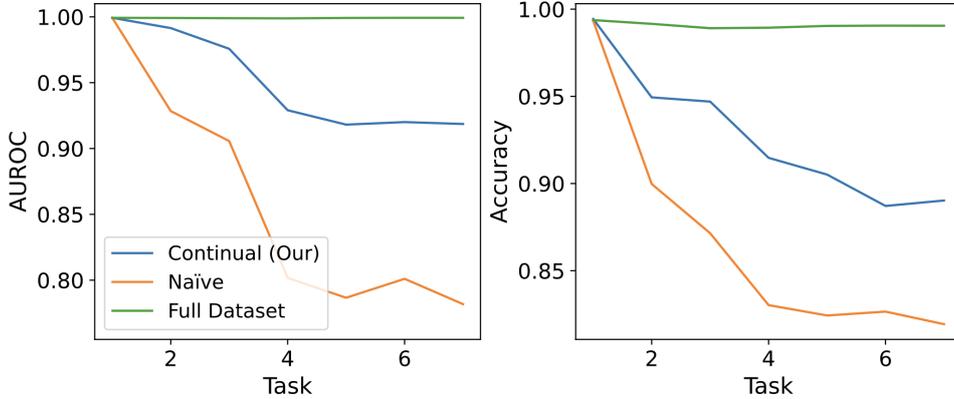


Figure 3.15: Evolution of the AUROC and overall Accuracy over tasks for the different training strategies for the BNN model.

Table 3.10: Binary Classification Metrics

Model	Metric (Mean value \pm Standard Error)			
	AUROC	Accuracy	F1 Weighted	F1 Macro
Continual (Our)	0.93 ± 0.02	0.89 ± 0.02	0.88 ± 0.02	0.85 ± 0.05
Naïve	0.77 ± 0.06	0.82 ± 0.02	0.79 ± 0.02	0.68 ± 0.04
Full Dataset	0.98 ± 0.01	0.98 ± 0.01	0.98 ± 0.01	0.97 ± 0.01

Another aspect we investigated concerns the generalization capabilities of our binary classifier. For this purpose, we provided it with samples belonging to the six unrepresented classes (refer to the bottom of Table 3.3) to simulate a *zero-day-attack* scenario. Our model reaches a True Positive Rate of 0.83 ± 0.03 , the one trained on the Full Dataset 0.86 ± 0.02 and the Naïve model 0.77 ± 0.05 . Thus, Continual Learning has a negligible impact on the generalization capabilities of a baseline BNN in a scenario with zero-days attacks.

Multiclass classification

Table 3.11 provides a summary of the overall performance in multiclass classification. For our model, we divided the analysis into two parts: the first row (*Attacks Only* scenario) reports the metrics computed on samples that are previously correctly classified by the binary classifier. Following that, we present the *Overall* performance of the multiclass classification computed across all classes, including the benign one (i.e., system-level evaluation). For comparison, we also considered DQDA and BNN stand-alone models trained on all available data in the typical static setting (i.e., *Full* dataset), where all classes are present during training. It is interesting to note that the DQDA classifier performs remarkably well for discriminating attacks

Table 3.11: Multiclass Classification Metrics

Scenario	Model	Metric (Mean value \pm Standard Error)		
		Accuracy	F1 Weighted	F1 Macro
Attacks Only	Continual (Our)	0.98 ± 0.01	0.98 ± 0.01	0.99 ± 0.01
Overall	Continual (Our)	0.88 ± 0.02	0.87 ± 0.02	0.84 ± 0.02
	BNN (Full)	0.89 ± 0.01	0.90 ± 0.01	0.69 ± 0.01
	DQDA (Full)	0.52 ± 0.01	0.55 ± 0.01	0.53 ± 0.01

(i.e., as done in our approach) but struggles in a multiclass setting when benign traffic is also present: in this case, the performance drops significantly. This fully justifies our choice of adopting a more flexible classifier (i.e., a BNN) for binary benign-attack classification, and relying on DQDA for the residual complexity of further classifying the attack among different attack types. Overall, we argue that the adoption of Continual Learning and the combination of two simple but specialized models can show decent overall performance compared to the same models trained conventionally, while removing the need of keeping historical data when re-training with new data is performed.

3.6.6 Discussion

In this Section, we introduced a novel approach based on Continual Learning for continuous training of an IDS upon discovering new attacks. Our method leverages the inherent hierarchy of the problem by utilizing a binary BNN continuously updated by means of Variational Inference, and a generative DQDA multiclass classifier only for further attack classification. Numerical results on a real-world dataset demonstrate that our proposed approach effectively manages class-incremental training without the need of storing historical data on past attacks. For this reason, it is well-suited for scenarios with stringent resource constraints, where the demand for thin models and limited storage size of training datasets is prevalent.

One notable limitation of the proposed solution is its inability to automatically recognize (i.e., without an external intervention) new types of attacks. As part of future work, we plan to enhance the model by incorporating Out-of-Distribution detection capabilities.

Chapter 4

Federated ML-Based Intrusion Detection

This Chapter is devoted to the problem of training an ML-based model in a distributed way. As mentioned in Chapter 1, a limitation of current ML-based approaches for developing an IDS is that the model training is typically based on data and computational power elaborated and owned by a centralized node (e.g., a server). As a result, the central server has access to all network traffic generated by the communication among devices participating in the training process, as well as to each device's local data, which could raise privacy concerns. This issue may become even more critical in IoT scenarios, given the volume and sensitivity of data exchanged by certain devices, such as wearable or smart devices. Moreover, different organizations, such as private companies or financial institutions, may be reluctant to share sensitive data about experienced intrusions but remain interested in collaboratively training a common, enhanced model to improve predictive performance.

Given this premise, it would be valuable to explore whether an ML-based IDS could be trained in a distributed way, enhancing privacy and efficiency by keeping data local while still enabling the exchange of knowledge across local nodes. Federated Learning aims to achieve this goal by offering new opportunities for ML algorithms to be applied more efficiently in IDS.

The material covered in this Chapter is based on the following paper:

Contributions

- **J. Talpini**, F. Sartori, M. Savi, *A Clustering Strategy for Enhanced FL-Based Intrusion Detection in IoT Networks*, in International Conference on Agents and Artificial Intelligence (ICAART), Feb. 2023

4.1 FL-based IDS: Related Work

Despite FL has been introduced quite recently, it has received a lot of interest from the research community [LSTS20] and is beginning to attract attention in the field of network intrusion detection. The standard FL workflow is based on the FedAvg algorithm [MMR⁺17], which essentially relies on a weighted average of the models received from each client after local training. A crucial aspect in a FL setting is the statistical heterogeneity of the client datasets, which can be harmful to training performance [TLR22a; SMS20; LSTS20]. In literature, many works can be found that aim at analyzing this issue and propose possible solutions both for anomaly and signature-based IDS.

In the context of anomaly-based IDS, an early contribution is [NMM⁺19] where a hybrid approach, called DioT, was presented as self-learning distributed system for detecting anomalous behaviors in IoT networks. Another relevant contribution is [RTTM20], where the author demonstrated that FL-based models outperform on-device learning models, achieving performance comparable to that of centralized learning.

Regarding the signature-based models, that are the focus of this thesis, several works have proposed to exploit FL to improve IDS.

For instance, [LLL⁺21] leverages FL approach with fog computing, where fog nodes collaborate to detect DDoS attacks. In this approach, the authors employ Gated Recurrent Units as the machine learning technique, with FedAvg used for aggregation. However, these studies do not rely on publicly available datasets so it is challenging to evaluate the effectiveness of their proposed methods, raising concerns about reproducibility and their experimental setting. Specific studies for IoT domains are those of [HYW⁺20], [AMCA20]. However, the authors do not provide details on the implementation or evaluation aspects, such as data distribution and/or number of clients, moreover, they base their analysis on the outdated dataset NSL-KDD, which makes it difficult to assess their approach fully. A relevant contribution to the field is the work of [CSGV⁺21], where authors take explicitly into account the problem of statistical heterogeneity across clients, by evaluating its impact on the FL model (specifically, a logistic regression). As expected the performance is higher for balanced scenarios, obtained by balancing the attacks among all the clients using a simple entropy-based strategy, to ensure that each party trains a local model on the same number of samples for each class. While they perform this balancing operation in a simulated environment by simply re-organizing the overall dataset, in real-world cases their approach would require IoT devices to send portions of traffic to other IoT devices all logically interconnected through a mesh network, which is not ideal as it breaks clients' data privacy. A similar contribution in the field of malware detection is [RSCB22], which also includes the analysis in an

adversarial setup with malicious participants poisoning the federated model. Another contribution is [FFS⁺22], where authors leverage an FL-based intrusion detection system for agricultural IoT infrastructures. A general survey on FL and network security can be found at [FNS22].

As emphasized particularly in [CSGV⁺21], the natural setting for an FL-based IDS is a heterogeneous setting, where data across clients are non iid. In literature, many works can be found that aim at analyzing this issue and propose possible solutions [TLR22a; SMS20; LSTS20; MNB⁺21].

A possible approach to solve the issue consists of learning a personalized model for each FL client [MNB⁺21]. For instance, the MOCHA algorithm proposed in [SCST17] considers a multi-task learning setting and defines a deterministic optimization problem where the correlation matrix of the client is exploited as a regularization term. The contribution of [MNB⁺21] assumes that each local data distribution can be modeled as a mixture of unknown distributions, that provides a principled way to serve personalized models to clients not seen at training time. Another possible approach is to consider FL with heterogeneous data as a meta-learning problem. In this scenario, the goal is to obtain a single global model, and then let each client fine-tune it using its local data [JKRK19].

Lastly, it is possible to tackle this issue by considering a *clustered* FL framework [GCYR20; SMS20]. For instance [SMS20] proposed a centralized clustering algorithm based on the geometric properties of the FL loss surface, so that each client may deploy a more specialized model. The proposed procedure is applied after FL training, once it has converged to a stationary point, which may lead to high computational costs [GCYR20]. Another example is given by [SO21], where K-Means clustering is leveraged for grouping FL clients according to some common features and then collaboratively training an FL model per cluster, rather than deploying a single global model. The paper shows that this leads to an increase in performance in the inference phase.

However, in our context of IDS, it may be unwise to let each client (or a cluster of them) deploy a specialized model since it would be desirable for each of them to be able to recognize also new attacks, seen by other IoT devices. Therefore we leveraged the peculiarities of the considered domain to try to fill this gap by proposing a three-tier clustering architecture where a single global model is trained, and IoT devices are clustered together based on a novel similarity score.

4.2 Proposed approach

4.2.1 Proposed Architecture

Figure 4.1 shows our proposed architecture for FL-based intrusion detection. It consists of a three-tier system including *IoT devices*, *data aggregators* and a *central node*. These components are responsible for any of the main tasks performed by the system: (i) *FL-based model training* (blue arrows in the figure), (ii) *labeled data exchange* (green arrows), and (iii) *IoT devices clustering* (red arrows).

With respect to existing FL-based architectures for intrusion detection [CSGV⁺21], the main difference is the presence of the data aggregators. These nodes act as FL clients and, during the FL-based model training procedure that involves themselves and the central node (i.e., FL model aggregator) executing FedAvg, they use labeled traffic data gathered from a subset of clustered IoT devices, which is opportunely aggregated. Traffic data from any IoT device consists of a set of historical attack/benign traffic samples that should have been collected and labeled by the owner/administrator of such an IoT device.

Unlike [CSGV⁺21], the training data can be balanced among the nodes in charge of model training (i.e., the data aggregators) without the need to exchange part of these data among IoT devices, which is cumbersome in most IoT real-world scenarios and could lead to privacy issues if IoT devices are owned by different parties. In addition, IoT devices are relieved from any training task, which is often very demanding from a computational point of view given their hardware equipment. Then, each data aggregator offloads the trained *FL model* to the IoT devices it is responsible for, so that it can be used for intrusion detection (i.e., inference) on live traffic by the IDS service implemented in any IoT device.

A fundamental aspect of our proposed system is thus how IoT devices are clustered and consequently associated with any data aggregator. This is key to ensure that local FL models are trained on likely balanced datasets so that overall intrusion detection performance is improved. Details on our clustering procedure are provided in the following sub-Section: from an architectural point of view, this procedure is carried out by the central node with some high-level attack information provided by the IoT devices. Such an information, as we will better describe later, includes the statistical distribution of the attacks experienced by the IoT devices in the past.

Once the clusters' composition is computed by the central node, the cluster membership of each IoT device is communicated to its associated data aggregator and to the IoT device itself, so that (i) traffic labeled data can be sent from the IoT device to the designated data aggregator for FL-based model training and (ii) the trained FL model can be offloaded from the

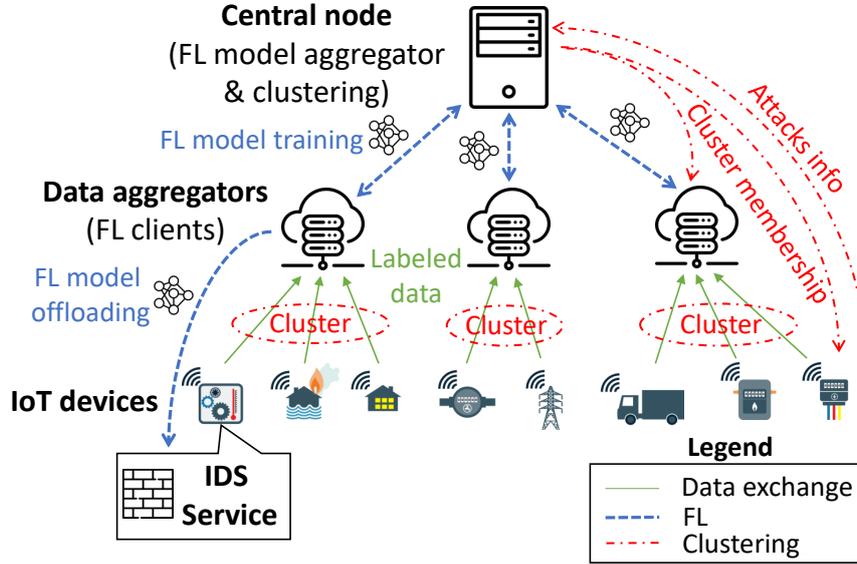


Figure 4.1: FL-based intrusion detection with IoT devices' clustering: system architecture.

designated data aggregator to the IoT device for local intrusion detection.

Our system could be adopted in different scenarios. As a matter of example, let's consider an Internet Service Provider (ISP) that wants to offer an IDS service to its customers, each one owning a bunch of IoT devices at the far edge (e.g. for smart agriculture [SD21]). In this specific case, the data aggregators' logic can be implemented by the edge computing nodes [KAH⁺19] owned by the ISP and disseminated at the edge of its infrastructure. In addition, the central node can be implemented in the cloud datacenter managed by the same ISP. In this specific case, the FL model used for inference by the IoT devices can be trained in the cloud-to-things continuum while ensuring its best possible quality, regardless of how balanced or imbalanced the labeled attack traffic data from any single IoT device is.

Moreover, performing federated learning between edge computing nodes and the central cloud can help reduce the amount of data that needs to be moved around in the ISP network if compared to a *centralized* solution, where model training is fully executed in the cloud [SO21]. In fact, only model weights need to be transferred between edge computing nodes and the cloud, while labeled data related to attacks needs only to be moved from IoT devices to designated edge computing nodes, and is thus kept as much as possible locally. Note also that sharing the labeled attacks' data with edge computing nodes should not be considered a potential source of privacy leakage, as this data is kept within the ISP network domain offering the IDS service.

4.2.2 Proposed Clustering Strategy

To define our strategy, we took inspiration from some findings reported in [CSGV⁺21]. In that paper, the authors showed that FL works well in a *balanced scenario*, when all the clients have the same number of samples for each class.

In fact, this was expected also from the general discussion about Federated Learning of Chapter 2 and in particular from Equation 2.27 that shows that the statistical heterogeneity is detrimental for the generalization capabilities of an FL-based model.

Starting from these premises the main idea of the proposed approach is to estimate the second term of Equation (2.27) (i.e. the discrepancy term) by introducing a proxy for it that depends solely on the label distribution, which is then minimized. This approach focuses solely on the label is motivated by a causal perspective on the data’s generative model. Specifically, let X represent the model inputs and Y the labels to be predicted. Assuming that X causes Y (denoted $X \rightarrow Y$), this is referred to as causal or discriminative prediction [Mur23]. Conversely, if we assume that Y causes X (denoted $Y \rightarrow X$), it is known as anticausal or generative prediction [Mur23]. As an example of the latter case, suppose that X is a medical image and Y represents the true disease state of a patient, as determined by other methods (e.g., a lab test). Here, we have $Y \rightarrow X$ because changes in the disease state will alter the image’s appearance [Mur23]. Similarly, suppose a network traffic trace (X) refers to a malicious or benign activity (Y). In that case, we expect that changing the type of attack will also alter the pattern of network flows, i.e. $Y \rightarrow X$. Therefore, under this assumption, it may be sufficient to balance the label’s distribution across the clients, to reduce the overall discrepancy of the clients’ data distribution and find a setting where standard FL algorithms are expected to perform well.

Here we propose a *clustering strategy* with the aim of automatically choosing an optimal configuration of the clients’ partition in each cluster, by analyzing their label distributions. The aim is to find a meaningful score so that, for a given number of clusters, its minimization results in an effective clusters configuration, which can then be deployed in the three-tier architecture described in the previous sub-section

For each client dataset of size N , given the total number of classes K (i.e., benign and/or attack traffic) and the number of per-class samples n_i , it is possible to compute the normalized Shannon entropy of the labels probability distribution $\mathbf{p} = \{p_1; \dots; p_K\}$ as [Mac95]:

$$\mathbb{H}(\mathbf{p}) = \frac{-\sum_{i=1}^K p_i \ln(p_i)}{\ln(K)} \quad (4.1)$$

where p_i is the probability of a sample belonging to the i -th class to occur, which can be expressed as $p_i = n_i/N$. The Shannon entropy is a measure of the uncertainty associated with a given probability distribution and it is maximized when the distribution is flat [Mac95] (i.e., when all classes have the same number of samples).

This is similar to what was done in [CSGV⁺21]. However, groups with the same entropy may have different label distributions, so it is necessary to take into account also their similarity. For instance, two IoT devices where all the traffic of any of them belongs to one class, which is mutually exclusive with respect to the class of the other device, have the same entropy (equal to zero) but provide substantially different information about the experienced attacks.

Some of the most common choices for measuring the similarity between two probability distributions are: (i) the Kullback-Leibler divergence; its symmetrized extensions, such as (ii) the Jensen and Shannon distance [Lin91], and (iii) the Hellinger distance [Hel09]. Here we decided to exploit the latter because it satisfies some desirable properties: it is symmetric, it satisfies the triangle inequality and it lies in the range $[0, 1]$.

The Hellinger distance between two discrete probability distributions $\mathbf{p} = \{p_1; \dots; p_K\}$ and $\mathbf{q} = \{q_1; \dots; q_K\}$ is defined as:

$$d(\mathbf{p}, \mathbf{q}) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^K (\sqrt{p_i} - \sqrt{q_i})^2}. \quad (4.2)$$

It may be desirable to have clusters where each group has a high entropy and that all group pairs have a low distance between their label distribution. In this way, it is ensured that training data among clusters is balanced as much as possible. Starting from these considerations we define a *similarity score* for ranking the overall balance provided by different groups/clusters¹ configurations. Denoting a clustering configuration of N groups as $\mathbf{R} = \{\mathbf{p}_1; \dots; \mathbf{p}_N\}$, we define the clustering score as:

$$S(\mathbf{R}) = \frac{1}{N} \sum_{i=1}^N \frac{1}{\mathbb{H}(\mathbf{p}_i)} + \frac{1}{2 \binom{N}{2}} \sum_{\substack{i,j=1 \\ i \neq j}}^N d(\mathbf{p}_i, \mathbf{p}_j). \quad (4.3)$$

The first term takes into account the entropy of each group and heavily punishes configurations in which even just one group has low entropy, as such a condition is strongly in contrast with a balanced scenario. The second term describes the dissimilarity of the label distribution between groups. The sum runs over the possible groups' pairs, since the distance is symmetric

¹Here, the terms *groups* and *clusters* are used interchangeably.

we included a factor $1/2$ which accounts for adding each pair of groups twice while the factor $\binom{N}{2}$ represents the number of pairs combinations.

Therefore, for a fixed number of clusters N , the clustering configuration that minimizes S , as defined in Equation (4.3), should be chosen. The number of groups can be considered as a parameter of the proposed clustered federated learning workflow. It has to be properly tuned by taking into account also the physical constraints of the considered network, like the number of aggregators, in order to achieve the best trade-off between possible performance gains obtained by clustering and the amount of devices' traffic data that needs to be aggregated. In Equation (4.3) all the possible pairs of clusters have been considered, which may lead to a computational issue for a large number of clusters. However, it is expected that the best performance is achieved with a few number of clusters, which makes the computation is feasible: this will be evident from the evaluations of the next section.

Dataset Description For exploring the feasibility of the adoption of a federated learning approach for the detection of intrusions in IoT networks, the choice of a realistic dataset plays a crucial role. In [CSGV⁺21] it is possible to find an extensive review of different existing datasets. As done in that paper and given its properties, here we exploit the CIC-ToN-IoT dataset [SLP21a], which is based on the ToN_IoT set [AMT⁺20]. The dataset includes real collected data and is based on 83 features describing the network traffic between different source/destination IP addresses. It is organized *per flow*: each row represents a flow and is annotated as belonging to one over a total of 10 classes, including benign traffic and nine different attacks.

From the dataset, we selected the 16 destination IP addresses, each associated with a different IoT device, with more samples. For each of them, we considered also the flows related to transmitted traffic (i.e., when they act as sources) so that each IoT device's dataset consists of received and sent traffic, mimicking a realistic scenario. After this operation, the overall length of our dataset (not partitioned) was 9849282 samples. We added a binary feature that takes into account the direction of the traffic with respect to an IoT device and we dropped some device-specific features such as the IP address and the flow timestamp. The resulting dataset is unbalanced and class distributions are highly dissimilar across the clients.

Lastly, each IoT device's dataset is divided into train, validation and test set, 60%, 20%, 20% respectively, and standardized using the training data, so that each feature distribution presents a zero mean and unit variance. The validation set is exploited for tuning the hyper-parameters of the classifier and of the federated learning setting (batch size, learning rate, number of local epochs and number of rounds), and the architecture of the classifier. In the next subsections, we recall settings related to the chosen classifier, to the baseline approaches and to our proposed approach.

4.3 Adopted Classifier and Baseline Approaches

As we are interested in characterizing the impact of clustering clients in different ways, we selected a simple classifier, represented by a multi-layer perceptron consisting of an input layer, a single hidden layer with 64 neurons, and a final outer layer. The hidden layer makes use of a hyperbolic tangent activation function while the outer of Softmax. The optimization of each client is performed using the Adam optimizer with an initial learning rate of 0.001 and the cross-entropy as a loss function; each MLP is trained for up to 15 epochs with batches of 512 samples. The federated learning training workflow varies depending on the considered approach, as described below. In every case, we simulated the FL training procedure using TensorFlow Federated, version 0.19.

The first baseline approach is represented by the usual federated learning approach, where the 16 selected IoT devices are the FL clients, which train local models using locally collected data. In this specific case, no data aggregators are included in the related architecture, which only consists of FL clients and of a centralized aggregator. In this scenario, we trained the global model for 200 consecutive rounds using FedAvg as an aggregation function. As no clustering is performed in this case, we will refer to this baseline approach as *no groups*.

The second baseline approach is a centralized scenario where each client transmits its own data to a central server, which thus has a global dataset at its disposal. Then, the classifier is trained simultaneously on the global dataset with the previously described parameters and hyper-parameters. We refer to this second baseline approach as *centralized*.

4.4 Illustrative numerical results

We experimented with different numbers of configurations, with the goal of verifying that our proposed similarity score represents an effective metric for IoT devices' clustering. We considered three *cases*, associated with different N values: (i) $N = 2$, i.e., two clusters with eight IoT devices each, (ii) $N = 4$, i.e., four clusters with four IoT devices each, and (iii) $N = 8$, i.e., eight clusters with two IoT devices each. Given N , the choice of how to partition the IoT devices and assign them to data aggregators follows the scheme reported in Algorithm 1. We randomly chose different partitions of clients obtaining different K realizations of the clusters. For each of them, we computed the score defined in Eq. (5), and then we trained our model in a federated learning way for the best and the worst group realizations according to the score. This procedure is repeated for all the different N values defined above.

Algorithm 3 Clusters selection strategy

Require: Labels distribution of each of the N randomly-generated clusters $\{\mathbf{p}\}_{j=1}^N$
Require: K random realizations $\{\mathbf{R} \equiv [\mathbf{p}_1, \dots, \mathbf{p}_N]\}_{k=1}^K$ of the clusters
Require: Array for the K clusters scores \mathbf{S}
for $k = 1, \dots, K$ **do**
 $S_k = S([\mathbf{p}_1, \dots, \mathbf{p}_N]_k)$ ▷ as in Eq.(4.3)
end for
Let $\mathbf{S}^* = \text{sort}(\mathbf{S})$
 $\mathbf{S}^*[0] \rightarrow$ Best groups realization (\mathbf{R}_{best})
 $\mathbf{S}^*[K] \rightarrow$ Worst groups realization ($\mathbf{R}_{\text{worst}}$)
return $\mathbf{R}_{\text{best}}, \mathbf{R}_{\text{worst}}$

Table 4.1: Test performance for different training group configurations.

Configuration	Similarity Score	F1 Mean	F1 STD
Centralized	-	0.891	0.110
$N = 2$ (2 groups)	2.11	0.859	0.123
	2.51	0.828	0.141
$N = 4$ (4 groups)	2.16	0.856	0.132
	2.68	0.797	0.156
$N = 8$ (8 groups)	2.53	0.814	0.146
	$+\infty$	$+\infty$	0.167
No Groups	$+\infty$	0.730	0.207

For each of the considered cases, we tested the trained classifier by means of FL on each client test set. We considered the *F1-score* as a metric for assessing the performance of the classification on a single client. All the experiments were performed on a Fujitsu workstation with 32 GB of RAM, equipped with an Intel Core i7 and an NVIDIA Quadro P1000 GPU.

In Table 1, for each considered configuration obtained by running Alg. 1 and for the baseline strategies, we report the arithmetic mean of F1-score computed over all the 16 IoT devices, along with the standard deviation (STD). We decided to exploit the arithmetic mean rather than the most commonly used weighted mean (over the number of clients' samples) since a few IoT devices are associated with a very large amount of benign traffic: considering a weighted mean would lead to an overestimated (and thus optimistic) performance.

Moreover, in Fig. 2 we present a graphical representation of the distribution of IoT devices F1-scores through violin plots. Violin plots feature a kernel density estimation of the underlying distribution: the shape of the violin plots represents a visual aid on the behavior of the classifier as a function of the groups' configuration along with the considered baseline

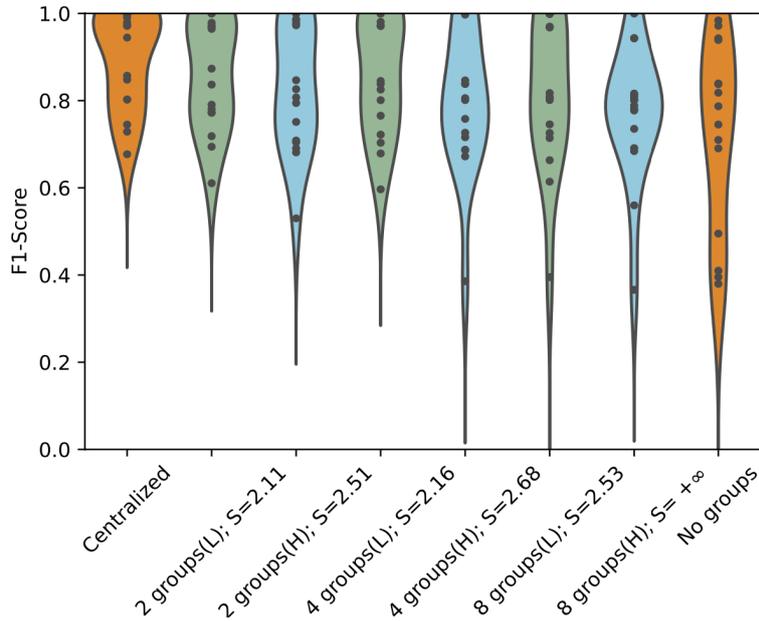


Figure 4.2: Violin plots of test performance (L=Lowest, H=Highest). The y-axis reports the F1-score for each IoT device test set. The violin plots represent a pictorial behavior of the test performance for a given cluster configuration. Violins with denser tails at higher F1-Score are considered better choices.

models. Violins with denser tails represent better stability: in this case, the groups' configuration allows the trained classifier to produce fewer outliers with low F1-score during the test phase.

From Table 1 and Fig. 2 it is possible to notice that our strategy is better than the standard federated learning approach (*no group*) in terms of F1-score, and thus that the proposed score can be effectively used to assess groups unbalance. This is reflected in systematically higher average F1-scores and low STDs for configurations that lead to a lower similarity score, which are configurations with 4 and 2 clusters. In these settings performances are comparable, however, the configuration with four groups should be preferred since it would increase the granularity offered by our system and hence it would reduce the expected computational overhead of aggregators with respect to the configuration with two clusters. The best F1-score is however obtained by *centralized*, as all class samples are present during the training of the global model.

It should also be noted that, in our strategy, the training rounds can be significantly reduced with respect to *no group*. While in *no groups* 200 training rounds were performed, when clustering is adopted we could run the federated learning training for only 90 communication rounds, after which the overall training progress did not improve anymore.

Lastly, we tested our models on new 35 IoT devices' datasets, not included

Table 4.2: Test performance on new 35 clients, for different training group configurations.

Configuration	Similarity Score	F1 Mean	F1 STD
Centralized	-	0.867	0.151
$N = 2$ (2 groups)	2.11	0.849	0.153
	2.51	0.831	0.176
$N = 4$ (4 groups)	2.16	0.853	0.155
	2.68	0.819	0.177
$N = 8$ (8 groups)	2.53	0.827	0.162
	$+\infty$	0.658	0.201
No Groups	$+\infty$	0.621	0.204

in the training phase and randomly chosen from the remaining devices in the CiC-ToN-IoT dataset. This new dataset has an overall length of 175726 samples. Table 2 shows the results for this test, which essentially confirm the performance trends inferred from Table 1. Moreover, it also shows that the proposed approach can train ML models that are able to generalize well to new IoT devices’ data. This is a relevant aspect, especially for practical applications, since it may happen that some IoT devices are not able to share their data labels due to stringent privacy policies or because they have not collected enough historical samples yet.

4.5 Discussion

In this Chapter, a federated-learning-based Intrusion Detection System for IoT networks based on IoT devices’ clustering is presented. In summary, the proposed approach relies on a new clustering similarity score, which is a meaningful quantity for grouping IoT devices and its minimization is beneficial in a FL training scenario both in terms of classification accuracy and number of rounds. More precisely our approach increases intrusion detection performance with respect to a conventional FL approach up to +17% in terms of F1-score with a halved number of training rounds. Moreover, the classifier trained with the proposed framework is able to generalize well to new unseen IoT devices, thus effectively sharing the knowledge of the attacks to new devices without sharing local data.

Here we have exploited a random search for choosing the clustering configuration, as a future work it would be possible to develop a custom algorithm for finding the near-optimal clustering score for a given collection of clients and number of clusters, as long as new tests on different datasets. Moreover, we plan to further investigate the effects of other aggregation algorithms that can overcome FedAvg limitations in unbalanced data scenarios, as well as

specific techniques for dealing with unrepresented attack classes to improve the overall classification performance.

Chapter 5

Towards Trustworthy FL-based Intrusion Detection

The previous section focused on improving the overall predictive performance of ML models in a federated setting. A natural question is how the FL training approach impacts a model’s trustworthiness.

More specifically, in this Chapter, we aim to investigate the calibration properties of a model trained in a federated way and propose a simple way to improve the calibration in this setting by means of a custom calibration module applied after the training of any base classifier.

5.1 On calibrating a FL ML-based IDS

Calibration has been extensively studied in the literature within the traditional centralized setting (see [GPSW17] for a comprehensive overview), as machine learning models, including deep neural networks, are often poorly calibrated. Approaches to address calibration issues can either involve modifications to the model itself or adjustments to the standard loss function (e.g., temperature scaling, focal loss), or they can be applied post-training, assuming the availability of a fresh dataset independent of the training set. In line with the latter approach, several methods exist to improve calibration, including *isotonic regression* and *Platt scaling* [GPSW17].

However, the calibration of machine learning models in federated settings has received limited attention so far. Among the few relevant efforts, recent research [PYTL24] demonstrated that existing federated learning schemes often fail to ensure proper model calibration following weight aggregation, raising concerns about the deployment of FL models in safety-critical domains. To address this, they proposed FedCal, a post-hoc calibrator based on a multi-layer perceptron, with a substantial number of free parameters, which introduces a risk of overfitting and leads to a non-negligible communication

overhead between the clients and the central server.

Similarly, [QMC⁺23] explored the problem of model calibration in FL by proposing a strategy that leverages prototype-based summaries of each client’s data cluster to facilitate effective calibration. However, their approach is designed for a cross-silo FL setting, which permits peer-to-peer communication between clients. In contrast, this work focuses on the cross-device FL scenario, where clients communicate exclusively with a central server.

Another recent study [CHHB24] proposed modifying the local training loss by incorporating a calibration loss to encourage better model calibration. While effective for neural network models, this method is unsuitable when pre-trained classifiers are involved, as envisioned in this manuscript, since it requires access to the training process.

All the previous works are related to model calibration without focusing on any specific application domain. To the best of our knowledge, in the domain of FL-based network intrusion detection, the issue of proper model calibration remains unexplored, and this section presents a simple approach to try to fill this gap.

5.1.1 Proposed Approach

As mentioned earlier, proper calibration is a prerequisite for reliable learning performance. While model trustworthiness is typically studied in centralized settings, here we propose a *federated calibration module* (or *federated calibrator*) that can take as input any pre-trained model. It operates in federated scenarios without requiring modifications to the standard FedAvg [MMR⁺17] aggregation architecture. More specifically, the module adopts a novel approach: not only is the model trained in a federated way, but it is also calibrated in the same manner, leveraging local calibration information from the local clients.

The desired goal of the calibrator module is as follows: given any classifier that outputs predictive scores across C classes $[f_0, \dots, f_{C-1}]$ (i.e., benign or specific attacks) for a given input x (i.e., network traffic sample), establish a meaningful mapping transforming these scores into well-calibrated probabilities. To achieve this goal, we chose to rely on *Platt scaling* – also known as sigmoid or logistic calibration – due to its simplicity and parametric nature that naturally fits the federated learning workflow.

More specifically, suppose a *binary classification* problem and that our base classifier (e.g., a neural network or a decision tree) predicts some score for the 0 -th class, denoted as $f_0(x)$, for a given input x . Platt scaling maps this score to a probability by means of a *sigmoid function* with two learnable parameters, denoted here as a and b , as follows:

$$p(y = 0|f_0(x)) = \text{sigmoid}(a \cdot f_0(x) + b) \tag{5.1}$$

Algorithm 4 Federated Calibrator

Require: Pre-trained classifier, $\tilde{y} = f(x)$, and the calibration module $\hat{y} = \text{calibrator}(\tilde{y}, w)$

Require: For each client k , a fresh calibration dataset $\mathcal{D}_k = \{(x_i, y_i)\}_{i=1}^{N_k}$ and the corresponding predictions provided by the base classifier $\{\tilde{y}_i\}_{i=1}^{N_k}$

Require: M federated training rounds, learning rate η

Server randomly initializes global model weights w^1

for $m = 1, \dots, M$ **do**

for $k = 1, \dots, K$ **do**

$\{\hat{y}_i = \text{calibrator}(f(x_i), w^m)\}_{i=1}^{N_k}$ ▷ calibrator predictions

$\mathcal{L}_k(w) = \sum_{i=1}^{N_k} \text{cross-entropy}[(y_i, \hat{y}_i)]$ ▷ local loss estimation

$w_k^m \leftarrow w_k^m - \eta \nabla \mathcal{L}_k(w)$ ▷ weights update

end for

Server aggregates clients' updates:

$w_{\text{global}}^{m+1} \leftarrow \text{FedAvg}(\{w_k^m, N_k\}_{k=1}^K)$ ▷ Eq. (5.2)

end for

return w_{global}^M ▷ calibrator parameters global estimate

Those parameters are estimated through maximum likelihood on a validation set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ of input-output pairs (x_i and y_i , respectively) that, in this context, we call *calibration dataset*. In the case of a classification task (i.e., the task considered here), the loss function reduces to the usual cross-entropy loss between the predicted probabilities for a given input, and the corresponding class label.

The formulation of Equation (5.1) can be easily extended to a *multi-class classification* task by applying the same equation to each class independently, and then normalizing the output to ensure a valid probability interpretation. The great advantage of Platt scaling is that it has only a few parameters (as it scales linearly with the number of classes) making it suitable even for calibration datasets of small size. Additionally, it is versatile and, as already pointed out, it can be applied to any pre-trained model.

Our simple yet effective proposal is to *train this calibrator following the standard federated learning pipeline*, e.g. by adopting FedAvg [MMR⁺17] in a centralized server. Specifically, the server (i.e., FL model aggregator) sends to the clients (or to a random subset of them) a pre-defined calibrator model (in terms of parameters a and b of the sigmoid function) for local training. Each client trains the given model with its own data, by minimizing a given loss function (i.e., cross-entropy) and hence computing a local update of the weights, here denoted as $w \equiv [a, b]$ following Equation (5.1).

At the end of each round, the server collects all local updates related to a and b and combines them to update the central model parameters. Following this iterative process, an arbitrary number of clients can concur to model calibration without transferring the collected data (i.e., calibration dataset) to

a centralized location, since only locally-trained sigmoid function parameters need to be sent. As said, the common aggregation function adopted here for federated calibration is FedAvg, which computes the globally updated weights w_{global} as a weighted average over all clients’ weights w_k .

$$w_{\text{global}} = \frac{1}{\sum_{k=1}^K n_k} \cdot \sum_{k=1}^K n_k w_k. \quad (5.2)$$

Overall, the proposed calibration workflow is coordinated by the central server. It takes as input a pre-trained *base model* and provides, as output, a calibrated version of it by executing the federated calibration module and the related workflow, whose details are reported in Algorithm 4.

Note that the base model may have been trained following any possible training paradigm. However, given its peculiarities, our calibration procedure is especially recommended in the case of a model trained by means of federated learning, as it follows the same workflow. Specifically, the model could first be trained using FL, and then calibrated giving it as input to our federated calibration module.

5.1.2 Experimental Setting

As a base model, we considered an extreme gradient-boosted decision tree classifier implemented in the library *XGBoost* [CG16]. This model often outperforms more complex models (e.g., deep learning models) on tabular data such as the one considered here [SZA22; MLBP22], being very appealing in the case of network traffic classification.

Each client locally trains its own XGBoost model, then the central server aggregates the local models to define a single global model following the federated learning schema. Boosted decision trees are aggregated as follows: at each FL round, each client sends its boosted tree to a central server; the server then treats these trees as bootstrapped versions of a classifier and combines them into an ensemble to create the final model. So, if we have K clients and M rounds, the final classifier will consist of $K \cdot M$ trees. The model aggregation procedure is performed exploiting the Flower library [BTM⁺20], and the federated training of the three-based model required 50 training rounds and 10 local epochs per round.

Regarding the calibration module, Platt scaling has been implemented through Pytorch [P⁺19] as a fully-connected neural network without hidden layers, with a diagonal weight matrix and with a sigmoid (i.e., the Platt scaling sigmoid of Equation (5.1)) as an activation function. This allowed an easy implementation of the calibration module directly in the Flower workflow. The calibrator’s FL training rounds took 20 rounds and 10 local epochs per round.

To assess the impact of federated learning on the calibration properties of a base model and evaluate the effectiveness of our proposed approach, we considered the following baselines:

- **Base model.** This is the classifier obtained in a federated way without any calibration module.
- **Centralized calibration.** It represents the favorable scenario, where the central server has access to a calibration set so that the calibration can be performed in the usual centralized setting. However, this raises concerns about user privacy since data a portion of data are shared with the central server.
- **Local calibration.** Another potential approach is personalized calibration, where each client independently trains its calibrator using its own local data.
- **Proposed approach.** The previously described federated calibration.

The dataset exploited is NF-ToN-IoT, and we applied the preprocessing steps described in the previous chapter. However, we partitioned the clients differently from the previous experiments, to study the effect of statistical heterogeneity better.

To distribute data among clients we exploited a common partition method used in the literature, which is based on a symmetrical Dirichlet distribution, governed by a concentration parameter α . Such a parameter can be used to indicate the *statistical heterogeneity*, where lower values of α (say < 1) lead to more heterogeneous settings (i.e., more different sample distributions across classes for the different clients). In particular, we considered 10 clients with varying α between 0.2 and 0.9. It is worth mentioning that in a network intrusion detection scenario it is common to have statistical heterogeneity [CSGV⁺21] and therefore we concentrated our analysis on a heterogeneous scenario, i.e., with $\alpha < 1$.

Moreover, to leverage the simplicity of the calibration module, which allows training the calibration with a small amount of data, we applied a random sub-sampling of the classes except for the minority one, so that all the classes are equally represented at calibration time.

5.1.3 Illustrative numerical results

As evaluation metrics, we considered Expected Calibration Error (ECE), where lower values indicate better calibration, along with standard predictive performance metrics such as accuracy and F1 score. The objective is to minimize the ECE of the base classifier, bringing it closer to that of a centralized setting, without adversely impacting classification performance.

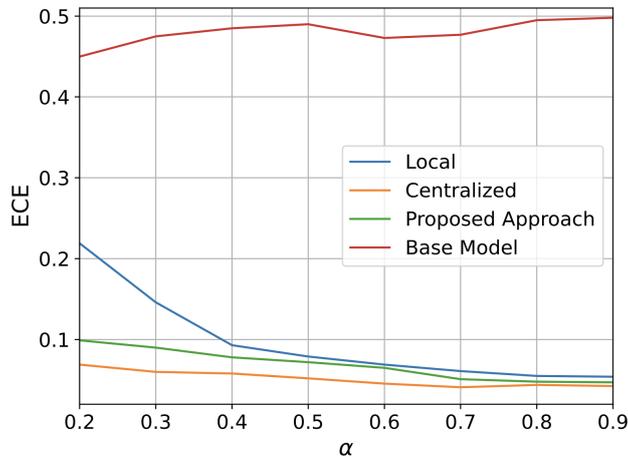


Figure 5.1: Expected Calibration Error for the considered models as a function of the statistical heterogeneity. Curves represent the mean ECE obtained over 10 different experiments with varying seeds.

Figure 5.1 shows the ECE as a function of the heterogeneity parameter α for the considered baselines. It can be seen that the base model is highly uncalibrated for all the values of α , thus raising concerns about the reliability of standard FL-based models. On the other hand, we can see that the proposed approach shows a comparable behaviour to centralized calibration, but without the need of a centralized calibration dataset, thus guaranteeing privacy and efficiency since data remains local.

Moreover, the proposed approach leads to better calibration with respect to a personalized, local calibration. The difference, as expected, is higher in more heterogeneous settings, where local data are not representative of the overall population as typically happens in a network intrusion detection scenario. In fact, for instance, certain regions may be more exposed to certain cyber attacks than others, leading to imbalances in the data collected from each client. The ability of the model to perform well under these conditions is important to ensure its effectiveness in practical, large-scale intrusion detection systems.

It is worth noting that the calibration module does not negatively impact on the classification performance, as shown in Figure 5.2, for both accuracy and F1-macro. Here, we can see that our approach guarantees similar classification performance to the centralized calibration case and to the base model. Again, local calibration shows a sub-optimal behavior, especially in more heterogeneous settings. In addition, similar results are obtained for the *Weighted F1-score*, with even closer performance between our approach, centralized calibration and base model. However, we decided not to report them for the sake of conciseness.

Finally, to further investigate the relationship between ECE and class

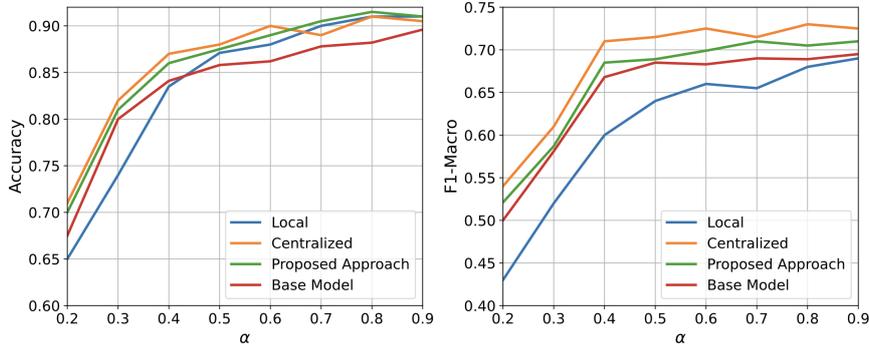


Figure 5.2: Accuracy and Macro F1-Score for the considered models as a function of the statistical heterogeneity. Curves represent the mean ECE obtained over 10 different experiments with varying seeds.

imbalance, we report in Figure 5.3 the per-class ECE, with classes sorted in decreasing order of sample size. The results highlight that minority classes (e.g., ransomware, mitm, etc.) exhibit significantly higher ECE values, indicating a stronger calibration challenge in these cases for all the baselines, showing how Platt scaling struggles in performing its task.

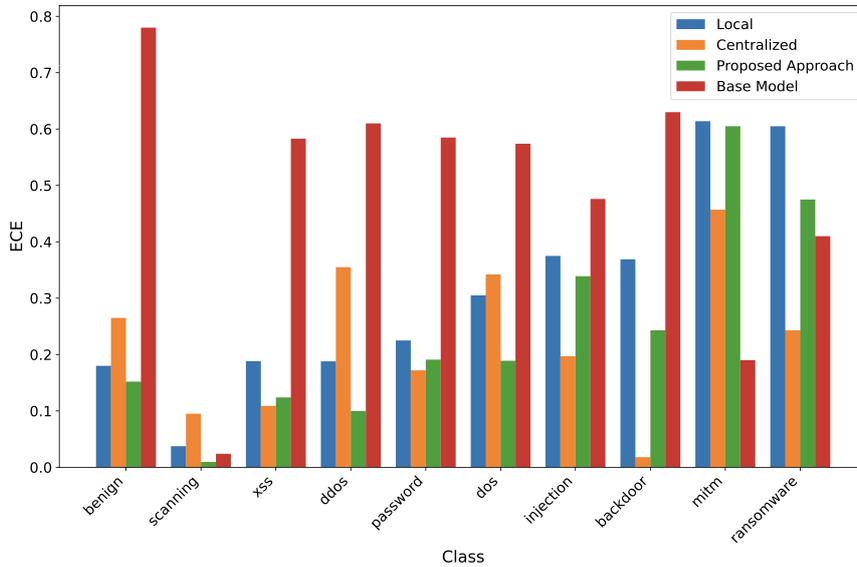


Figure 5.3: ECE per class for the evaluated models at a fixed heterogeneity $\alpha = 0.3$. The classes are ordered in decreasing order based on their sample size.

5.1.4 Discussion

The proposed module demonstrates good calibration performance at small α values, i.e., with heterogeneous client’s data, and maintains comparable per-

formance to server-based centralized calibration as α increases (i.e., $\alpha > 0.4$), while offering advantages in terms of privacy, as no calibration data must be shared with the server. At the same time, our federated calibration outperforms a local calibration strategy, where each client calibrates separately the base model (e.g. trained by means of FL): separate calibration steps at different nodes might leverage partially representative data and, hence, result in non-trustworthy models.

In addition, the classification performance in terms of accuracy and F1-score of our proposed approach is not affected with respect to the base model, and it is comparable to that of a model obtained by performing centralized calibration.

These features make our approach particularly well-suited for the deployment of reliable ML-based Intrusion Detection Systems, where data are typically unbalanced and where privacy and efficient resource usage are essential. However, it still faces challenges to work well with under-represented classes, highlighting an area for potential improvements. In addition, for future works, we plan to investigate other approaches to calibration in a federated learning setting, like isotonic regression and/or methods based on the conformal prediction framework.

5.2 Towards One-Shot FL

The materials presented here are part of an ongoing project initiated during my research period abroad at Inria Centre at Université Côte d’Azur. The main motivation was to define a way to develop a Bayesian model in a federated way, to exploit the proprieties of these models discussed in Chapter 3. However, we recognized that our work addressed a broader gap in the machine learning literature. As a result, our efforts shifted toward making a methodological contribution to the field of machine learning, extending beyond the specific application to network intrusion detection.

5.2.1 Introduction and Related Work

In the previous Section, we discussed the problem of calibrating a pre-trained classifier with a federated learning setting. Here, we focus on showing *how* Bayesian Inference can naturally tackle the problem of combining models trained in a distributed setting. In particular, we will show how a Bayesian approach can tackle the problem in a *one-shot* approach, i.e., in one single round of communication between the central server and the clients.

The one-shot FL is raising attention from the research community since it promises to solve some challenges associated with the usual multi-round setting [JWJ24]. These drawbacks include: a frequent connection with the central server, the need for the clients to repetitively train the model, as long as new updates are received from the server, and lastly the continuous exchange of information between clients and the central server enlarges the attack surface [GTS19].

In the literature, few approaches address the problem of enabling FL in a single communication round. These methods can be broadly categorized into two groups. The first category is based on *knowledge distillation*, while the second group is based on *neuron matching*.

The methods falling in the first group view the client models as an ensemble and the overall idea is to try to distill the knowledge from this ensemble into a single global model. To accomplish that, some works assume the existence of a dataset at the server (e.g., [GSK⁺21; LKSJ20]) while others mimic this scenario by training generative models, like GAN or VAE, to synthesize an auxiliary dataset at the server side (e.g., [ZJP⁺20; ZPM⁺20; HLRS23]). The availability of a dataset (both real or synthesized) raises, however, privacy concerns, while requiring an increased computation cost on the server and require careful hyperparameter tuning, which is itself a challenge to implement in one-shot FL [JWJ24].

On the other hand, methods based on neuron matching rely on the invariance of Neural Networks with respect to permutations of the neurons. Some work leverage this property, by first aligning the weights of the client

models according to a common ordering (called matching) and then average the aligned client models (e.g. [SJ20; LLW⁺22; JSS⁺]). Although this approach has been shown to work well when combining simple models like feedforward NNs, the performance drops considerably for more complex models such as CNNs [JWJ24].

There is also another line of works, referred to as *model fusion*, intending to fuse the capabilities of multiple existing models into a single model (e.g., [MR22; YTC⁺23; JRPPC]). These approaches were not specifically designed for one-shot FL, instead, they mostly used pre-trained models without considering the effect of data heterogeneity during their training. A relevant contribution to the field, in line with our work is [JWJ24]. Here, the author proposes a one-shot FL approach based on the Fisher information matrix. Each client computes a local estimate of the model weights by minimizing the local loss function as well as its curvature around the minimum, and then all this information is sent to the server. The server aggregates the received local loss estimates, approximated as a quadratic function, by summing them and employs SGD to find the global minimum. Additionally, a regularization term is included in the server loss to prevent SGD from drifting too far from the mean of the local model parameters.

5.2.2 Problem Setting

In Section 2.4 we presented the standard approach to FL, based on statistical learning theory. Here we tackle the same problem from a probabilistic perspective. The setting is the following: we have a collection of C distributed dataset of input-outputs pairs $\mathcal{D}_c = \{(\mathbf{y}_i, \mathbf{x}_i)\}_{i=1}^{N_c}$ and we want to fit a parametric function $\mathbf{y} = f(\mathbf{x}, \mathbf{w})$ in a distributed way. In other words, we aim to combine the models trained on each \mathcal{D}_c so that the resulting model is equivalent to the one trained on $\mathcal{D} = \cup \mathcal{D}_c$.

The starting point is the likelihood function $p(\mathcal{D}|\mathbf{w}) = p(\mathcal{D}_1, \dots, \mathcal{D}_C|\mathbf{w})$. If we assume that the clients' data are independent, given the model, we can factorize the likelihood function as follows:

$$p(\mathcal{D}_1, \dots, \mathcal{D}_C|\mathbf{w}) = \prod_{c=1}^C p(\mathcal{D}_c|\mathbf{w}) \quad (5.3)$$

this result has already been exploited in the literature, e.g. [JWJ24; AMCA20].

However, while those works focus solely on inferring the maximum of the posterior, providing a *unimodal* approximation of the local loss, we propose a *multimodal* method to approximate the entire posterior, capturing its different modes and fully embracing a Bayesian approach.

To achieve this, we need a principled approach to combine the different client posteriors, starting from the following lemma:

Proposition 5.2.1 (Global Posterior). *Given C datasets and the posteriors $p(\mathbf{w}|\mathcal{D}_c)$ of the same model $f(\mathbf{w})$. Under the assumptions: i) the function $f(\mathbf{w})$ can fit $\mathcal{D} = \cup\mathcal{D}_c$, ii) the datasets are conditionally independent, given the model, iii) the prior is the same across clients; the posterior can be written as:*

$$p(\mathbf{w}|\mathcal{D}_1, \dots, \mathcal{D}_C) = \frac{1}{p(\mathcal{D}_1, \dots, \mathcal{D}_C)} \cdot \frac{\prod_{c=1}^C p(\mathbf{w}|\mathcal{D}_c)}{p(\mathbf{w})^{C-1}} \quad (5.4)$$

The first term of the right-hand side is a normalizing constant and does not depend explicitly on the model parameters, while the factor $p(\mathbf{w})^{C-1}$ avoids the over-counting of the prior. We emphasize that this result allows for the exact combination of distributed inference outcomes, without any loss of information. In other words, combining the posteriors via Equation (5.4) provides the same information about the model as if it had been trained in the usual centralized setting, by combining directly the clients' data.

If we are interested in finding the maximum of the global posterior (e.g., to find the ‘‘optimal’’ global model) we can rely on the log-posterior and drop the terms that do not depend on \mathbf{w} , resulting in the following loss:

$$\mathcal{L}(\mathbf{w}) = \sum_{c=1}^C \log(p(\mathbf{w}|\mathcal{D}_c)) - (C - 1)\log(p(\mathbf{w})) \quad (5.5)$$

It is worth mentioning that under the assumption of an (improper) flat prior the obtained results reduce to what has been presented in [AMCA20] and the starting point of [JWJ24]

5.2.3 Proposed Approach

As mentioned, we aim to account for the different modes of the posterior. To achieve this, we employed a mixture of Gaussian distributions to approximate the local posterior pdf of each client. More specifically, we are exploiting a mixture of Laplace approximations (see Section 2.2) of independently trained deep neural networks [EDHK21]. The rationale behind this choice lies in combining a lightweight approach to describe the local curvature of the loss with a global perspective provided by the different ensembles. More concretely each client train M times the same model on the same data but varying randomly the starting point of the gradient descent algorithm, exploited for the training. By adopting this method, it follows that the overall posterior at the server will be a mixture of Gaussian and, in general, multi-modal. More precisely, starting from Equation (5.4), the posterior for C clients and M mixture can then be written as:

$$p(\mathbf{w}|\hat{\mathbf{w}}_{1:M}^{1:C}, \Lambda_{1:M}^{1:C}, \mathcal{D}) \propto \prod_{c=1}^C \sum_{m=1}^M \frac{1}{M} \mathcal{N}(\mathbf{w}|\hat{\mathbf{w}}_m^c, \Lambda_m^c, \mathcal{D}_c) \cdot \frac{1}{p(\mathbf{w})^{C-1}} \quad (5.6)$$

where: $\hat{\mathbf{w}}_m^c$ is the m -th maximum-a-posteriori estimate of the c -th client and similarly Λ_m^c is the precision matrix. The server is responsible for computing this expression based on the Gaussian mixtures received from each client.

As mentioned earlier, the posterior is generally multi-modal, indicating that it is possible to find different and good parameter settings that can fit the data well. Therefore, it may be advantageous to consider a mixture of these solutions and use all of them to make predictions, in line with a *Bayesian Model Averaging* perspective, described in Section 2.2. This ensemble represents the final model that the server sends back to the clients.

To find the different modes of the global posterior, the server adopts the same approach as each client by running SGD M times on the approximate global log-posterior, initializing from the median of the different MAP solutions obtained from the clients' mixtures. The details of the previously described procedure are given in the pseudocode presented in Algorithm 5.

We stress again the mixtures model arising from the necessity of capturing different modes of the loss function, with the overall aim of approximating a Bayesian Model Averaging, as in Equation (2.17).

5.2.4 Experimental Setting

To assess the performance of the proposed approach and to compare it with other baselines we conducted several experiments on standard benchmarks and settings common in the literature (see [JWJ24]). We considered the following datasets:

- FashionMNIST [XRV17]. It is a dataset of Zalando's article images, consisting of a training set of 60.000 samples and a test set of 10.000 instances. Each sample is a 28×28 grayscale image, associated with a label from 10 classes. Here we refer to this dataset as FMNIST.
- SVHN [NWC⁺11]. SVHN is a real-world image dataset for developing machine learning and object recognition algorithms obtained from house numbers in Google Street View images. It can be seen as similar in flavor to MNIST (e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over 600.000 digit images) and comes from a significantly harder, real-world problem. Each sample is associated with a label from 10 classes.
- CIFAR10 [KNH10]. It consists of 60.000 32×32 color images in 10

Algorithm 5 Bayesian One-Shot FL

Require: Data $\mathcal{D}_{1:C}$ (data of each client); number of mixtures M ; client and server learning rates η_c, η_s and number of iterations K_c, K_s , respectively.

- 1: **Server** randomly initializes M global model weights $\mathbf{w}_{1:M}^0$
- 2: **Server** selects a subset C_t of clients
- 3: **Server** broadcasts global models weights $\mathbf{w}_{1:M}^0$ to selected clients C_t
- 4: **for** each client c in C_t **in parallel do**
- 5: $\hat{\mathbf{w}}_{1:M}^{(c)}, \Lambda_{1:M}^{(c)} \leftarrow \text{ClientTraining}(\mathbf{w}_{1:M}, \mathcal{D}_c, K_c)$
- 6: **Client** c sends updated mixtures weights and precision matrices $\hat{\mathbf{w}}_{1:M}^{(c)}, \Lambda_{1:M}^{(c)}$ to the server
- 7: **end for**
- 8: **Server** aggregates clients' updates to infer global mixture parameters:
- 9: $\hat{\mathbf{w}}_{1:M}^{\text{global}} \leftarrow \text{ServerTraining}(\hat{\mathbf{w}}_{1:M}^{1:C}, \Lambda_{1:M}^{1:C})$
- 10: **Clients** receive $\hat{\mathbf{w}}_{1:M}^{\text{global}}$ and make predictions by ensembling:
$$p(\mathbf{y}|\mathbf{x}, \mathcal{D}_{1:C}) = \frac{1}{M} \sum_{i=1}^M p(\mathbf{y}|\mathbf{x}, \hat{\mathbf{w}}_i^{\text{global}})$$

- 11: **function** CLIENTTRAINING($\mathbf{w}_{1:M}^0, \mathcal{D}_c, K$)
 - 12: **for** each mixture model m **do**
 - 13: **for** $k = 0, \dots, K_c - 1$ **do** iterations
 - 14: $\mathbf{w}_m^{k+1} \leftarrow \mathbf{w}_m^k - \eta_c \sum_{i \in \mathcal{I}} \nabla \ell(\mathbf{w}_m^k; \mathbf{x}_i, y_i)$
 - 15: **end for**
 - 16: Set $\hat{\mathbf{w}}_m = \mathbf{w}_m^{K-1}$
 - 17: Compute precision matrix of Laplace approximations Λ_m
 - 18: **end for**
 - 19: **return** $\hat{\mathbf{w}}_{1:M}, \Lambda_{1:M}$
 - 20: **end function**
 - 21: **function** SERVERTRAINING($\hat{\mathbf{w}}_{1:M}^{1:C}, \Lambda_{1:M}^{1:C}, K_s$)
 - 22: Global loss $\mathcal{L}(\mathbf{w}|\hat{\mathbf{w}}_{1:M}^{1:C}, \Lambda_{1:M}^{1:C})$ computation, from Equation (5.6)
 - 23: Starting points computation $\mathbf{w}_{1:M}^0 = \text{median}_c(\hat{\mathbf{w}}_{1:M}^{1:C})$
 - 24: **for** each mixture model m **do**
 - 25: **for** $k = 0, \dots, K_s - 1$ iterations **do**
 - 26: $\mathbf{w}_m^{k+1} \leftarrow \mathbf{w}_m^k - \eta_s \nabla \mathcal{L}(\mathbf{w}_m^k)$
 - 27: **end for**
 - 28: Set $\hat{\mathbf{w}}_m^{\text{global}} = \mathbf{w}_m^{K-1}$
 - 29: **end for**
 - 30: **return** $\hat{\mathbf{w}}_{1:M}^{\text{global}}$
 - 31: **end function**
-

different classes (e.g., airplane, dog, truck..), with 6000 images per class. There are 50000 training images and 10000 test images.

Regarding the models, as in the literature, we exploited two CNN: LeNet [LBBH98] for FashionMNIST, and a custom CNN [WLL⁺20] for the other two datasets.

We exploited the open-source library [DKI⁺21] to perform Laplace approximation of the local posterior. More specifically, we exploited a diagonal approximation for all parameters except the parameter belonging to the last layer for which we employed a full precision matrix. For all the experiments we chose a Gaussian diagonal prior with variance $\sigma^2 = 0.1$. Moreover, we employed a tempered posterior for each local model by raising the posterior to the power $1/T$: $p_{temp}(\mathbf{w}|\mathcal{D}) = p(\mathbf{w}|D)^{1/T}$ with a parameter $T = 0.1$. This choice leads to a reduction of the uncertainty associated with each parameter. This plays the role of a regularizer that improves our approximation of the true Laplace by mitigating the overestimation of variance in certain directions, which occurs when covariances between layers are ignored [RBB18]. Additionally, it enhances the Laplace approximation itself by preventing it from placing too much probability mass in low-probability regions of the true posterior [RBB18].

We compared our approach with the following state-of-the-art (SOTA) approaches:

- Fedfisher [JWJ24]. As we briefly discuss in the related work section, this is a SOTA approach based on an unimodal approximation of the loss function as a quadratic function. The Fisher matrix has been approximated through Kronecker factorization, which gives better results with respect to a diagonal approximation. Here, we exploit the Kronecker approximation, referred to as FedFisherKFAC.
- REGMEAN [JRPPC]. It is a SOTA model fusion approach that merges models in their parameter space, guided by weights that minimize prediction differences between the merged model and the individual models.
- DENSE [ZCL⁺22]. It is a SOTA approach for one-shot FL-based that relies on data-free knowledge distillation by exploiting GANs to artificially generate data for distillation at the server.
- OTFUSION [SJ20]. It is a popular neuron-matching method that utilizes optimal transport to align neurons across the models averaging their associated parameters, rather than directly averaging the weights.
- FISHER MERGE [MR22]. It is a model fusion baseline that similarly

to our contribution employs a diagonal approximation to the hessian of the log-posterior.

To simulate data heterogeneity among C client datasets, we partition the original image dataset into C subsets using a symmetric Dirichlet sampling procedure with parameter α [HQB19]. A smaller value of α results in a more heterogeneous data split. For LeNet each client training last 20 epochs while for the CNN on SVHN and CIFAR10 50 epochs with SGD as optimizer. The server performs 900 steps for minimizing the global loss through the Adam optimizer with standard default hyperparameters.

5.2.5 Illustrative Numerical Results

In this section, we compare the different methods on the aforementioned dataset, by varying the heterogeneity parameter. In all the settings we used the same parameter (e.g, base models and number of local epochs). To allow for a fair comparison between us and FedFisher we excluded the availability of a validation dataset used originally to measure the validation performance after every 100 steps and use the model which achieves the best validation performance as the final FedFisher model.

Table 5.1 shows the average accuracy for our approach for a varying number of mixtures, from 1 to 5, moreover, we also reported the result obtained with FedFisher for a quick comparison. Results are obtained over 5 experiments with different random seeds and a fixed number of clients, equal to 5. As expected, performance improves in more homogeneous scenarios and with an increase in the number of mixture components across all experimental settings.

In Table 5.2 we repeat a similar experiment but with 10 clients and at fixed heterogeneity parameter $\alpha = 0.3$ to gain insights on the behavior of the proposed approach with more clients. It is worth noting that, in this scenario, each client dataset is smaller, which increases the risk of local models overfitting and makes the aggregation procedure more challenging. Again, we observe an improvement with an increasing number of mixture components.

Table 5.3 shows the average accuracy for our method with a fixed number of mixtures, $N = 5$, along with the other considered baselines. Again, results are obtained over 5 experiments with different random seeds and a fixed number of clients, equal to 5. Additionally, we report the mean accuracy across datasets for each model and experimental setting to facilitate a clearer comparison. We repeated the same procedure also with a larger of clients and Table 5.4 summarizes the numerical results.

Overall, our proposed approach shows decent performance, comparable and sometimes outperforming other state-of-the-art models, with up to $\sim 10\%$

Table 5.1: Test set performance metrics, 5 clients

Mean Accuracy & STD (5 seed)						
Dataset	N=1	N=2	N=3	N=4	N=5	FedFisher KFAC
$\alpha = 0.05$						
FMNIST	45.50 ± 2.03	47.77 ± 1.84	47.54 ± 1.62	51.60 ± 1.448	51.77 ± 1.84	50.92 ± 1.87
SVHN	49.46 ± 1.72	49.22 ± 2.33	48.80 ± 2.57	50.17 ± 1.41	51.39 ± 0.90	51.31 ± 4.49
CIFAR10	26.25 ± 2.67	30.79 ± 3.13	31.40 ± 3.92	32.78 ± 2.11	32.93 ± 3.68	36.73 ± 2.08
$\alpha = 0.1$						
FMNIST	60.78 ± 2.20	65.98 ± 1.44	68.35 ± 0.56	66.96 ± 0.89	69.51 ± 0.78	64.46 ± 1.64
SVHN	60.15 ± 1.80	59.94 ± 1.54	60.77 ± 1.51	62.50 ± 1.40	62.87 ± 2.06	58.59 ± 1.19
CIFAR10	33.99 ± 3.18	37.65 ± 2.26	41.24 ± 1.19	41.38 ± 3.35	41.84 ± 2.54	45.92 ± 1.47
$\alpha = 0.2$						
FMNIST	67.73 ± 2.61	70.16 ± 1.44	73.35 ± 1.09	74.37 ± 0.66	74.63 ± 0.81	70.95 ± 1.19
SVHN	72.16 ± 2.41	73.82 ± 1.48	76.14 ± 0.61	75.59 ± 2.81	77.69 ± 0.19	69.61 ± 5.14
CIFAR10	44.45 ± 1.53	46.06 ± 1.15	46.04 ± 1.49	48.92 ± 2.30	49.12 ± 1.15	49.65 ± 2.51

Table 5.2: Test set performance metrics, 10 clients

Mean Accuracy & Standard Error (5 seed)					
Dataset	N=1	N=2	N=3	N=4	FedFisher KFAC
$\alpha = 0.3$					
FashionMNIST	71.81 ± 3.79	73.94 ± 2.21	77.56 ± 1.96	76.58 ± 1.41	74.83 ± 3.10
SVHN	73.26 ± 1.81	71.90 ± 1.45	75.19 ± 0.81	75.43 ± 1.89	73.7 ± 0.31
CIFAR10	41.57 ± 1.29	42.29 ± 3.84	47.86 ± 3.05	47.92 ± 2.51	50.47 ± 2.23

Table 5.3: Test set performance metrics, 5 clients

Mean Accuracy & STD (5 seed)						
Dataset	N=5 (Our)	FedFisher KFAC	REGMEAN	DENSE	OTFUSION	FISHER MERGE
$\alpha = 0.05$						
FMNIST	51.77 ± 1.84	50.92 ± 1.87	48.04 ± 4.18	38.48 ± 5.56	37.26 ± 1.01	43.11 ± 6.18
SVHN	51.39 ± 0.90	51.31 ± 4.49	65.57 ± 1.83	56.89 ± 0.75	39.17 ± 0.42	43.53 ± 2.54
CIFAR10	32.93 ± 3.68	36.73 ± 2.08	33.77 ± 0.52	30.61 ± 1.95	29.59 ± 1.77	28.44 ± 3.37
Average	45.36	46.33	49.13	41.99	35.33	38.36
$\alpha = 0.1$						
FMNIST	69.51 ± 0.87	64.46 ± 1.64	56.40 ± 5.54	47.16 ± 1.73	43.54 ± 2.43	57.38 ± 5.66
SVHN	62.87 ± 2.06	58.59 ± 1.19	71.01 ± 1.29	62.65 ± 0.91	51.94 ± 0.67	52.66 ± 4.04
CIFAR10	41.84 ± 2.54	45.92 ± 1.47	40.86 ± 0.77	38.04 ± 1.78	40.41 ± 0.73	36.56 ± 1.51
Average	58.07	56.29	56.09	49.32	45.30	48.87
$\alpha = 0.2$						
FMNIST	74.63 ± 0.81	70.95 ± 1.19	71.43 ± 1.6	69.07 ± 4.23	57.06 ± 2.19	66.89 ± 3.50
SVHN	77.69 ± 0.19	69.61 ± 5.14	78.24 ± 0.81	77.13 ± 3.39	72.19 ± 1.00	68.72 ± 0.68
CIFAR10	49.12 ± 1.15	49.65 ± 2.51	43.42 ± 1.54	44.94 ± 2.50	40.64 ± 2.07	40.54 ± 3.11
Average	67.16	63.41	64.36	63.71	56.63	58.72

Table 5.4: Test set performance metrics, 10 clients

Mean Accuracy & STD (5 seed)						
Dataset	N=4 (Our)	FedFisher KFAC	REGMEAN	DENSE	OTFUSION	FISHER MERGE
$\alpha = 0.3$						
FMNIST	75.83 ± 0.63	74.83 ± 3.11	73.95 ± 3.18	74.33 ± 3.95	64.34 ± 4.54	66.84 ± 6.07
SVHN	75.73 ± 1.89	73.71 ± 0.69	78.54 ± 0.95	64.99 ± 5.75	67.76 ± 3.39	63.99 ± 0.39
CIFAR10	47.98 ± 2.09	50.49 ± 1.73	44.48 ± 1.74	45.39 ± 3.18	41.39 ± 1.28	38.10 ± 2.58
Average	66.51	66.34	65.66	61.57	57.83	56.31

improvement in some cases like FashionMNIST, across varying heterogeneity settings and varying number of clients.

5.2.6 Discussion

In this Section, we presented a methodological contribution to the One-Shot federated learning paradigm based on Bayesian principles that exploits a mixture of Gaussians to approximate both the local and, consequently, the global posterior. Our illustrative numerical results indicate that our methods achieve competitive and promising performance compared to existing state-of-the-art methods.

One of the main limitations of the proposed approach is its computational cost since it is based on a mixture of N models trained in parallel instead of just one. However, it is worth mentioning that there are approaches that mitigate this problem (e.g., [HJF⁺21]) that mimic ensemble prediction but with just one single model. For future work, we plan to extend the experimental analysis by exploring additional types of Laplace approximations, such as those using diagonal or Kronecker-factorized covariance matrices. Moreover, we plan to study the adaptability of the proposed approach to dynamic environments, where, for instance, new clients join the federated learning scheme. Last, We also aim to apply the proposed model in the context of network intrusion detection.

Conclusion

In this thesis, we investigated how the peculiarities of network intrusion detection should be reflected in the design principles behind the ML models employed in this field. We briefly summarize the main contributions presented in this manuscript, followed by possible future research directions.

Model Trustworthiness. Given the safety-critical nature of network intrusion detection, we argued in Chapter 3 for the necessity that a model’s predictions align with our expectations, particularly for tasks relevant to the domain, such as recognizing semantically novel types of inputs (e.g., new kinds of attacks). Specifically, we demonstrated how uncertainty-aware models can inherently address these challenges in a principled way. Furthermore, we proposed a custom model to improve predictive performance compared to current state-of-the-art models.

Continual Learning. In Chapter 3, we also addressed the problem of adapting an ML model to new patterns. To this end, we proposed a hierarchical model particularly suited for resource-constrained scenarios. The model is based on a Bayesian neural network that continuously adapts to new types of attacks and normal traffic, along with a generative classifier responsible for identifying the specific types of attacks in a class-incremental setting.

Distributed Training. Modern networks often include edge nodes with some computational power. It is therefore possible to allow these nodes to train a shared model while keeping data local, thus enhancing privacy and computational efficiency. Federated learning aims to address this objective, but common approaches to FL struggle in the presence of statistical heterogeneity among clients, which is typical in intrusion detection scenarios. To address this, in Chapter 4, we propose a clustering approach to FL with the overall aim of mitigating the effects of statistical heterogeneity and improving the predictive performance of an IDS trained in a federated way.

Trustworthiness in FL settings. Up to now, we have discussed model trustworthiness and federated learning independently. In Chapter 5, we presented some preliminary results aimed at merging these two themes. We propose a federated approach to calibrate any given pre-trained model and we evaluate it in an intrusion detection scenario. Last we presented a

methodological contribution to the ML field, where we leveraged Bayesian inference to design an algorithm aimed at enabling one-shot federated learning, reaching competitive performance concerning state-of-the-art models.

Future Research Directions

In this thesis, our emphasis has been on designing models starting from the peculiarities of the considered domain. Although we have introduced novel approaches to address these, it is important to acknowledge that several challenges remain unresolved.

Regarding model trustworthiness, there is room for improvement, such as enhancing the OoD detection capabilities of the considered models. Uncertainty quantification is a continuously evolving field, and ML-based Intrusion Detection Systems stand to benefit from newer, more advanced approaches greatly. Furthermore, the methodologies used to evaluate these new techniques require thorough investigation to ensure robust and reliable testing frameworks. In this regard, it is essential to have open-source, realistic datasets where the models can be tested in real-world scenarios where the issues discussed in this thesis, like distribution shifts, could degrade the reliability of ML models. In addition, it would be interesting to explore alternative frameworks for addressing the issues mentioned above, beyond the Bayesian approach, such as conformal prediction or fuzzy logic.

Federated learning is a promising direction for training an IDS in a distributed way, but it presents several challenges that need to be properly addressed. In the context of IDS, it is crucial to explore FL algorithms that provide guarantees of uncertainty awareness and hence their reliability. Additionally, it would be relevant to extend existing FL algorithms to handle dynamic scenarios, such as distribution shifts and, more broadly, data streams.

In conclusion, we hope this work serves as a small step toward a more unified and holistic approach to ML-based network intrusion detection, focusing on methodological improvements that could help enhance the trustworthiness and scalability of machine learning models, particularly in comparison to traditional, centralized approaches.

Bibliography

- [AATS23] Oluwadamilare Harazeem Abdulganiyu, Taha Ait Tchakoucht, and Yakub Kayode Saheed. A systematic literature review for network intrusion detection system (ids). *International journal of information security*, 22(5):1125–1162, 2023.
- [ABB⁺21] Shahar Avin, Haydn Belfield, Miles Brundage, Gretchen Krueger, Jasmine Wang, Adrian Weller, Markus Anderljung, Igor Krawczuk, David Krueger, Jonathan Lebensold, et al. Filling gaps in trustworthy development of ai. *Science*, 374(6573):1327–1329, 2021.
- [ABC⁺16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.
- [AGM⁺18] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. *Advances in neural information processing systems*, 31, 2018.
- [ALM⁺23] Giovanni Apruzzese, Pavel Laskov, Edgardo Montes de Oca, Wissam Mallouli, Luis Brdalo Rapa, Athanasios Vasileios Grammatopoulos, and Fabio Di Franco. The role of machine learning in cybersecurity. *Digital Threats: Research and Practice*, 4(1):1–38, 2023.
- [AMCA20] Noor Ali Al-Athba Al-Marri, Bekir S Ciftler, and Mohamed M Abdallah. Federated mimic learning for privacy preserving intrusion detection. In *2020 IEEE international black sea conference on communications and networking (BlackSeaCom)*, pages 1–6. IEEE, 2020.

- [AMT⁺20] Abdullah Alsaedi, Nour Moustafa, Zahir Tari, Abdun Mahmood, and Adnan Anwar. Ton-iot telemetry dataset: A new generation dataset of iot and iiot for data-driven intrusion detection systems. *IEEE Access*, 8:165130–165150, 2020.
- [AOS⁺16] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.
- [APC22] Giovanni Apruzzese, Luca Pajola, and Mauro Conti. The cross-evaluation of machine learning-based network intrusion detection systems. *IEEE Transactions on Network and Service Management*, 19(4):5152–5169, dec 2022.
- [ARI⁺22] Shamsair Ali, Saif Ur Rehman, Azhar Imran, Ghazif Adeem, Zafar Iqbal, and Ki-Il Kim. Comparative evaluation of ai-based techniques for zero-day attacks detection. *Electronics*, 11(23):3934, 2022.
- [ASY⁺19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2623–2631, 2019.
- [AYMS21] Ahmed Aleesa, MOHAMMED Younis, Ahmed A Mohammed, and N Sahar. Deep-intrusion detection system with enhanced unsw-nb15 dataset based on deep learning techniques. *Journal of Engineering Science and Technology*, 16(1):711–727, 2021.
- [BB16] Abhijit Bendale and Terrance E Boult. Towards open set deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1563–1572, 2016.
- [BBCC19] Daniel S Berman, Anna L Buczak, Jeffrey S Chavis, and Cherita L Corbett. A survey of deep learning methods for cyber security. *Information*, 10(4):122, 2019.
- [BCKW15a] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International conference on machine learning*, pages 1613–1622. PMLR, 2015.
- [BCKW15b] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In

- International conference on machine learning*, pages 1613–1622. PMLR, 2015.
- [Bis] Christopher M Bishop. *Pattern recognition and machine learning*, volume 4. Springer.
- [BMGP23] Sherine Wise Betsy, Anitha Murugesan, N. Bala Sundara Ganapathy, and N. Pughazendi. A novel framework for network intrusion detection in healthcare domain. In *2023 4th International Conference on Signal Processing and Communication (ICSPC)*, pages 43–46, 2023.
- [BP75] Richard E Barlow and Frank Proschan. Importance of system components and fault tree events. *Stochastic Processes and their applications*, 3(2):153–173, 1975.
- [Bra97] Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.
- [BTM⁺20] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchu Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, et al. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.
- [CFLS22] Nicola Capuano, Giuseppe Fenza, Vincenzo Loia, and Claudio Stanzione. Explainable artificial intelligence in cybersecurity: A survey. *Ieee Access*, 10:93575–93600, 2022.
- [CG16] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.
- [CGJ96] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 4:129–145, 1996.
- [CHHB24] Yun-Wei Chu, Dong-Jun Han, Seyyedali Hosseinalipour, and Christopher Brinton. Unlocking the potential of model calibration in federated learning. *arXiv preprint arXiv:2409.04901*, 2024.
- [Cla04] Benoit Claise. Cisco systems NetFlow services export version 9. Technical report, IETF RFC, 2004.

- [CSGV⁺21] Enrique Mármol Campos, Pablo Fernández Saura, Aurora González-Vidal, José L Hernández-Ramos, and et al. Evaluating federated learning for intrusion detection in internet of things: Review and challenges. *Computer Networks*, page 108661, 2021.
- [CSLB20] Chin-Wei Chen, Ching-Hung Su, Kun-Wei Lee, and Ping-Hao Bair. Malware family classification using active learning by learning. In *2020 22nd International Conference on Advanced Communication Technology (ICACT)*, pages 590–595, 2020.
- [CUH15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.
- [CWL⁺20] Ginevra Carbone, Matthew Wicker, Luca Laurenti, Andrea Patane, Luca Bortolussi, and Guido Sanguinetti. Robustness of bayesian neural networks to gradient-based attacks. *Advances in Neural Information Processing Systems*, 33:15602–15613, 2020.
- [DA22] Mahendra Data and Masayoshi Aritsugi. An incremental learning algorithm on imbalanced data for network intrusion detection systems. In *Proceedings of the 10th International Conference on Computer and Communications Management*, pages 191–199, 2022.
- [Daw82] A Philip Dawid. The well-calibrated bayesian. *Journal of the American Statistical Association*, 77(379):605–610, 1982.
- [DCKM⁺23] Roberto Doriguzzi-Corin, Luis Augusto Dias Knob, Luca Mendozzi, Domenico Siracusa, and Marco Savi. Introducing packet-level analysis in programmable data planes to advance network intrusion detection, 2023.
- [DHLDVU18] Stefan Depeweg, Jose-Miguel Hernandez-Lobato, Finale Doshi-Velez, and Steffen Udfluft. Decomposition of uncertainty in bayesian deep learning for efficient and risk-sensitive learning. In *International Conference on Machine Learning*, pages 1184–1193. PMLR, 2018.
- [DKI⁺21] Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux-effortless bayesian deep learning. *Advances in Neural Information Processing Systems*, 34:20089–20103, 2021.

- [DLT⁺17a] Joshua V Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*, 2017.
- [DLT⁺17b] Joshua V Dillon, Ian Langmore, Dustin Tran, et al. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*, 2017.
- [EDHK21] Runa Eschenhagen, Erik Daxberger, Philipp Hennig, and Agustinus Kristiadi. Mixtures of laplace approximations for improved post-hoc uncertainty in deep learning. *arXiv preprint arXiv:2111.03577*, 2021.
- [Eur16] European Parliament. European Parliament. General Data Protection Regulation (GDPR). <https://gdpr-info.eu/>, 2016. [Accessed: 07-September-2024].
- [Eur22] European Union Agency for Cybersecurity (ENISA). ENISA Threat Landscape 2022. <https://www.enisa.europa.eu/publications/enisa-threat-landscape-2022>, 2022. [Accessed: 03-August-2023].
- [EY⁺10] Ran El-Yaniv et al. On the foundations of noise-free selective classification. *Journal of Machine Learning Research*, 11(5), 2010.
- [FFS⁺22] Othmane Friha, Mohamed Amine Ferrag, Lei Shu, Leandros Maglaras, Kim-Kwang Raymond Choo, and Mehdi Nafaa. Felids: Federated learning-based intrusion detection system for agricultural internet of things. *Journal of Parallel and Distributed Computing*, 165:17–31, 2022.
- [FNS22] Elena Fedorchenko, Evgenia Novikova, and Anton Shulepov. Comparative review of the intrusion detection systems based on federated learning: Advantages and open challenges. *Algorithms*, 15(7):247, 2022.
- [GBR⁺12] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.
- [GCYR20] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. An efficient framework for clustered federated learning. *Advances in Neural Information Processing Systems*, 33:19586–19597, 2020.

- [GG16] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.
- [GHC20] Chuanxing Geng, Sheng-jun Huang, and Songcan Chen. Recent advances in open set recognition: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3614–3631, 2020.
- [GIG17] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. In *International conference on machine learning*, pages 1183–1192. PMLR, 2017.
- [GPSW17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.
- [GSK⁺21] Xuan Gong, Abhishek Sharma, Srikrishna Karanam, Ziyang Wu, Terrence Chen, David Doermann, and Arun Innanaje. Ensemble attention distillation for privacy-preserving federated learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15076–15086, 2021.
- [GTS19] Neel Guha, Ameet Talwalkar, and Virginia Smith. One-shot federated learning. *arXiv preprint arXiv:1902.11175*, 2019.
- [Guo22] Yang Guo. A review of machine learning-based zero-day attack detection: Challenges and future directions. *Computer Communications*, 2022.
- [HAT⁺20] Hanan Hindy, Robert Atkinson, Christos Tachtatzis, Jean-Noël Colin, Ethan Bayne, and Xavier Bellekens. Utilising deep learning techniques for effective zero-day attack detection. *Electronics*, 9(10):1684, 2020.
- [HBG23] Mehrdad Hajizadeh, Sudip Barua, and Pegah Golchin. Fsa-ids: A flow-based self-active intrusion detection system. In *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9, 2023.
- [HCS⁺19] Vikas Hassija, Vinay Chamola, Vikas Saxena, Divyansh Jain, and et al. A survey on iot security: Application areas, security

- threats, and solution architectures. *IEEE Access*, 7:82721–82743, 2019.
- [Hel09] Ernst Hellinger. *Journal für die reine und angewandte mathematik*. 210, 1909.
- [HG18] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks, 2018.
- [HHGL11] Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.
- [HJF⁺21] Marton Havasi, Rodolphe Jenatton, Stanislav Fort, Jeremiah Zhe Liu, Jasper Snoek, Balaji Lakshminarayanan, Andrew M. Dai, and Dustin Tran. Training independent subnetworks for robust prediction, 2021.
- [HJJ23] Arash Heidari and Mohammad Ali Jabraeil Jamali. Internet of things intrusion detection systems: a comprehensive review and future directions. *Cluster Computing*, 26(6):3753–3780, 2023.
- [HLRS23] Clare Elizabeth Heinbaugh, Emilio Luz-Ricca, and Huajie Shao. Data-free one-shot federated learning under very high statistical heterogeneity. In *The Eleventh International Conference on Learning Representations*, 2023.
- [HQB19] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.
- [HTA⁺23] Hanan Hindy, Christos Tachtatzis, Robert Atkinson, David Brosset, Miroslav Bures, Ivan Andonovic, Craig Michie, and Xavier Bellekens. Leveraging siamese networks for one-shot intrusion detection model. *Journal of Intelligent Information Systems*, 60(2):407–436, 2023.
- [HYW⁺20] Xinhong Hei, Xinyue Yin, Yichuan Wang, Ju Ren, and Lei Zhu. A trusted feature aggregator federated learning for distributed malicious attack detection. *Computers & Security*, 99:102033, 2020.
- [HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

- [IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.
- [Jay03] E. T. Jaynes. *Probability theory: The logic of science*. Cambridge University Press, Cambridge, 2003.
- [JKRK19] Yihan Jiang, Jakub Konečný, Keith Rush, and Sreeram Kannan. Improving federated learning personalization via model agnostic meta learning. *arXiv preprint arXiv:1909.12488*, 2019.
- [JRPPC] Xisen Jin, Xiang Ren, Daniel Preotiuc-Pietro, and Pengxiang Cheng. Dataless knowledge fusion by merging weights of language models. In *The Eleventh International Conference on Learning Representations*.
- [JSD⁺17] Roberto Jordane, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 625–642, 2017.
- [JSS⁺] Keller Jordan, Hanie Sedghi, Olga Saukh, Rahim Entezari, and Behnam Neyshabur. Repair: Renormalizing permuted activations for interpolation repair. In *The Eleventh International Conference on Learning Representations*.
- [JWJ24] Divyansh Jhunjhunwala, Shiqiang Wang, and Gauri Joshi. Fedfisher: Leveraging fisher information for one-shot federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1612–1620. PMLR, 2024.
- [KAH⁺19] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235, 2019.
- [KB17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [KKK19] Muhammad Ashfaq Khan, Md Rezaul Karim, and Yangwoo Kim. A scalable and hybrid intrusion detection system based on the convolutional-lstm network. *Symmetry*, 11(4):583, 2019.
- [KKM⁺20] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh.

- Scaffold: Stochastic controlled averaging for federated learning. In *International conference on machine learning*, pages 5132–5143. PMLR, 2020.
- [KMA⁺21] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.
- [KNH10] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL <http://www.cs.toronto.edu/kriz/cifar.html>, 5(4):1, 2010.
- [KNJ⁺20] Harmanpreet Kaur, Harsha Nori, Samuel Jenkins, Rich Caruana, Hanna Wallach, and Jennifer Wortman Vaughan. Interpreting interpretability: Understanding data scientists’ use of interpretability tools for machine learning. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI ’20, page 1â14, New York, NY, USA, 2020. Association for Computing Machinery.
- [Kon16] Jakub Konečný. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.
- [KPR⁺17] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.
- [LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [Lin91] J. Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991.
- [LKSJ20] Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. *Advances in neural information processing systems*, 33:2351–2363, 2020.
- [LL19] Qing Lyu and Xingjian Lu. Effective media traffic classification using deep learning. In *Proceedings of the 2019 3rd International Conference on Compute and Data Analysis*, ICCDA

'19, page 139–146, New York, NY, USA, 2019. Association for Computing Machinery.

- [LLL⁺21] Jianhua Li, Lingjuan Lyu, Ximeng Liu, Xuyun Zhang, and Xixiang Lyu. Fleam: A federated learning empowered architecture to mitigate ddos in industrial iot. *IEEE Transactions on Industrial Informatics*, 18(6):4059–4068, 2021.
- [LLLT13] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.
- [LLP⁺20a] Jeremiah Liu, Zi Lin, Shreyas Padhy, Dustin Tran, Tania Bedrax Weiss, and Balaji Lakshminarayanan. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. *Advances in Neural Information Processing Systems*, 33:7498–7512, 2020.
- [LLP⁺20b] Jeremiah Liu, Zi Lin, Shreyas Padhy, Dustin Tran, Tania Bedrax Weiss, and Balaji Lakshminarayanan. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. *Advances in Neural Information Processing Systems*, 33:7498–7512, 2020.
- [LLW⁺22] Chang Liu, Chenfei Lou, Runzhong Wang, Alan Yuhan Xi, Li Shen, and Junchi Yan. Deep neural network fusion via graph matching with applications to model ensemble and federated learning. In *International Conference on Machine Learning*, pages 13857–13869. PMLR, 2022.
- [LPB17] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.
- [LSTS20] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.
- [LSZ⁺20] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020.

- [LWOL20] Weitang Liu, Xiaoyun Wang, John Owens, and Yixuan Li. Energy-based out-of-distribution detection. *Advances in neural information processing systems*, 33:21464–21475, 2020.
- [LWY⁺23] Chaomeng Lu, Xufeng Wang, Aimin Yang, Yikai Liu, and Ziao Dong. A few-shot-based model-agnostic meta-learning for intrusion detection in security of internet of things. *IEEE Internet of Things Journal*, 10(24):21309–21321, 2023.
- [Mac92] David JC MacKay. Bayesian interpolation. *Neural computation*, 4(3):415–447, 1992.
- [Mac95] David JC MacKay. Probable networks and plausible predictions—a review of practical bayesian methods for supervised neural networks. *Network: computation in neural systems*, 6(3):469, 1995.
- [Mac03] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.
- [Mar23] Othmane Marfoq. *Tackling heterogeneity in federated learning systems*. Theses, Université Côte d’Azur, December 2023.
- [MKKY18] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks, 2018.
- [MKvA⁺21] Jishnu Mukhoti, Andreas Kirsch, Joost van Amersfoort, Philip HS Torr, and Yarín Gal. Deterministic neural networks with inductive biases capture epistemic and aleatoric uncertainty. *arXiv preprint arXiv:2102.11582*, page 13, 2021.
- [MLBP22] Tijana Markovic, Miguel Leon, David Buffoni, and Sasikumar Punnekkat. Random Forest based on Federated Learning for Intrusion Detection. In *AIAI 2022*. Springer, 2022.
- [MLL⁺18] Erxue Min, Jun Long, Qiang Liu, Jianjing Cui, Zhiping Cai, and Junbo Ma. Su-ids: A semi-supervised and unsupervised framework for network intrusion detection. In *Cloud Computing and Security: 4th International Conference, ICCCS 2018, Haikou, China, June 8–10, 2018, Revised Selected Papers, Part III 4*, pages 322–334. Springer, 2018.
- [MMR⁺17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient

- learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.
- [MMRS20] Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. Three approaches for personalization with applications to federated learning. *arXiv preprint arXiv:2002.10619*, 2020.
- [MNB⁺21] Othmane Marfoq, Giovanni Neglia, Aurélien Bellet, Laetitia Kamani, and Richard Vidal. Federated multi-task learning under a mixture of distributions. *Advances in Neural Information Processing Systems*, 34:15434–15447, 2021.
- [MPP⁺20] Virraji Mothukuri, Reza Parizi, Seyedamin Pouriye, Yan Huang, and et al. A survey on security and privacy of federated learning. *Future Generation Computer Systems*, 10 2020.
- [MR22] Michael S Matena and Colin A Raffel. Merging models with fisher-weighted averaging. *Advances in Neural Information Processing Systems*, 35:17703–17716, 2022.
- [Mur23] Kevin P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023.
- [MWF24] Mayra Macas, Chunming Wu, and Walter Fuertes. Adversarial examples: A survey of attacks and defenses in deep learning-enabled cybersecurity systems. *Expert Systems with Applications*, 238:122223, 2024.
- [NCH15] Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.
- [NDZ⁺19] Jeremy Nixon, Michael W Dusenberry, Linchuan Zhang, Ghasen Jerfel, and Dustin Tran. Measuring calibration in deep learning. In *CVPR workshops*, pages 38–41, 2019.
- [Nea12] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.
- [NLB⁺18] Cuong V Nguyen, Yingzhen Li, Thang D Bui, et al. Variational continual learning. In *International Conference on Learning Representations*, 2018.

- [NMM⁺19] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Hossein Fereidooni, Nadarajah Asokan, and Ahmad-Reza Sadeghi. Diot: A federated self-learning anomaly detection system for iot. In *2019 IEEE 39th International conference on distributed computing systems (ICDCS)*, pages 756–767. IEEE, 2019.
- [NWC⁺11] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 4. Granada, 2011.
- [NYC15] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 427–436, 2015.
- [NYKW12] Hieu T Nguyen, Joseph Yadegar, Bailey Kong, and Hai Wei. Efficient batch-mode active learning of random forest. In *2012 IEEE Statistical Signal Processing Workshop (SSP)*, pages 596–599. IEEE, 2012.
- [OIIT23] Chrysoula Oikonomou, Ilias Iliopoulos, Dimosthenis Ioannidis, and Dimitrios Tzovaras. A multi-class intrusion detection system based on continual learning. In *2023 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 86–91. IEEE, 2023.
- [OK11] Patrick O’Connor and Andre Kleyner. *Practical reliability engineering*. John Wiley & Sons, 2011.
- [P⁺19] Adam Paszke et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS 2019*, 2019.
- [PKP⁺19] German I Parisi, Ronald Kemker, Jose L Part, et al. Continual lifelong learning with neural networks: A review. *Neural networks*, 113:54–71, 2019.
- [PSM⁺22] Sai Prasath, Kamalakanta Sethi, Dinesh Mohanty, et al. Analysis of continual learning models for intrusion detection system. *IEEE Access*, 10:121444–121464, 2022.
- [PSP⁺24] Theodore Papamarkou, Maria Skoularidou, Konstantina Palla, Laurence Aitchison, Julyan Arbel, David Dunson, Maurizio Filippone, Vincent Fortuin, Philipp Hennig, Aliaksandr Hubin,

et al. Position paper: Bayesian deep learning in the age of large-scale ai. *arXiv e-prints*, pages arXiv–2402, 2024.

- [PVG⁺11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [PYTL24] Hongyi Peng, Han Yu, Xiaoli Tang, and Xiaoxiao Li. FedCal: Achieving Local and Global Calibration in Federated Learning via Aggregated Parameterized Scaler. In *International Conference on Machine Learning (ICML)*, 2024.
- [QMC⁺23] Zhuang Qi, Lei Meng, Zitan Chen, Han Hu, Hui Lin, and Xiangxu Meng. Cross-silo prototypical calibration for federated learning with non-iid data. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 3099–3107, 2023.
- [RBB18] Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. In *6th international conference on learning representations, ICLR 2018-conference track proceedings*, volume 6. International Conference on Representation Learning, 2018.
- [Roe99] Martin Roesch. Snort: Lightweight intrusion detection for networks. In *LISA*, pages 229–238. USENIX, 1999.
- [RSCB22] Valerian Rey, Pedro Miguel Sánchez Sánchez, Alberto Huertas Celdrán, and G er ome Bovet. Federated learning for malware detection in iot devices. *Computer Networks*, 204:108693, 2022.
- [RSER21] Ishai Rosenberg, Asaf Shabtai, Yuval Elovici, and Lior Rokach. Adversarial machine learning attacks and defense methods in the cyber security domain. *ACM Computing Surveys (CSUR)*, 54(5):1–36, 2021.
- [RTTM20] Sawsan Abdul Rahman, Hanine Tout, Chamseddine Talhi, and Azzam Mourad. Internet of things intrusion detection: Centralized, on-device, or federated learning? *IEEE Network*, 34(6):310–317, 2020.

- [RW06] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.
- [SCST17] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. Federated multi-task learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17*, page 4427–4437, Red Hook, NY, USA, 2017. Curran Associates Inc.
- [SD21] Bam Sinha and R. Dhanalakshmi. Recent advancements and challenges of internet of things in smart agriculture: A survey. *Future Generation Computer Systems*, 126, 08 2021.
- [SG18a] Lewis Smith and Yarin Gal. Understanding measures of uncertainty for adversarial example detection. *arXiv preprint arXiv:1803.08533*, 2018.
- [SG18b] Lewis Smith and Yarin Gal. Understanding measures of uncertainty for adversarial example detection. *arXiv preprint arXiv:1803.08533*, 2018.
- [SGJM23] D Shankar, G Victo Sudha George, Janardhana Naidu JNSS, and P Shyamala Madhuri. Deep analysis of risks and recent trends towards network intrusion detection system. *International Journal of Advanced Computer Science and Applications*, 14(1), 2023.
- [SH20] Mohammad Hossein Shaker and Eyke Höllermeier. Aleatoric and epistemic uncertainty with random forests, 2020.
- [SHK⁺14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
- [SJ20] Sidak Pal Singh and Martin Jaggi. Model fusion via optimal transport. *Advances in Neural Information Processing Systems*, 33:22045–22055, 2020.
- [SKF⁺23] Freddie Bickford Smith, Andreas Kirsch, Sebastian Farquhar, Yarin Gal, Adam Foster, and Tom Rainforth. Prediction-oriented bayesian active learning. In *International Conference on Artificial Intelligence and Statistics*, pages 7331–7348. PMLR, 2023.

- [SLG⁺18] Iman Sharafaldin, Arash Habibi Lashkari, Ali A Ghorbani, et al. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.
- [SLP21a] Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. Evaluating standard feature sets towards increased generalisability and explainability of ml-based network intrusion detection. *arXiv preprint arXiv:2104.07183*, 2021.
- [SLP21b] Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. Towards a standard feature set for network intrusion detection system datasets. *Mobile Networks and Applications*, 27(1):357–370, nov 2021.
- [SMS20] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. *IEEE transactions on neural networks and learning systems*, 32(8):3710–3722, 2020.
- [SMS⁺22] Deepak Kumar Sharma, Jahanavi Mishra, Aeshit Singh, Raghav Govil, Gautam Srivastava, and Jerry Chun-Wei Lin. Explainable artificial intelligence for cybersecurity. *Computers and Electrical Engineering*, 103:108356, 2022.
- [SNPS18] Nathan Shone, Tran Nguyen Ngoc, Vu Dinh Phai, and Qi Shi. A deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(1):41–50, 2018.
- [SO21] Marco Savi and Fabrizio Olivadese. Short-term energy consumption forecasting at the edge: A federated learning approach. *IEEE Access*, 9:95949–95969, 2021.
- [SP10] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE Symposium on Security and Privacy*, pages 305–316, 2010.
- [SPG⁺22] Manuela M. C. Souza, Camila Pontes, Joao Gondim, Luis P. F. Garcia, Luiz DaSilva, and Marcelo A. Marotta. A novel open set energy-based flow classifier for network intrusion detection, 2022.
- [SSS⁺10] Anna Sperotto, Gregor Schaffrath, Ramin Sadre, Cristian Morariu, Aiko Pras, and Burkhard Stiller. An overview of ip

- flow-based intrusion detection. *IEEE communications surveys & tutorials*, 12(3):343–356, 2010.
- [SYL⁺17] Yanyao Shen, Hyokun Yun, Zachary C Lipton, Yakov Kronrod, and Animashree Anandkumar. Deep active learning for named entity recognition. *arXiv preprint arXiv:1707.05928*, 2017.
- [SZA22] Ravid Shwartz-Ziv and Amitai Armon. Tabular Data: Deep Learning is not All You Need. *Information Fusion*, 81:84–90, 2022.
- [SZGL⁺23] Ravid Shwartz-Ziv, Micah Goldblum, Yucen Lily Li, C. Bayan Bruss, and Andrew Gordon Wilson. On representation learning under class imbalance, 2023.
- [TJZ⁺21a] Zachary Tauscher, Yushan Jiang, Kai Zhang, et al. Learning to detect: A data-driven approach for network intrusion detection. In *2021 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pages 1–6. IEEE, 2021.
- [TJZ⁺21b] Zachary Tauscher, Yushan Jiang, Kai Zhang, Jian Wang, and Houbing Song. Learning to detect: A data-driven approach for network intrusion detection. In *2021 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pages 1–6. IEEE, 2021.
- [TL23] Ankit Thakkar and Ritika Lohiya. A review on challenges and future research directions for machine learning-based intrusion detection system. *Archives of Computational Methods in Engineering*, 30(7):4245–4269, 2023.
- [TLD⁺22] Dustin Tran, Jeremiah Liu, Michael W Dusenberry, Du Phan, Mark Collier, Jie Ren, Kehang Han, Zi Wang, Zelda Mariet, Huiyi Hu, et al. Plex: Towards reliability using pretrained large model extensions. *arXiv preprint arXiv:2207.07411*, 2022.
- [TLR22a] Stefanos Tsimenidids, Thomas Lagkas, and Konstantinos Rantos. Deep learning in iot intrusion detection. *Journal of Network and Systems Management*, 30, 01 2022.
- [TLR22b] Stefanos Tsimenidids, Thomas Lagkas, and Konstantinos Rantos. Deep learning in iot intrusion detection. *Journal of Network and Systems Management*, 30, 01 2022.

- [USB17] Muhammad Fahad Umer, Muhammad Sher, and Yaxin Bi. Flow-based intrusion detection: Techniques and challenges. *Computers & Security*, 70:238–254, 2017.
- [VAS⁺19a] Ravi Vinayakumar, Mamoun Alazab, K Padannayil Soman, Prabaharan Poornachandran, Ameer Al-Nemrat, and Sitalakshmi Venkatraman. Deep learning approach for intelligent intrusion detection system. *Ieee Access*, 7:41525–41550, 2019.
- [VAS⁺19b] Ravi Vinayakumar, Mamoun Alazab, KP Soman, Prabaharan Poornachandran, Ameer Al-Nemrat, and Sitalakshmi Venkatraman. Deep learning approach for intelligent intrusion detection system. *Ieee Access*, 7:41525–41550, 2019.
- [VASTG20a] Joost Van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. Uncertainty estimation using a single deep deterministic neural network. In *International conference on machine learning*, pages 9690–9700. PMLR, 2020.
- [VASTG20b] Joost Van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. Uncertainty estimation using a single deep deterministic neural network. In *International conference on machine learning*, pages 9690–9700. PMLR, 2020.
- [VdMH08] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.
- [VR20] Abhishek Verma and Virender Ranga. Machine learning based intrusion detection systems for iot applications. *Wireless Personal Communications*, 111:2287–2310, 2020.
- [VWB⁺16] Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios S. Nikolopoulos. Challenges and opportunities in edge computing. In *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 20–26, 2016.
- [WI20] Andrew G Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, 33:4697–4708, 2020.
- [WLHS21] Tingting Wang, Qiujuan Lv, Bo Hu, and Degang Sun. A few-shot class-incremental learning approach for intrusion detection. In *2021 International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2021.

- [WLL⁺20] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. *Advances in neural information processing systems*, 33:7611–7623, 2020.
- [WVB⁺18] Yeming Wen, Paul Vicol, Jimmy Ba, Dustin Tran, and Roger Grosse. Flipout: Efficient pseudo-independent weight perturbations on mini-batches, 2018.
- [WWHX20] Haohan Wang, Xindi Wu, Zeyi Huang, and Eric P Xing. High-frequency component helps explain the generalization of convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8684–8694, 2020.
- [XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.
- [YTC⁺23] Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. Resolving interference when merging models. *arXiv preprint arXiv:2306.01708*, 1, 2023.
- [YZLL21] Jingkang Yang, Kaiyang Zhou, Yixuan Li, and Ziwei Liu. Generalized out-of-distribution detection: A survey. *arXiv preprint arXiv:2110.11334*, 2021.
- [YZY19] Yuhang Ye, Tong Zhang, and Chenguang Yang. Fisher loss: A more discriminative feature learning method in classification. In *2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 746–751, 2019.
- [ZCB21] Tommaso Zoppi, Andrea Ceccarelli, and Andrea Bondavalli. Unsupervised algorithms to detect zero-day attacks: Strategy and application. *IEEE Access*, 9:90603–90615, 2021.
- [ZCL⁺22] Jie Zhang, Chen Chen, Bo Li, Lingjuan Lyu, Shuang Wu, Shouhong Ding, Chunhua Shen, and Chao Wu. Dense: Data-free one-shot federated learning. *Advances in Neural Information Processing Systems*, 35:21414–21428, 2022.
- [ZJP⁺20] Yuheng Zhang, Ruoxi Jia, Hengzhi Pei, Wenxiao Wang, Bo Li, and Dawn Song. The secret revealer: Generative model-inversion attacks against deep neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 253–261, 2020.

- [ZLY⁺22] Shuai Zhou, Chi Liu, Dayong Ye, Tianqing Zhu, Wanlei Zhou, and Philip S Yu. Adversarial attacks and defenses in deep learning: From a perspective of cybersecurity. *ACM Computing Surveys*, 55(8):1–39, 2022.
- [ZPM⁺20] Yanlin Zhou, George Pu, Xiyao Ma, Xiaolin Li, and Dapeng Wu. Distilled one-shot federated learning. *arXiv preprint arXiv:2009.07999*, 2020.
- [ZZDZ23] Mingyuan Zang, Changgang Zheng, Lars Dittmann, and Noa Zilberman. Towards continuous threat defense: In-network traffic analysis for iot gateways. *IEEE Internet of Things Journal*, pages 1–1, 2023.
- [ZZGS21] Zhao Zhang, Yong Zhang, Da Guo, and Mei Song. A scalable network intrusion detection system towards detecting, discovering, and learning unknown attacks. *International Journal of Machine Learning and Cybernetics*, 12:1649–1665, 2021.

La borsa di dottorato cofinanziata con risorse del
Programma Operativo Nazionale Ricerca e Innovazione 2014-2020 (CCI 2014IT16M2OP005),
risorse Fondo Sociale Europeo REACT-EU, Azione I.1 “Dottorati Innovativi con caratterizzazione industriale”, Azione
IV.4 “Dottorati e contratti di ricerca su tematiche dell’innovazione” e Azione IV.5 “Dottorati su tematiche Green”



UNIONE EUROPEA
Fondo Sociale Europeo



*Ministero dell'Università
e della Ricerca*



REACT EU

