UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

PH.D. THESIS

IN

INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

A HIERARCHICAL LEARNING FRAMEWORK FOR NETWORK TRAFFIC ANALYSIS: DESIGN, IMPLEMENTATION, AND USE CASES

GIAMPAOLO BOVENZI

TUTOR: PROF. ANTONIO PESCAPÈ

COORDINATOR: PROF. DANIELE RICCIO

XXXIV CICLO

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

TESI DI DOTTORATO

UNIVERSITÀ DEGLI STUDI DI NAPOLI "FEDERICO II"

DIPARTIMENTO DI INGEGNERIA ELETTRONICA
E DELLE TECNOLOGIE DELL'INFORMAZIONE

DOTTORATO DI RICERCA IN
INFORMATION TECHNOLOGIES AND ELECTRICAL ENGINEERING

A HIERARCHICAL LEARNING FRAMEWORK FOR NETWORK TRAFFIC ANALYSIS: DESIGN, IMPLEMENTATION, AND USE CASES

GIAMPAOLO BOVENZI

Il Coordinatore del Corso di Dottorato	Il Tutore
Ch.mo Prof. Daniele RICCIO	Ch.mo Prof. Antonio PESCAPÈ

A. A. 2021–2022

“A nonno Arcangelo e nonna Consiglia”

Acknowledgments

First of all, I want to express my sincere gratitude to my advisor Prof. Antonio Pescapè for his guidance and inspiration during these years. A big thank you go also to my labmates (in rigorous alphabetical order) Antonio, Alfredo, Ciro, Domenico, Fabio, Giuseppe, Idio, Saeed, and Valerio. With their different approaches and points of view, they have been and are a priceless source of inspiration. They guided me through this impervious path and their support and advice brought me closer and closer to my dream job. I would like to state my appreciation to Dr. Eng. Alessandro Finamore, my internship advisor at the Huawei R&D Center, Paris. He gave me different perspectives on research, playing an important role in my professional growth. I am also grateful to Prof. Edmundo Monteiro (University of Coimbra, Portugal) and Prof. Mohammad Shojafar (University of Surrey, United Kingdom) for their valuable comments on the draft version of my Ph.D. Thesis, which allowed me to improve it significantly. My thanks and love go to my parents Annamaria and Luigi, my brothers Alessandro and Dario, and the rest of my big family (it would take a separate thesis to name them all) for their unconditional and constant support. My gratitude goes also to my friends being able to make me smile, despite everything. Finally, special thanks go to Alessandra for her patience, presence, and love. The anxiety infused by this society is lighter when shared with someone on the same page.

Contents

List of Acronyms

1DCNN 1-Dimensional Convolutional Neural Network. 49, 50, 111, 112, 113, 117

2DCNN 2-Dimensional Convolutional Neural Network. 49, 50, 111, 113, 117

AC Attack Classification. 11, 13, 21, 55, 91, 97, 98, 99

AD Anomaly Detection. 11, 12, 13, 21, 55, 90, 91, 92, 95, 97, 98

AE AutoEncoder. 51, 52, 91, 95

AT Anonymity Tool. xv, 10, 19, 20, 63, 64, 65, 66, 67, 69, 72, 76, 78, 79, 81

BD Big Data. xv, xvii, 9, 15, 16, 17, 19, 22, 25, 34, 35, 36, 37, 38, 39, 54, 55, 58, 60, 67, 72, 86

BiGRU Bidirectional Gated Recurrent Unit. 50

BN Bayesian Network. 49, 65, 66, 71, 75, 76

CGC Combined Global Classifier. xv, 43

CLS Contextual Label Smoothing. xix, 46, 47

CLSGC Contextual Label Smoothing Global Classifier. 46

CR Classified Ratio. xv, 81, 82

DAE Deep AutoEncoder. 11, 51, 52, 91, 95, 98, 99

DDoS Distributed Denial of Service. 15, 89, 90, 94, 99

DIR Direction. 107

DL Deep Learning. 5, 11, 14, 17, 18, 20, 21, 22, 25, 33, 41, 49, 50, 59, 60, 65, 71, 104, 105, 106, 111, 112, 114, 116

DoS Denial of Service. 91, 93, 99, 100

DT Decision Tree. 12, 13, 15, 48, 64, 66, 71, 75

FC Flat Classifier. 7, 8

FPR False Positive Rate. 91, 96, 98, 99, 100, 101

GC Global Classifier. 8, 9, 10, 14, 18, 41, 44

HLGC Hierarchical Loss Global Classifier. 45, 46

IAT Inter-Arrival Time. 107

ID Intrusion Detection. 11, 13, 90, 91

IDS Intrusion Detection System. xvi, 10, 12, 13, 21, 90, 91, 93, 98, 99, 100, 101

IF Isolation Forest. 50, 98

IoT Internet of Things. 17, 20, 21, 89, 90, 91, 92, 93, 97, 98, 99

k-NN k-Nearest Neighbors. 13, 14

LCL Local Classifier per Level. 8, 10, 14

LCN Local Classifier per Node. 7, 11

LCPN Local Classifier per Parent Node. xv, 8, 9, 10, 11, 12, 13, 14, 15, 18, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 53, 54, 55, 59, 60, 63, 67, 69, 70, 72, 89, 90, 92, 93, 96, 99, 100, 101, 103, 111, 113

LOF Local Outlier Factor. 50, 98

LSTM Long Short-Term Memory. 49, 50, 111, 112, 113, 117

M2-DAE Multi Modal-Deep AutoEncoder. 52, 93, 95, 97, 98, 99

MCC Multi-Class Classifier. 48

MD Misuse Detection. 11, 12, 13, 21, 90, 91, 92, 98, 99, 100, 101

MGC Multitask Global Classifier. xv, 45

MIMETIC MultImodal DL-based MobilE TraffIc Classification. 50, 111, 113, 117

ML Machine Learning. 5, 10, 11, 13, 14, 15, 17, 18, 19, 20, 21, 22, 25, 33, 35, 36, 37, 39, 40, 41, 55, 59, 65, 66, 71, 75, 84, 99, 104, 105, 106, 110, 111, 112, 114

MLGC Multi-Label Global Classifier. 47

MLP MultiLayer Perceptron. 14, 49, 96, 98, 99

NB Naïve Bayes. 48, 65, 66, 71, 75, 96, 98, 99

NGC Naïve Global Classifier. xv, 42, 43, 47

NIDS Network Intrusion Detection System. 20, 21, 90, 91

NN Neural Network. 12, 15, 41, 43, 45

NTA Network Traffic Analysis. 3, 4, 5, 8, 9, 10, 13, 14, 15, 17, 18, 19, 21, 26, 31

OC-SVM One-class Support Vector Machine. 50, 98

OCC One-Class Classifier. 48, 98

P2P Peer-to-Peer. 10, 11, 12, 13, 14, 65, 66

PL Payload Length. 107

RF Random Forest. 13, 14, 18, 48, 64, 66, 71, 72, 73, 74, 75, 76, 82, 84, 86, 91, 96, 98, 99, 104, 105, 110, 111, 112, 113

SVM Support Vector Machine. 12, 13, 14, 15, 71

TC Traffic Classification. xvii, 11, 16, 19, 20, 33, 36, 40, 41, 49, 64, 65, 66, 67, 69, 70, 71, 72, 73, 75, 76, 77, 79, 80, 81, 82, 84, 85, 86, 87, 103, 104, 105, 106, 110, 111

TIGC Task-Incremental Global Classifier. xv, 44, 45

TO Traffic Object. 15, 26, 27, 28, 29, 30, 35, 52, 69, 92, 93, 94, 95, 96, 99

TPR True Positive Rate. 11, 91, 92, 96, 98, 99

VPN Virtual Private Network. 10, 14

XGB eXtreme Gradient Boosting. 48, 111, 112, 113

List of Figures

List of Tables

List of Algorithms

Introduction

Network traffic analysis covers the entire set of operations and techniques used to gain knowledge about the status of a network, in order to manage and administer it properly. Therefore, traffic analysis solutions resort to modeling techniques applied to the network traffic with the aim of aiding *network operators* and *internet service providers* to achieve a clear snapshot of what is traversing their network.

The main challenges we identified in the nowadays Internet traffic are strictly related to the (i) the huge number of Internet enabled devices which generates *heterogeneous network traffic*; and (ii) the increasing of the generated traffic in terms of *traffic volume*. Accordingly, the main objective of this doctoral thesis is proposing a Hierarchical Learning Framework for Network Traffic Analysis, in order to enhance the *fine-grained network knowledge* and the *scalability of traffic analysis solutions*. This framework enhances traffic analysis exploiting hierarchical dependencies among network traffic classes in order both to improve the fine-grained modeling of network traffic and to enable a modular and scalable learning process, enabling fast retraining.

To this extent, the proposal takes advantage of state-of-the-art machine and deep learning solutions, thus fostering designed hierarchical learning approaches and it is evaluated on different types of Internet traffic (viz. three scenarios), such as the traffic generated by privacy-preserving solutions (e.g., VPNs, Anonymity Tools), network attacks or malware (e.g., Scans, Denial of Services, Botnets), mobile applications (e.g., Games, Social Networks, Video on Demand).

Chapter 1

Background & Positioning

In this chapter, we first introduce *network traffic analysis* and its applications (Sec. 1.1.1). Then, in Sec. 1.1.2 we delve into background methodological aspects, presenting the formalization which constitutes the backbone of the hierarchical learning theory. Following, works related to hierarchical learning solutions in other domains (e.g., protein sequence classification) whose applications fall within the network traffic analysis are separately analyzed in Sec. 1.1.3. Then, in Sec. 1.2 challenges of network traffic analysis are explained and the positioning of the proposed solutions is provided. Finally, a summary section (1.3) closes the chapter.

1.1 Background

1.1.1 Network Traffic Analysis

Network Traffic Analysis (NTA) is an umbrella term which covers the entire set of operations and techniques used to gain knowledge about the status of a network in order to manage and administer it properly [1]. Therefore, NTA solutions resort to modeling techniques applied to the traffic flowing the monitored networks, with the aim of aiding *network operators* and *Internet service providers* to achieve a clear snapshot of what is traversing their network. Along with this definition, NTA enables for the control of the network via traffic engineering methodologies and tools falling within several applications, like accounting, advertising, network monitoring (i.e. fault detection), network traffic measurement (e.g., bandwidth management), security assessment, quality of service enforcement, intrusion detection, anomaly detection, application

identification, service differentiation, and user's activity profiling [2, 3, 4, 5]. Broadly speaking, NTA can be enforced at different points in the temporal line, namely in the past, in the present, and in the future, providing post-mortem, early/near-real-time, and predictive hints, respectively. These three kinds of hints characterize two main tasks, namely *binary/multi-class classification* and *time-series prediction*, where the former covers the past (e.g., labeling) and the present (e.g., real-time identification), and the latter focuses on the future (e.g., load forecasting). It is worth to underline that this taxonomization hides some forms of interleaving of such tasks, with each task which potentially feeds the other (e.g., a classification system can enhance a prediction system providing the class of traffic).

Because network operators and Internet service providers should understand the nature of the communications flowing across the Internet to properly manage networks, this thesis focuses on binary/multi-class classification and its applications, like *traffic identification*, *traffic classification*, *anomaly detection*, *misuse detection*, *attack classification*, *malware detection*, *malware classification*, and *website fingerprinting* [6]. Detailing, **traffic identification and classification** are the tasks which associate traffic objects to the class of traffic they belong, where a traffic object is a collection of network packets sharing common characteristics (e.g., a biflow is composed by packets exchanged between two network sockets). Both traffic identification and classification are also a well-established study area, with several surveys available like [7, 8, 9]. The main difference between these tasks is the binary nature of traffic identification and the multiclass nature of the traffic classification. Focusing on security related applications, the task which aims at separating legitimate traffic from the unwanted one (e.g., attack related) is named **anomaly detection** and it deals with binary classification. Moreover, when the specific attack is inferred we fall in the **attack classification** task, which is multi-class classification. Furthermore, the objective of **misuse detection** is of combining anomaly detection and attack classification, resulting in a multiclass classification system with the capability of distinguishing among benign traffic and (kinds of) attack [10, 11]. Similarly to the difference between anomaly detection and attack classification, **malware detection** has the main objective of separating legitimate traffic to which generated by malware, and **malware classification** gives a deeper hint about the malware generating the identified traffic [12]. Finally, **website fingerprinting** wants to understand to which website (down to the webpage) a user has been connected, by inspecting the traffic generated during the retrieving of such websites. This application is very related to

privacy issues because applicable to identify censorship [13].

Nowadays, traffic analysis applications are more or less wounded by the advancements in privacy & security preserving technologies and standardization (viz. security by design), and by the ever-increasing complexity of the Internet itself, i.e. in terms of the number and diversity of available services and of Internet-enabled devices. In particular, the increased adoption of ubiquitous traffic encryption—with the widespread adoption of protocols such as TLS or QUIC, which want to drastically reduce the portion of non-encrypted network packets—and the definitive shift towards the adoption of nonstandard port numbers have practically wounded the deep-packet-inspection- and port-based classification approaches, respectively. In addition, the extreme dynamicity of the Internet traffic, the heterogeneity of devices connecting to the Internet—especially when considering mobile devices which have ecosystems of tools that ease the installation of new apps and their updates—are all contributing to this renewed interest [14], and call for the need for complex solutions.

Recently, to tackle abovementioned challenges affecting the majority of the networks, researchers are designing and evaluating advanced modeling techniques based on Machine Learning (ML) and Deep Learning (DL) approaches, with the progresses of *artificial intelligence* solutions experienced in the last years reflecting their benefits also in network traffic analysis [15, 16]. Moreover, using ML techniques to perform NTA also meets the need of maintaining privacy by classifying encrypted communication leveraging only statistical attributes, rather than decrypting its content [17]. From a chronological standpoint, NTA literature can be categorized into two "waves". The first wave, ignited in the early 2000's, and centered around the use of ML methods using per-packet (e.g., packet size, packets inter-arrival time) or per-flow (e.g., total bytes, packets, ports) features as input targeting the classification of a handful of applications. Several works demonstrated that even when just a few packets of a flow were observed, the classification was accurate [18, 19], and could be sustained at line-rate speed [20]—"early" NTA was born. Inspired by the success of image processing in computer vision, in the last years DL techniques re-ignited the interest towards NTA, with several DL-based classifiers being proposed using as input either raw payload bytes or the same traffic features discovered during the first wave [14, 21, 22, 23, 24, 25]. Accordingly, in this doctoral thesis, different types of Internet traffic have been analyzed through ML and DL solutions, such as the traffic generated by privacy-preserving solutions (e.g., VPNs, Anonymity Tools), network attacks or malware (e.g., Scans,

Denial of Services, Botnets), and mobile applications (e.g., Games, Social Networks, Video on Demand).

1.1.2 Hierarchical Learning

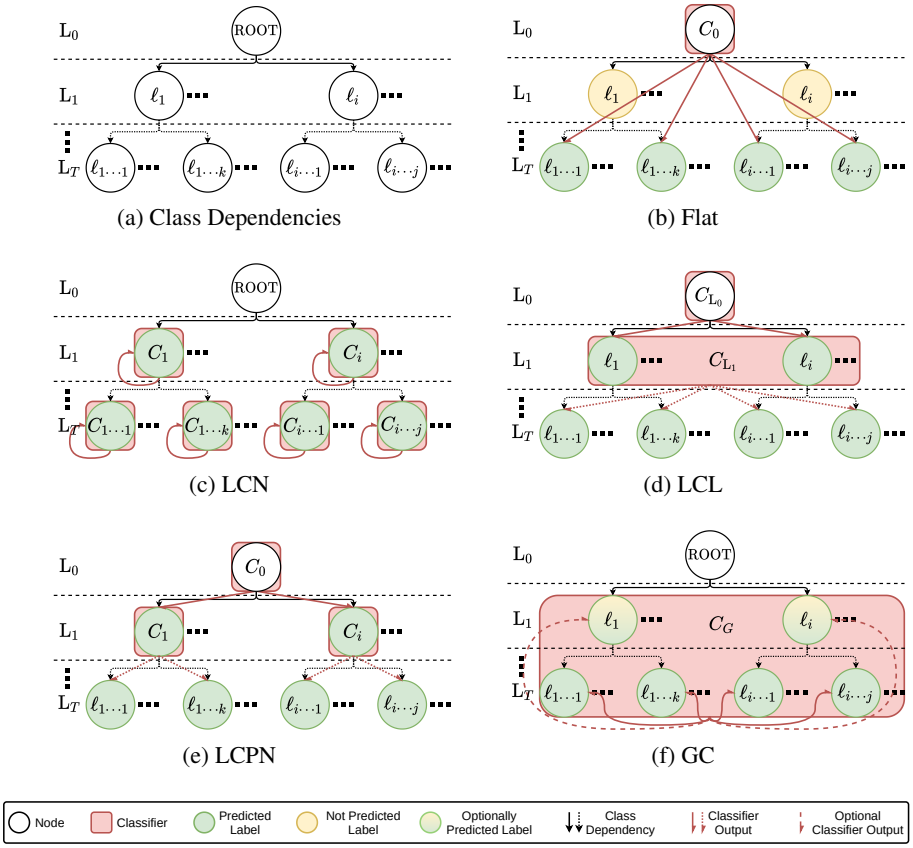


Figure 1.1: Hierarchical Learning Taxonomies.

In this section, the taxonomization of existing approaches for hierarchical learning is provided, which is based on the way the (hierarchical) dependencies among traffic classes are exploited. Used notations are detailed in Tab. 1.1. In Fig. 1.1a is reported the (hierarchical) dependencies tree. In detail, the classes object of classifications are arranged in a T-levels tree, from the ROOT to the leaf nodes at level L_T, e.g, node $\ell_{i\cdots j}$. Moreover, the level L_i has L_i nodes, and

the node ℓ_i has $\mathcal{L}_i$ child nodes and $\mathcal{S}_i$ training samples. It is worth noting that subscripts of labels represent the list of ancestors starting from L_2, e.g., $\ell_{i\cdots j}$ has the oldest (coarsest-granularity) ancestor in ℓ_i, and ending with the actual label index, e.g., $\ell_{i\cdots j}$ is the j^{th} child of its parent node. It is worth noting that C_* is the notation for classifier nodes.

Table 1.1: Table of notations.

Symbol	Meaning
T	Number of levels in the hierarchy.
L_0	ROOT node/level.
$\mathrm{L}_i\|_{0<i<T}$	i^{th} intermediate level.
L_T	Leaf level.
ℓ_{ijk}	k^{th} child node of the ℓ_{ij} node.
L_i	Number of nodes of the L_i level.
$\mathcal{L}_i$	Number of children nodes of the ℓ_i node.
$\mathcal{S}_i$	Number of training samples of the ℓ_i node.
$C_{i\cdots}$	Classifier node which the same subscript notation as $\ell_{i\cdots}$, used for Flat, LCN, and LCPN approaches.
C_{L_i}	Classifier node at the level L_i, used for for LCL.
C_G	Global classifier.

Hierarchical learning approaches usually fall into three main categories [26]:

- *Flat Classifier (FC)*, i.e. hierarchical dependencies are not considered at all, resorting to a single classifier coping with leaf classes; from Fig. 1.1b is shown a single classifier C_0 which distinguishes among classes belong L_T, viz. leaf nodes.
- *top-down classifier*, where a set of local classifier are trained and the final solution is obtained by solving sub-problems (i.e. *divide-et-impera*); this category translates into three solutions, differing by location into the hierarchy of dependencies in which local classifiers are placed (viz. in which way classes are grouped w.r.t. hierarchical dependencies). These solutions are:
  - *Local Classifier per Node (LCN)*, where a binary (viz. one-vs-rest) classifier per node/class is trained, e.g., in Fig. 1.1c, $C_{i\cdots j}$ classifies traffic into $\{\ell_{i\cdots j}, \overline{\ell_{i\cdots j}}\}$.

  - *Local Classifier per Level (LCL)*, characterized by a multiclass classifier per level, e.g., in Fig. 1.1d, C_{L_1} classifies traffic into $\{\ell_{11}, \cdots, \ell_{1n}, \ell_{i1}, \cdots, \ell_{im}\}$.
  - *Local Classifier per Parent Node (LCPN)*, which exploits a multiclass classifier per parent-node, e.g., in Fig. 1.1e, C_i classifies traffic into $\{\ell_{i1}, \cdots, \ell_{im}\}$.

- *Global Classifier (GC)* (viz. big-bang), where a single model is trained by considering the hierarchical dependencies among classes, i.e., in Fig. 1.1f the C_{G} provides dependencies-aware classification of traffic into both leaf and (optionally) non-leaf nodes.

Subsequently, details for each category are provided, focusing only on the LCPN approach for the top-down classifier category.

Flat Classifier

The FC consists of a single model trained on the finest granularity classes. This approach is the classical way NTA is actually carried out by several works, with no dependencies between classes imposed to the learning process. It is worth noticing that the FC could be seen as a LCL approach, where only the model built on the last level is used: practically the FC is a naïve solution to classification inconsistencies by which the LCL approach is affected (viz. verdicts at different levels may mismatch existing, valid branch). Natively, the FC enables the inference at higher hierarchy levels, by simply following up the hierarchical dependencies of predicted classes.

Top-down Local Classifier per Parent Node

The LCPN is a top-down classification approach that makes heavily adhere the classification model to the dependencies among traffic classes. In few words, the LCPN results in the deployment of multiple multi-class classifiers, each one corresponding to a parent node in the hierarchy and distinguishing among child classes. The LCPN is the most intuitive approach when hierarchical dependencies among traffic classes are exploited. It brings some native advantages, mostly linked to its *divide-et-impera* nature, like the locality of design choices, i.e. the selection of algorithms, inputs, and optimizations is locally-performed. Moreover, if the hierarchical dependencies reflect on the pattern of involved classes (considering NTA, when the hierarchy reflects communication behaviour of traffic classes), classification errors of the LCPN should fall

within similar classes. Despite this, the LCPN approach suffers from the propagation of error throughout the hierarchy, that could be mitigated via a *censoring mechanism* that practically enforces the *non-mandatory-leaf-prediction* paradigm, by stopping the propagation to lower level classifier when the confidence of the classification of a sample is below a certain threshold (i.e. *reject option*).

It is worth to note that among advantages of the application of the LCPN approach, like scalability and easier (re-)trainability, its privacy preserving nature is of utmost importance: multiple levels of classification enable a distributed deployment of such solutions, with the ideal near-user-adoption of sensitive data for modeling.

Global (or Big-Bang) Classifier

The GC approach, also known in literature as *big-bang approach*, is a category of techniques that builds (viz. trains) a single model encapsulating the knowledge about dependencies among classes. As detailed below, each technique is characterized by the way this knowledge is exploited, e.g., via the loss function. Naïvely, a GC could be seen as a classifier which predicts each label of the hierarchy, resulting in the association of each sample to a class per level. Moreover, another approach can be defining an extended set of classes based on the level-wise combination of all the labels, by obtaining an extended classification output which pushes down malformed dependencies (branches). Moreover, it is interesting how the latest proposed advancements in classification models for NTA are exploitable for training a GC, i.e. task-incremental learning and multitask learning. Finally, recently has been proposed a hierarchical loss to capitalize dependencies without modifying the underlying model. To the same extent, contextual label smoothing is another way to obtain a GC, where dependencies are enforced by modifying the one-hot-encoded target accordingly to the hierarchy.

1.1.3 Related Works

In this section we report the literature which is strictly related to our proposed framework. In detail, we first present a wider analysis of the hierarchical learning solutions proposed in the literature, with the focus on NTA applications. Because the literature analysis we carried out shows no application of Big Data (BD) frameworks in order to enhance hierarchical learning approaches, we report works deal with NTA applications by leveraging BD technologies to train

"flat" (viz. non-hierarchical) classification models.

Hierarchical Learning for Network Traffic Analysis

The majority of (non-NTA) proposals for hierarchical learning come from fields like protein sequences classification, object detection, image classification, patient diagnosis prediction and classification, and text recognition. Among the selected works, the LCPN solution (which is the most intuitive enforcement of hierarchical dependencies in the learning process) is the broadest applied, but several solutions that exploit advancements of ML (i.e. *task-incremental learning*, *multi-task learning*, *hierarchical loss*, *contextual label smoothing*) in order to obtain GC solutions are also proposed. About LCPN, an abundance of interesting and recent works are available, like [27, 28, 29, 30, 31, 32, 33]. Among these, some leverage an LCPN architecture to address feature selection in a hierarchical fashion, and others directly tackle the classification task. Furthermore, focusing on GC approaches, several attempts are done for task-incremental [34, 35], and something extra for multi-task [36, 37, 38]. Noteworthy, a hybridization of LCPN and GC approaches is proposed in [39], but this results in a trivial post-processing of the LCPN output.

Focusing on NTA-related tasks, the hierarchical modeling approach attracted the attention of networking researchers from the beginning, due to the intrinsic hierarchical nature of Internet traffic classes: just think of the classical chain `(TRANSPORT) PROTOCOL / SERVICE / APPLICATION`, where given an application a trivial branch of labels is easily recognizable. Interestingly, most of works which try to solve NTA tasks via hierarchical approaches falls into a confined set of these, where the LCPN is the most in vogue and limited attention is posed only to LCL and GC solutions. Noteworthy, the application of hierarchical learning approaches to NTA ranges within a broad traffic categories, such as network attacks, Peer-to-Peer (P2P), video, Virtual Private Network (VPN), Anonymity Tools (ATs)-related, and mobile applications. It is worth to underline that a per-paper analysis is subsequently performed by sorting related works from the oldest to the newest, with the exception for works from the same authors which are grouped (i.e. the oldest work places the group).

Zhang et al. [40] were among the first to propose a hierarchical Intrusion Detection System (IDS) based on Radial Basis Functions. In particular, they propose (and compare) a Serial Hierarchical IDS (SHIDS) and a Parallel Hierarchical IDS (PHIDS) solutions. In detail, SHIDS is composed by

pipelined modules, each performing single-attack detection (apart from the first implementing Anomaly Detection (AD)), which is something similar to the LCN approach. Differently, PHIDS implements a three-stage Intrusion Detection (ID), made of AD, coarse-grain and per-category Attack Classification (AC), resulting in an LCPN solution where the benign branch is no further explored. Results over KDD'99 dataset show improvement of Radial Basis Functions against backpropagation learning model, and performance gains in using PHIDS over SHIDS (viz. LCPN performs better than LCN).

Similarly, Khan and Khan [41] propose a two-stage LCPN architecture composed by a first level performing Misuse Detection (MD) and a second level tackling AC. The proposed approach is compared with a flat multi-class MD, showing higher True Positive Rate (TPR) and lower runtime. In [42], the same authors propose a two-stage DL-based model, where each node is a Deep AutoEncoder (DAE). In detail, the first stage implements binary MD and the second stage implements multi-class MD. Interestingly, the first level soft output is used as an additional feature for the second level. The devised model, tested on both KDD'99 and UNSW-NB15 datasets, achieves 99.99% and 89.13% accuracy, respectively.

In [43] authors present a three-stage LCPN ML-based model. They conduct a three-stage binary MD, notably including benign instances in the same class of challenging attacks at stage 1 and 2. Results on KDD'99 dataset show poor overall performance, but the proposal gains on challenging attacks.

Among the first in applying an LCPN hierarchical approach to Traffic Classification (TC) were [44]. In their paper the authors focused on the classification of network traffic by imposing a three-level hierarchy for P2P traffic, falling in the binary classification P2P vs. non-P2P traffic at the first level, in the classification of P2P traffic types at the second level (i.e. file-sharing, messenger, and TV), and in the application classification (11 P2P and 5 non-P2P) for P2P traffic type and for non-P2P traffic at the third level. The classification is performed at biflow level, leveraging 39 statistical values post-mortem-collected and achieving precision and recall $\geq 95\%$ in P2P/non-P2P recognition and $\geq 93\%$ in P2P-type classification, while a recall drop down to 71% is observed at the last level. However, results lack in comparison with other (non-hierarchical) learning approaches, focusing on the sensitivity of the per-node feature-selection.

Following a similar approach, in [45] the authors propose a four-level LCPN-based hierarchical approach, tackling the classification of Internet applications. The proposal is composed by: the first layer performing a binary

classification of known and unknown traffic; the second layer which performs protocol identification by extending the known traffic branch; the third layer which extends P2P and HTTP protocol branches to catch the generating application; the fourth layer being dedicated to the Video applications, which are subdivided into four video application classes. Both (per-classifier) feature selection and (coarse-grained) optimization, by evaluating different classifiers such as Decision Trees (DTs), Neural Networks (NNs), and Support Vector Machines (SVMs), are considered. Results show a fair comparison between hierarchical and nonhierarchical approaches, electing the hierarchical as the best option in terms of classification accuracy and computational complexity, mainly due to the per-node model optimization.

In the same year, [46] proposed a *multilateral* (viz. multi-view) and hierarchical learning approach, further refined into *FORMULA* framework [47], operating on biflow-segmented traffic. Authors defined multiple taxonomies resulting in multiple LCPN solutions, one for each taxonomy. Taxonomies identified are four, namely service, application, protocol, and function, and each taxonomy is divided into at most three levels, e.g., the service taxonomy is divided into service type, service name, and provided service levels. Each combination of taxonomies' branches is associated with a particular traffic type, which is the target of classification. Results are strictly related to the proposed solution and no comparison with other learning approaches is conducted. Despite this, the proposal shows high explainability of user activity.

Recently, in [48] a two-level hierarchical classification framework is presented to identify the services running within HTTPS connections leveraging a set of features robust to alteration (e.g., statistics of interarrival time and packet/payload sizes). The proposed evaluation method, based on real traffic traces, achieves a recall within $[95, 100]\%$ in 50 out of the 68 HTTPS services considered.

To a similar extent, in [49] authors propose a two-level hybrid (viz. performing both AD and MD) IDS based on LCPN. The first level performs AD while at the second level traffic declared as benign (resp. anomaly) is further inspected by a MD (resp. another AD) module, to catch challenging attacks. In detail, at the first level, the whole traffic is inspected by an AD module. Differently, at second level, traffic declared as benign (resp. anomaly) is further inspected by a MD (resp. another AD) module, to catch challenging attacks. Finally, anomaly/anomaly or benign/attack verdicts identify anomalies. Results on KDD'99 show that it outperforms state-of-the-art hybrid solutions, reaching 91.86% True Positive Rate (TPR) and 0.78% False Positive Rate (FPR).

Differently, [50] propose a two-level LCPN ML-based ID method that performs binary MD at first level and AC at second. The novelty of the approach is in the treatment of categorical (resp. numerical) features are used to feed the MD (resp. AC) module, practically feeding different levels of the hierarchy with diverse information. Evaluation on NSL-KDD dataset results in 96% accuracy.

The LCPN approach is also evaluated in [51] to classify Internet video traffic. In particular, the authors design and implement a two-level hierarchy, using k-Nearest Neighbors (k-NN) models, where the first level splits networking flows into asymmetric and symmetric, and then the second level classifiers infer the application generating each flow, with the asymmetric branch classifier distinguishing among Asymmetric Standard Definition Video, Asymmetric High Definition Video, and HTTP Download Video, and the symmetric branch classifier within QQ, Xunlei, and Sopcast. Also in this case, the authors perform a post-mortem classification, extracting 40 features for each networking flow, and selecting the top 4 of them via feature selection. Finally, the authors perform a wide comparison of their proposal against several ML models, like Random Forest (RF), DT, and SVM, the latter leveraged also in a hierarchical learning fashion, showing the superiority of their proposal. In detail, experimental results report $\geq 97\%$ F-measure in discriminating among all the considered (video) traffic and superior performance w.r.t. existing alternatives, while providing only a slight time complexity increase.

More recently, Schueller et al. [52] have proposed a hierarchical two-level IDS, where the first level performs a (flow-based) AD task leveraging a SVM, while the second one conducts (packet-based) AC via SNORT. Their proposal is validated on DARPA IDS dataset.

Beyond the evaluation of "pure" LCPN approaches, also some hybridizations with other techniques are proposed in literature, like the Chaining approach [53]. In this paper, the authors afforded the NTA of network video traffic, suggesting an approach that is the combination of a Chaining approach with LCPN. In detail, they propose a cascade of binary classifiers, where some results in a further multi-class finer grain classification (like LCPN). They identified 7 video categories, namely Asymmetric Standard Definition Video, Asymmetric Ultra Clear Video, Asymmetric High Definition Video, HTTP Download Video, Interactive Video Communications, P2P Video Share (P2P_video), and Internet Live Video. The cascade of binary classifiers provides classification outcomes, in order, Interactive Video Communications, HTTP Download Video, Video On Demand, Internet Live Video, and

P2P_video classes. It is worth noting that the Video On Demand category includes Asymmetric Standard Definition Video, Asymmetric Ultra Clear Video, and Asymmetric High Definition Video categories, which are classified by a local classifier which resides in the Video On Demand parent node. The NTA they propose is postmortem, leveraging 40 statistical features extracted from the fan-out of a target host. The authors compare their proposal with other learning approaches, like LCPN, Chaining, Bagging, and RF, showing the clear advantage by using their hybrid solution in terms of classification F1 Score.

Other than LCPN, less attention is paid to others hierarchical approaches, like LCL and GC. In the following, a couple of examples are provided. [54] proposed a LCL solution to classify the traffic generated over the Tor overlay. Their proposal enforces a multiclass classifier at each level of the hierarchy, resulting in 3 models. The first level splits the network traffic into Tor and Non-Tor, then the Tor branch is further divided into Traffic Types at level two, and into generating applications at level three. Authors focused on the classification of networking flows leveraging a post-mortem feature set. The proposal is not evaluated against nonhierarchical approaches or other hierarchical, but it is limitedly compared by varying the underlying models and the number of selected features.

Recently, [55] propose a GC approach based on multi-task learning, named Multi Task Hierarchical Learning, which is evaluated by solving malware detection (Stratosphere IPS & CICIDS2017 datasets) and application classification (VPN-nonVPN2016 dataset) tasks. Their proposal is a first attempt that exploits hierarchical relations between traffic classes in order to train a multi-task model, which is capable of classifying networking flows into a top-level (viz. coarse-grain) label and a mid-level (viz. fine-grain) one. In detail, the neural network they propose is formed by a backbone, which is shared among tasks and is connected with two task-specific branches. Each task classifies into top-level and mid-level labels. Results show the superiority (limited to the CICIDS2017 dataset) of Multi Task Hierarchical Learning against other ML models, like RF, k-NN, SVM, and MultiLayer Perceptron (MLP), when performing fine-grain predictions. However, the comparison the authors performed is not fair, because the DL model underlying Multi Task Hierarchical Learning is very complex w.r.t. experimented flat models, and it is not compared against the non-hierarchical version (viz. flat) of the proposal itself.

Big Data-enabled Network Traffic Analysis

Noteworthy, because the literature analysis we carried out shows no application of BD frameworks in order to enhance hierarchical learning approaches, we summarized in Tab. 1.2 (for completeness) works tackled NTA by adopting BD technologies, by providing their comparative overview along *multiple key aspects*, highlighted by the corresponding columns.

Still, the novelty of our framework lies in addressing the challenging integration of both models (characterizing the LCPN approach) and data parallelism (due to their interplay), aiming at a sophisticated and highly-effective NTA system. On the other hand, in the literature, the application of (manifold) *BD technologies* to NTA tasks is significantly represented as well by recent works proposing approaches to exploit advantages of distributed computing in NTA. In particular, they adopt BD technologies to define distributed ML models to enhance scalability or classification performance, and to meet real-time analysis requirements. By investigation of all above works, the most used BD technologies result to be Apache Hadoop and Apache Spark.

Additionally, the works in Tab. 1.2 focus on different kinds of *Traffic Types*, including web services (e.g., Facebook, Gmail, Skype, and Google), network attacks (i.e. Distributed Denial of Service (DDoS)), and typologies associated to different contexts like mobile applications. The considered *Traffic Objects* include, flows and biflows. In addition, several BD-related works focus on the finest granularity, i.e. packets. Concerning *Input Data*, all the reviewed approaches feed the classifiers with different sets of statistical features of the considered Traffic Objects (TOs). The most common features are related to the inter-arrival time, byte count, TCP flags, packet count, and payload length.

Also, classifiers are mostly common between the two approaches that leverage state-of-art ML models, like SVM, DT and related evolution, and NN. Furthermore, more than half of the reviewed works rely on a *private dataset*, thus precluding further comparison and advancements. The only exceptions are few works releasing only a part of their considered traffic data [56, 57]. Analogously, a significant share of reviewed works provides enough details for *reproducible implementation* of the approach proposed therein, e.g. [58, 59, 60].

Finally, in the reviewed works no cost analysis of real scenario deployment is provided, despite this gives a clear view about the trade-off with inference performance, looking for the optimization of a distributed deployment. These findings highlight the need for delving into the problem via successive splits and devising a *more sophisticated classification framework* for harder classifi-

cation tasks.

Table 1.2: Summary of previous works on BD-enabled TC.

BD Tech.	TO	Input Data	Traffic Type	ML Model	Open Dataset	Reproducible implementation	Paper
H	F	7 statistics (PTs, PR, PC, FL, DUR)	Applications	SVM	●	◐	[58]
S	P	30 packet fields (TTL, PR, FLGI, CHK, etc.)	DDoS vs. Normal	GA	◐	◐	[61]
H	F	6 statistics (BC, and PS at different ISO/OSI layers)	Applications	DT	●	◐	[62]
S	F	100 Tstat metrics	Web Services	Proposal	○	◐	[59]
H	P	TS, SRC, DST, PR and others header info.	DDoS vs. Normal	—	○	◐	[63]
S, I	B	PL for the first 5 packets, per direction PC, and BC, and PTs.	Web Services	GBT, RF, SVM, and NN	○	○	[64]
SS	B	248 statistics (PTs, IaT, BC at different ISO/OSI layers, per direction PC, FLGT, WS, IaT, RTT, and DA, etc.)	Applications	PM	◐	◐	[57]
S	P	49 (12 after selection) of packet fields (SRC, DST, PR, TTL, PT, etc.)	DDoS vs. Normal	NB, DT, and RF	●	◐	[65]
S	B	(PL, IaT, PT, FLGT) for the first 20 packets, Payload of first 784 bytes	Mobile Apps	CNN, LSTM	◐	◐	[56]

Legend of acronyms (— when not applicable):
BD Tech.: H (Apache Hadoop), I (IBM InfoSphere), S (Apache Spark), SS (Apache Spark Streaming);
TO: B (biflow), F (flow), P (packet);
Input Data: BC (Byte Count), CHK (IP checksum), DA (duplicate ACK flag), DST (destination IP), DUR (duration), FL (flow length), FLGI (IP flags), FLGT (TCP flags), IaT (inter–arrival–time), PC (packet count), PL (payload length), PR (protocol), PS (packet size), PT (port), RTT (Round Trip Time), SRC (source IP), TS (timestamp), TTL (time to live), WS (window size);
Traffic Type: DDoS (Distributed Denial–of–Service);
ML Model: CNN (Convolutional NN), DT (Decision Tree), GA (Genetic Algorithm), GBT (Gradient Boosted Tree), NB (Naïve Bayes), NN (Neural Network), PM (Pattern Matching), RF (Random Forest), SVM (Support Vector Machine);
Open Dataset: ○ (W/o publicly available dataset), ◐ (At least one publicly available dataset), ● (W/ publicly available dataset);
Reproducible implementation: ○ (W/o details for reproducibility), ◐ (W/ details for reproducibility), ● (W/ publicly available implementation).

1.2 Positioning of the Proposal

In this section the proposed framework is positioned with respect to existing challenges in NTA (Sec. 1.2.1), delving into the rationale behind the application of hierarchical learning to Internet traffic (Sec. 1.2.2), staring from the interleaving with BD technologies, and deepening the specific use cases we identified to validate the proposal.

1.2.1 Open Challenges

In addition to the challenges mentioned in Sec. 1.1.1, which drove the transition from previous modeling solutions (port- and payload-based) for traffic analysis to more advanced and privacy preserving ML- and DL-based solutions, we identified other open challenges which affect state-of-the-art solutions when applied to *fine-grained* traffic analysis in nowadays networks. These include two main challenges which are strictly linked to the growing adoption of Internet services: (i) the huge number of Internet enabled devices which generates *heterogeneous network traffic*, which hampers the acquisition of *fine-grained network knowledge*; and (ii) the increasing of the generated traffic in terms of *traffic volume*, which demands for *scalability of traffic analysis solutions*.

In detail, the number of Internet-enabled devices grows, just think to the number of Internet of Things (IoT) devices—that was anticipated to be over 7 billion in 2018, with a 3-fold increase expected by 2025, taking into account both consumer and industrial uses[1]—and to the widespread adoption of mobile devices—according to Ericsson mobility report[2], the number of smartphone subscriptions is expected to reach 7.7 billions by 2027, with a corresponding $+28\%$ predicted compound annual growth rate (viz. from 63 EB/month in 2021 to 281 EB/month in 2027) of traffic generated by smartphones. Moreover, IoT devices are frequently characterized by a low-cost manufacturing process (including hardware and software design decisions, such as unsecured network services, dangerous update mechanisms, and obsolete components) and insufficient user attention to setup. As a result of the many and major flaws in IoT devices, the primary goal of daily-released malware has changed to infecting IoT services.

In addition, the capacity (viz. available bandwidth) and coverage of network links with the raise of new communication technologies (e.g., 5G) also

[1] `https://tinyurl.com/iot-dev-2018`

[2] `https://tinyurl.com/ericsson-report-2021`

growth (always according to Ericsson, with 4.4 billions regarding 5G mobile subscription by the 2027, with a CAGR of $+37\%$ w.r.t. 2021), clearly putting pressure on NTA systems and posing additional problems in implementing effective and efficient solutions. As a result of this development, highly scalable systems must be designed and deployed to allow for a realistic (time-constrained) fine-grained analysis of massive volumes of heterogeneous network traffic.

1.2.2 Exploiting Hierarchical Solutions

In this doctoral thesis, in order to enhance the *fine-grained network knowledge* and the *scalability of traffic analysis solutions*, we propose a Hierarchical Learning Framework for Network Traffic Analysis. The framework enhances traffic analysis exploiting hierarchical dependencies among network traffic classes in order both to improve the fine-grained modeling of network traffic and to enable a modular and scalable learning process, enabling fast retraining. Furthermore, the adoption of hierarchical approaches allows for fine-grained performance gains, by splitting the NTA tasks in sub-problems. Equally important, although hierarchical approaches may result in increasing training complexity, top-down category can leverage *model parallelism*, due to the scalability and modularity of the resulting approaches. As a result, hierarchical ML-based NTA has recently appealed to the scientific community [45, 51, 66].

It is worth to underline that in this thesis we enforce the entire set of hierarchical approaches introduced in Sec. 1.1.2 (i.e. LCPN and 7 GCs) by leveraging (classical) ML algorithms (e.g., RF). However, despite the (re-)designing of existing (classical) ML algorithms is potentially feasible [67], the global classifier category of approaches is considered only for Neural Network models, except for a global solution (viz. *naïve global classifier*, Sec. 2.3.3) that can be applied also to (classical) ML algorithms.

Therefore, in this thesis both ML and DL hierarchical solutions are evaluated on three use cases, namely *classification of anonymity tools*, *intrusion detection for Internet of things devices*, and *classification of mobile applications*, respectively falling in Privacy, Security, and (fine-grained) Traffic Management applications.

Interleaving of Big Data Technologies and Hierarchical Solutions

The proposed framework suitably manages and capitalizes BD technologies. In this direction, an initial effort has been put forward by the scientific community and industry to apply BD technologies, e.g. Apache Spark or Apache Hadoop, to ML-based NTA [58, 62, 64], exploiting *data parallelism*. As NTA becomes more and more challenging, benefiting from both *model* and *data* parallelization approaches is of clear appeal, but none of the two is trivially applicable in itself. On one hand, model parallelism requires accurate architecture planning to fit the specific (classification) problem to reap the potential benefits in terms of classification effectiveness and design advantages. On the other hand, data parallelization via the adoption of BD does not represent a transparent enabler, as it may imply classification performance degradation, trading efficiency for effectiveness [56]. Accordingly, their combination is far from being trivial, as their interplay is not known a priori [68]. Nevertheless, jointly leveraging model and data parallelism is extremely promising to accommodate the needs arising from recent scenarios in computer networks that call for tools for processing huge amounts of data produced by heterogeneous devices (e.g., as those generated by IoT platforms) in a timely manner and at a predictable cost (e.g., leveraging cloud or fog platforms [69]). While the separate adoption of either model or data parallelization has been investigated to a certain extent, to the best of our knowledge their combination has not been explored in NTA literature. Based on the above motivations, the proposed framework boasts the benefits of both model and data parallelism and is able to provide the appealing characteristics of modularity, scalability, and fast retraining, which make it suitable for working with traffic of today's (and next-generation) networks. To meet the above desiderata, we investigate the interplay of model and data parallelism and evaluate their interaction along multiple dimensions. As a result, this thesis paves the way to a novel approach for designing NTA algorithms.

Hierarchical Solutions for Privacy

Because of the specific nature of AT encrypted traffic, ML classifiers represent the natural enabler, able to provide decisions based on the sole traffic-flow features [5] and to overcome shortcomings due to the application of common solutions (e.g., those based on payload inspection or earlier port-based ones [70]). It is worth to note that AT classification is something similar to website fingerprinting, because anonymous services are often reached by means of webpages. Moreover, looking into the TC of ATs is beneficial to designers since

it puts their efficacy to the test, identifies flaws, and points the way to making them more robust. These studies, on the other hand, are of importance to both providers and government agencies, since they give knowledge that may be used to enforce informed engineering regulations or prohibit users from conducting undesired acts. As a result, categorizing ATs traffic is a highly intriguing and hard research subject, with current methods that may be improved to obtain fine-grain knowledge of the (anonymous) generating application. The fine-grained traffic analysis is useful to enforce fine-grained network management solutions. In detail, ATs have been investigated in recent years by several studies from *different perspectives* including design improvement, AT delay and performance analysis, feasibility of effective attacks to ATs, users' behavior profiling and identity disclosure risk, and censoring policies enforced for ATs [13]. Among these crucial aspects, a cardinal issue is *understanding to what extent (encrypted) ATs traffic can be classified*, i.e. at which granularity ATs and related applications can be accurately recognized by external entities. On top of that, hierarchical classification represents a perfect match for TC of ATs, as (i) it allows fine-grained tuning and design, potentially leading to classification performance gains; (ii) it also brings a number of *"practical" benefits by design*, at cost of moderate complexity increase. For example, retraining does not involve all the nodes in the hierarchy when new applications leveraging anonymity networks are released. In addition, distributed deployment of TC tasks, thanks to the modularity of the framework, is enabled in the network (thus hierarchical classification could be achieved through chaining of *virtualized network functions*, each associated to a classifier). Albeit these benefits are granted by the hierarchical approach itself, research efforts are needed to deepen the aspects of design optimization (to obtain enhanced performance) and fine-grain evaluation to delve into privacy-level assessment of ATs.

Hierarchical Solutions for Security

ML and DL have been also applied with good results to Network Intrusion Detection Systems (NIDSs) design [40, 41]. Since IoT is highly dynamic from multiple viewpoints (e.g., the number and variety of devices, their spatial distribution, and the evolution of attacks) an IoT-tailored NIDS must cope with these challenges, and its design is expected to address yet unknown attacks, while retaining high efficiency to be deployed also onto on a massive number of resource-constrained devices. We remark that privacy issues are also present, with IoT devices likely deployed in domestic and other highly-

sensitive contexts. In detail, to fulfill their goals NIDSs may implement two main approaches: AD or MD, aiming at capturing any deviation from the profiles of normal activities, or identifying the patterns of known attacks, respectively. Indeed, ML-based intrusion detection has been widely adopted in last years, with researches investigating both AD [40, 49, 52], trained only on benign traffic (i.e. anomalies are identified as outliers), and MD [41, 43, 50], trained on both benign and malicious traffic [10]. MD could be binary (benign vs. malicious) or multi-class (benign vs. specific attacks). Further, several approaches fall within AC, where a preliminary phase, skimming benign events, is assumed. Also, a number of proposals for network intrusion detection in IoT environments can be found in literature [71, 72, 73, 74, 75]. Differently from our use case, IoT-aware IDSs in literature do not use an IoT dataset for validation, or target AD or MD separately. Also, the most related proposal [73] discriminates only among known attacks (i.e. no ability to detect unknown attacks). In this thesis, we evaluate a hybrid approach targeting both at the same time: the framework can be configured to obtain a hierarchical intrusion detection approach tailored for the demands of IoT scenarios.

Hierarchical Solutions for Traffic Management

Regarding the mobile application traffic, because the mayor carrier is the HTTPS protocol (port 443) which provides no hints for port-based approaches and takes advantage from encryption, ML and DL solutions have been successfully applied for NTA purposes [16]. However, the growing adoption of mobile devices which is changing the type and composition of traffic traversing our network by introducing a variety of content and services over the Internet, is posing an important challenge to NTA. As example, according to the latest Ericsson mobility report [76], between Q3 2019 and Q3 2020, mobile data traffic grew 50%. This increment is driven by both the growing number of smartphone subscriptions and by the increasing average data volume per subscription, impacted foremost by the fruition of video content. It is forecasted that the share of *video traffic*, that nowadays accounts for the 66% of all the data generated by mobile devices, will increase to 77% in 2026. Along this direction, enforcing hierarchical dependencies when modeling mobile network traffic, by properly exploiting its nature (e.g., by separating traffic by content, like video/non-video, then by category of service, and last by generating application), can introduce both classification performance gains at finer granularities and enhancements in terms of scalability of solutions.

1.3 Summary

This doctoral thesis is composed of two main parts, namely:

Part 1 **– Design and Implementation**
This part introduces design choices which characterize the proposed Hierarchical Learning Framework for Network Traffic Analysis, by formalizing hierarchical learning approaches and their translation with recent advancements in ML and DL, and providing a focus on BD-enabled solution for classification. Therefore, implementation details are provided to foster reproducibility for the evaluated use cases.

Part 2 **– Use Cases**
This part validates the proposed Hierarchical Learning Framework for Network Traffic Analysis effectiveness through three use cases strictly related to Privacy, Security, and Traffic Management tasks, by providing a clear understanding about performance improvements in terms of classification accuracy due to the exploitation of hierarchical dependencies in various ways, and providing a deepening which regards the scalability-related improvements introduced by our framework.

Conclusions and future perspectives end the dissertation.

Part I

Design and Implementation

Chapter 2

Hierarchical Learning Framework for Network Traffic Analysis

In this chapter we present the Hierarchical Learning Framework for Network Traffic Analysis we designed, implemented, and evaluated. In detail, a top-down solution (viz. local classifier per parent node) for hierarchical learning along with the translation of recent advancements in Machine Learning (ML) (e.g., multitask learning) are formalized in order to enforce class dependencies in the modeling phase, resulting in 8 hierarchical learning approaches. Moreover, we design the Big Data (BD)-enabled training for the top-down hierarchical solution to obtain a speedup of this phase.

From a high-level perspective, the proposed framework is composed of three main parts:

- *Traffic Segmentation Component* – which aggregates network packets with respect to the selected traffic object, practically performing preprocessing for raw data flowing the network;
- *Features Extraction Component* – which extracts profitable information from Internet traffic, fixing a traffic object, in order to feed the engine;
- *Hierarchical Learning Engine* – which performs classification, supports several Hierarchical Learning approaches, and provides structured data for evaluation. It is worth to underline that the Hierarchical Learning Engine is designed to be extensible in terms of supported ML and Deep Learning (DL) models.

Accordingly, in Secs. 2.1, 2.2, and 2.3 a detailed per-component description is provided, focusing on *traffic objects*, *traffic features*, and *hierarchical learning solutions*, respectively. With respect to the latter, a deepening about *big data-enabled training* is provided. Finally, in Sec. 2.4 we provide details about the implementation of such proposal by deepening technological-related aspects.

2.1 Traffic Segmentation Component

When dealing with raw network traffic, the earliest step to perform is the *traffic segmentation*, which starts from a sequence of raw network packets and applies on top of them arbitrary aggregation rules, in order to obtain Traffic Objects (TOs) (also known as *traffic views*). In the proposed framework, this task is accomplished by the Traffic Segmentation Component, which takes in input raw traffic and outputs a collection of TOs. Accordingly, the definition of a specific TO determines how raw traffic is segmented into multiple discrete traffic units [2], based on a criterion dependent on the intended application of the classification results. Therefore, the selected TO is practically the unit at which Network Traffic Analysis (NTA) related tasks are performed, e.g., actions following a traffic classification system can tract traffic on such unit of granularity.

Generally speaking, network traffic is composed of network packets, and such packets, having to traverse the underlying network from a source to a destination, can be characterized based on different information (viz. *routing and state information*) which are carried by them to reach the destination in a proper way. Detailing, network packets are composed of several layers of encapsulation, each identified by a particular protocol header. Each header contains plain (viz. unencrypted) information arranged in arbitrary fields that are specific to the particular protocol, which could be exploited to perform traffic segmentation. However, only the portion of the headers that contains routing and state data are exploitable to properly define a TO, namely (@ each TCP/IP layer):

@ **Network Access Layer** MAC header, which contains per-hop routing information about the physical (viz. network access) layer by providing the physical addresses (viz. *MAC addresses*) of the physical machines placed at extremes of the current hop;

@ **Internet Layer** IP header, whose routing information specify the network

addresses (viz. *IP addresses*) of the (physical) machines involved in the communication and the encapsulated transport *protocol* (e.g., TCP);

@ **Transport Layer** L4 (Transport) header, which contains routing information about the application addresses (viz. transport *ports*) of the source application and of the contacted service, and of TCP information regarding the status of the connection, namely the TCP flags.

In particular, the MAC addresses are only useful to identify traffic which flows between two machines on a single routing hop, e.g., when dealing with traffic collected over a LAN environment, MAC addresses uniquely identify the traffic which flows between two machines. Then, IP addresses could be used to identify endpoints of a communication, thus aggregating packets to obtain the entire trace of the traffic a user generates while interacting with a specific service. Finally, transport ports are useful to aggregate the network packets with respect to the contacted service. In fact, when Internet services used standard port numbers (assigned by IANA), the traffic could be divided into services by simply looking at the contacted transport port. Nowadays, the port number information is used in order to clearly identify TCP/UDP sockets.

Therefore, several *segmentation criteria* exist and can be differentiated basing on the set of selected routing or state information, those leading different TOs, like:

- *Flow*: stream of packets with the same 5-tuple (i.e. source IP, source port, destination IP, destination port, and transport-level protocol), thus taking into account their directions.

- *Biflow*: namely bidirectional flows, it includes flows of both directions of traffic, i.e. given the 5-tuple for flows, biflows are characterize by the *interchangeability* of `(IP address, port)` pairs of source and destination.

- *TCP Connection*: which differs from the biflow only in the initiation and termination heuristics which usually exploit the connection state information contained in the TCP (transport) header.

- *Service Burst*: consists in the aggregation of traffic directed to a single service, viz. packets sharing the same destination IP and port number, by aggregating packets with a time smaller than an arbitrary "burst" threshold.

Among these, flows and biflows are the most commonly used traffic objects [5, 77]. Moreover, it is worth noting that the packet arrival time could be used to further refine the traffic object (e.g., like the service burst definition). To this extent, for both flows and biflows, the termination can be defined based on an arbitrary timeout.

About the Traffic Segmentation Component, it is able to split raw network traffic into several traffic objects, like flows, biflows, TCP connections, and service bursts. Moreover, when required, a multiview segmentation could be obtained, namely providing *composition of traffic objects* in order to perform traffic analysis at multiple views.

2.1.1 About Hindrances

Although the definition of a TO seems to be easily applicable, the relations underlying them, e.g., if they are generated by the same host or peripheral network, or they are sent to the same server, strictly depend on the point of the network at which the raw traffic is collected.

In other words, the presence of NAT disrupts location-related correlations within TOs at various levels. In detail, despite IP addresses are usually modified during the traversal of the network because of the presence of NAT(s), the collection of the raw traffic after the ("multiple") NATTING does not invalidate the separation of raw traffic in TOs like biflows or flows. However, the collection of the raw traffic after the "single" NATTING results in the impossibility of defining host-granularity relations, i.e. one cannot distinguish if a (bi-)flow is initiated by the same host because each host is mapped to the same IP address. A similar drawback is related to the collection after "multiple" NATTING, which hampers (peripheral) network-related relations. On the other hand, when the NAT is enforced at the destination and the traffic is collected before the destination, it is not possible to understand if contacted servers are hosted on the same machine (viz. data center). However, usually the presence of a ("multiple") NAT is not considered, and (in general) the main underlying assumption is that the traffic is collected near to either the source host or the destination host, thus limiting the NAT(s) impact.

Moreover, the collection of biflows in the middle of the network (viz. far from the endpoints) is also tampered by the dynamicity of the routing protocols. In fact, when packets are sent from a source to a destination and vice versa, it is not assured that all the generated packets will follow the same path (viz. traverse the same sequence of routers) for both the directions, i.e. the captured traffic can be either the downstream flow or the upstream flow. De-

spite this, the biflow is still the most informative traffic object enabling the finest grain traffic analysis, because it catches the entire conversation between two hosts providing the identification of possible request/response patterns.

Finally, privacy-related solutions, like Anonymity Tools and VPNs, clearly hampers the capability of defining the hosts at the extremes of a communication, because their action obfuscates the two communicating parts.

2.2 Features Extraction Component

The Feature Extraction Component takes action once the Traffic Segmentation Component has divided the raw network traffic into traffic objects. The feature extraction procedure is strictly dependent on which traffic object has been selected, because the network characteristics are limited by the TO itself. However, all traffic features could be taxonomized by the granularity the statistics are computed.

In general, traffic features could be fine-grained, when refers to the finest granularity obtainable from traffic objects (viz. *packet-level*), and coarse-grained, when one refers to statistics (e.g., mean, standard deviation, variance, summation, minimum, maximum) extracted imposing an arbitrary level of aggregation for network packets belong the traffic object, i.e. considering the entire set of packets (viz. *TO-level*) or applying an (either temporal or spatial and either disjointed or overlapped) aggregation of network packets (viz. *aggregated*). To further dissert, packet-level and aggregated features usually come in the form of *time-series*. On the contrary, TO-level features are practically *statistics* about traffic objects. Moreover, the portion of packets (pertaining the same TO) which is used to compute the features further determines two classes of approaches, namely *early analysis* when the initial (first-n-packets) of a traffic object are employed, or *post-mortem analysis*, when the entire set of packets (in particular the traffic object trailers) are used.

Deepening, the features extraction phase exploits *non-routing information* from packet headers plus some information contained in the transport payload, to assign to each traffic object a set of characteristics (viz. features). From a *packet-level perspective*, the commonly extracted features (grouped in base of the corresponding TCP/IP Stack Layer) are:

- Transport Layer:
  - *payload length*: the number of bytes of the payload;
  - *payload bytes*: the raw bytes composing the payload;

  - *TCP flags*: the values of the TCP header corresponding to the TCP state flags;
  - *TCP window size*: the size of the TCP receive window;
  - *wrong fragments*: a binary value representing if the packets checksum field does match or not the actual checksum;

- Internet Layer:
  - *packet size*: the number of bytes the IP packet is composed of;
  - *interarrival time*: the time which occurs between two consecutive packets arrival instants;
  - *direction*: if the packet is either upstream or downstream transmitted;
  - *time-to-live*: the value of the TTL field in the IP header;

Instead, as abovementioned, TO-level or aggregated features are derived from the just listed features. In detail, we can identify several widely used statistics implicitly referring to the set of aggregated packets, like:

- *payload volume*: being the summation of the payload-lengths;
- *duration*: being the summation of the interarrival times (viz. the difference between the arrival of the last and the first packets);
- *number of packets*;
- *number of wrong fragments*: being the count of wrongly fragmented packets;
- *TCP flags count*: being the per flag count of transmitted TCP flags.

Moreover, also derived features could be defined, like the *byte rate*, which is the payload volume divided by the duration. Noteworthy, the aforelisted features could be practically extracted from all the TOs listed in Sec. 2.1. Moreover, the biflow TO enable for the features diversification in upstream- and downstream-related. Furthermore, for several applications (e.g., DDoS detection, user activity recognition), useful hints could be achieved leveraging statistics on the routing-related information, like the *number of TOs per source IP* or the *number of concurrent TOs*. However, such kinds of information are hampered by the scenarios depicted in Sec. 2.1.1.

Finally, it is worth mentioning that several works rely on routing-information features [24], like the *destination port* and the *protocol*. However, these works implicitly consider port-based or protocol-based analysis hints and can result in biased models.

2.3 Hierarchical Learning Engine

In this Section the Hierarchical Learning Engine is presented. In detail, we first discuss about the methodology underlying the Local Classifier per Parent Node in Sec. 2.3.1, and several Global Classifiers approaches are detailed in Sec. 2.3.3. Then, we present the methodology behind the usage of Big Data technologies in order to enhance the training phase of the proposed framework. Finally, we present state-of-the-art machine learning-based models for traffic analysis, and performance evaluation metrics in Secs. 2.3.4 and 2.3.5, respectively.

2.3.1 Local Classifier Per Parent Node Design

In this section the formalization of the Local Classifier per Parent Node (LCPN) approach included in the proposed framework is presented, as streamlined in Fig. 2.1, whose main components are discussed in the following. LCPN is widely used in the literature [45, 51] and requires a *multi-class classifier* for each parent node in the class hierarchy, trained to distinguish among its children nodes (usually less than L_t, with t being node classification level).

First, it is worth to recall that the proposed classification framework focuses on classes whose relationship can be summarized in the form of a tree (each class has one parent class, *at most*) with T classification *levels* and L_t classes to discriminate from at t^{th} level (whose number increases with the depth), organized in the corresponding set $\mathcal{L}_t \triangleq \{1, \ldots, L_t\}$. The tree structure can be explored with three alternative approaches, as described in Chapt. 1, namely top-down, big-bang, or flat. Here the focus is on LCPN which is a top-down approach. In this case, for each instance to be classified, the LCPN classifier first predicts its first-level (most generic) class, then it uses that predicted class to narrow the choice of classes to be predicted at the second level (i.e. allowed second-level predicted classes are the children of that predicted at the first level), and so on. Although errors at a certain class level could propagate downwards the hierarchy, this choice promotes the *architecture modularity*, which is crucial in NTA, as opposed to big-bang and flat

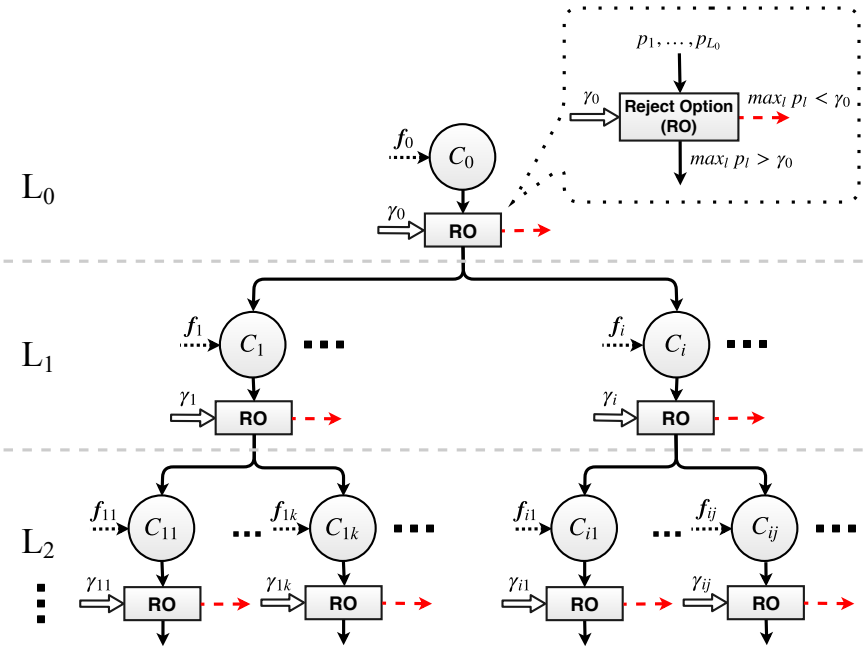


Figure 2.1: Sketch of the designed LCPN classifier.

classifiers.

Fig. 2.1 reports a sketch of the LCPN classifier. We highlight that the indexing of a node reflects the ordered list of its ancestors (except the *root* C_0), e.g., C_{ij} denotes the j^{th} L_2 classifier having C_0 and C_i, as grandparent and parent, respectively. Classifier C_{ij} is in charge of discriminating from $\bar{L}_{ij} < L_3$ classes, grouped within the (training) set T_{ij}, based on the associated prediction probabilities $p_{1,} \ldots p_{\bar{L}_{ij}}$.

The LCPN classifier is trained by *recursively* splitting the training set according to the tree structure. Specifically, the procedure starts from the root classifier C_0 trained using the whole set. On the other hand, each node concurring to $t > 1$ level classification uses a training set corresponding to a subset of $\mathcal{L}_t$, the elements all belonging to the same class at $(t-1)$. Reducing the number of classes per classifier hopefully simplifies the resulting problem and reduces the error scope of flat classification.

In addition, the classifier adopts a *progressive-censoring* (viz. non-mandatory leaf node prediction [26]) policy, accomplished by equipping each classifier node with a "reject option" that censors "unsure" classification out-

comes at intermediate layers (RO blocks in Fig. 2.1). In other words, the reject option forces the classification process to stop for a given instance when a classifier node (e.g., $C_{i,j}$) does not reach a clear verdict, i.e. when the highest class prediction probability (e.g., $\max_{\ell=1,\dots\bar{L}_{i,j}} p_\ell$) is below a threshold (e.g., γ_{ij}). This design choice—already introduced and justified in the mobile context in a flat scenario [17]—here avoids that misclassifications are propagated downwards, at the expense of *coarser-grained* predictions.

By looking at the hierarchy of classifiers reported in Fig. 2.1, it is apparent that its naïvest implementation resorts to the *same* classification algorithm and feature set throughout all the hierarchy. Nonetheless, although hierarchical classification can achieve a potential performance gain w.r.t. a flat approach even in this case (due to the decomposition of the classification task into subproblems), the proposed framework allows for further *optimization* [26]. Indeed, classification performance is expected to improve with more refined implementations, leveraging a *specific selection of features for classification*, and/or using *different classification algorithms at different nodes of the class hierarchy* (e.g., chosen from a pool of classifiers available). Referring to the hierarchical classifier illustrated in Fig. 2.1, one assumes that all classifiers in the hierarchy shall *operate on a common Traffic Classification (TC) object*, although not restricted to a specific one.

As previously explained, different sets of features (of different sizes) can be considered to feed the classifiers in the hierarchical architecture (to achieve accurate classification), as shown in Fig. 2.1. For instance, classifier C_{ij} is assumed to rely on M_{ij} features, collected in the vector $\boldsymbol{f}_{ij}$. The capability of handling a different set of pairs (traffic object, features set) is enforced by the per-node application of aforementioned Traffic Segmentation Component (Sec. 2.1) and Features Extractor (Sec. 2.2) components.

As in the case of set of features, the LCPN classifier allows for a different classifier to be employed at each node (see Fig. 2.1). Therefore, any ML/DL-based (as Internet traffic is majorly encrypted) supervised classifier can be adopted. In other words, the Hierarchical Learning Engine is able to assign a different ML- or DL-based model to each classifier/node which composes the LCPN classifier. For example, referring to the classifier C_{ij}, any ML/DL-based classifiers could be used to discriminate from $\bar{L}_{ij}$ classes within the set T_{ij}. The sole requirement for each classifier is to be able to provide its soft output vector, required by the censoring mechanism.

It is worth to underline that the LCPN classifier supports both these optimization degrees-of-freedom. In case both the per-node classifier and the

feature set are optimized at each node, the combinatorial explosion of the resulting optimization is herein circumvented by a decoupled design resorting to *per-node performance*, e.g., selecting the pair corresponding to the classifier and the number of features ensuring the highest score for the sub-classification problem the classifier node is in charge to solve. Nonetheless, it is worth to remark that an optimization based on complete enumeration or alternative heuristics does not contrast with the hierarchical classification architecture in Fig. 2.1.

2.3.2 Big Data-enabled Training Design

In this section we present the design choices which enable the distributed training by exploiting the LCPN composition of multiple train-independent classifiers. First, we introduce the design of operational workflow and the requirements of LCPN training phase. Then, the concepts of data and model parallelism are described, outlining their benefits and implications. Last, we present the BD infrastructure supporting our framework.

Big Data-enabled LCPN Design

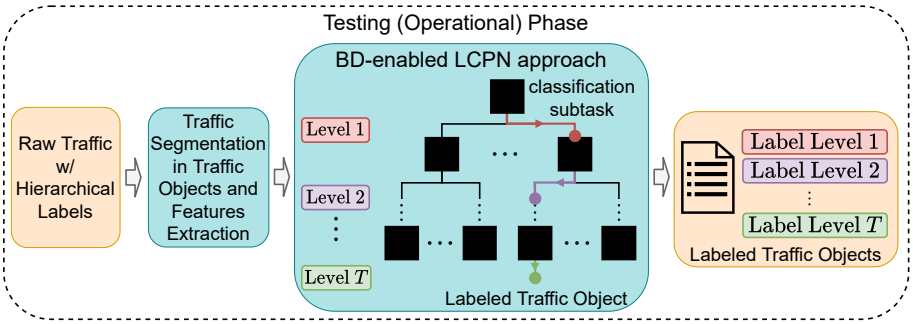


Figure 2.2: BD-enabled LCPN: classification workflow process (testing phase).

Figure 2.2 highlights how the designed LCPN classifier is integrated in our classification workflow that receives raw traffic as input to produce labeled traffic objects. In detail, it is worth to recall that the operations of the framework require an input of raw network traffic that can be effectively modeled in a *hierarchical fashion*. In other words, the action of associating network traffic to a label must be doable at *different degrees of granularity*. This is a

generic requirement that matches many and diverse real-life traffic analysis or management scenarios [45, 51], and can be easily tailored to specific needs.

Detailing the workflow, each traffic object is provided as input to the Hierarchical Learning Engine. Its output is a set of $\{\hat{\ell}_1, \ldots, \hat{\ell}_T\}$ predicted labels, each corresponding to a given classification *granularity level* (cf. Fig. 2.2). In detail, t^{th} level corresponds to L_t classes to discriminate from, with L_t *growing at finer granularities* (i.e. $L_{t+1} > L_t$).

It is worth to recall that in order to assign the T labels to each TO, a *tree dependence* for classes belonging to different levels should be imposed. Specifically, each class at $(t+1)$-th level has *at most* one parent class, which belongs to t-th classification level. In other words, the resulting model is made of multiple classifier nodes (whose number is denoted with N_c) arranged as a tree, which are traversed in a *top-down* fashion, imposing the design of a multi-class classifier for each parent node in the class hierarchy (viz. LCPN model), as described in Sec. 2.3.1. As mentioned in the above section, for each TO to be classified the LCPN first predicts $\hat{\ell}_1$, corresponding to the most generic class. The above label is then used to select the classifier node in charge of providing the label $\hat{\ell}_2$. This procedure is repeated until the T^{th} classification level. It is worth to underline that allowed classes for $\hat{\ell}_2$ are only the children of $\hat{\ell}_1$, thus narrowing the choice of classes to be predicted at second level. Making the different nodes in the tree aware only of a subset of the entire classification space is the natural outcome of a *divide-et-impera* approach. This results in a simplification of the classification problem, reducing the number of classes $(\bar{L}_1, \ldots, \bar{L}_{N_c})$ among which each node has to discriminate. Indeed, each node is trained to distinguish *only* among its child nodes.

Although errors at a given class level could propagate downwards the hierarchy, the hierarchical classification choice promotes *architecture modularity*, and enables *model parallelism*, specifically suitable for BD architectures (later shown in Sec. 2.3.2). Further, the LCPN approach enables a fine-grained (per-node) optimization of the feature set, the classifier, the hyperparameters, and even the TO. As a result, hierarchical classification is likely to achieve a significant classification performance gain against a *flat* counterpart, i.e. a single classifier solving the finest ($t = T$) classification task.

Training Requirements of the BD-enabled LCPN.

To operate in the test phase, the LCPN classifier needs to be previously initialized by a *training phase* (Fig. 2.3), that trains all the N_c ML classifier nodes of the hierarchy. For this phase, the *input* is a collection of N_c training sets

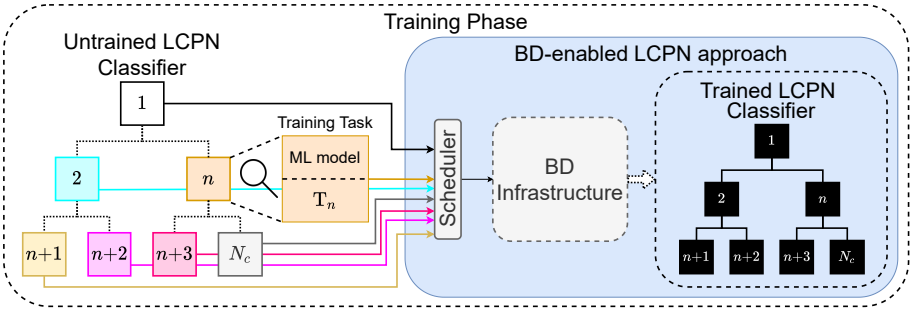


Figure 2.3: BD-enabled LCPN Classifier: training phase.

$\{\mathrm{T}_1, \ldots, \mathrm{T}_{N_c}\}$, *all* obtained starting from total training set T and retaining *only a subset* of its samples. Specifically, T_n contains only the (training) samples associated to the $\bar{L}_n$ labels constituting the TC task to be solved by n-th node. Accordingly, this implies $|\mathrm{T}_n| \ll |\mathrm{T}|$ due to the reduced number of classes ($|\cdot|$ denotes the training set size), except for the root node ($n = 1$). Indeed, in general $\bigcup_{n \in \mathcal{N}_t} \mathrm{T}_n = \mathrm{T}$ and $\bigcap_{n \in \mathcal{N}_t} \mathrm{T}_n = \emptyset$, where $\mathcal{N}_t$ denotes the set of classifiers concurring to t-th classification granularity level.

The *output* of the BD-enabled training is the set of trained classifier nodes composing the tree hierarchy. It is worth to remark that the training process of each node is *independent* on the others. Accordingly, although this phase may be implemented by a single entity training all N_c nodes in a sequential fashion, the proposed framework can leverage BD-infrastructures to exploit parallelism of both the LCPN classifier (model) and data as the following explained. This allows obtaining increased time efficiency and, with suitable design, also cost-effectiveness.

Data and Model Parallelism

The BD paradigm enables both batch and streaming distributed analysis, addressing the issues related to high data variability, volume, and velocity. Usually, they are used to obtain a time performance speedup and also contribute to shorten the training phase in ML applications. This may be achieved by exploiting both notions of *data* and *model* parallelism.

In a nutshell, *data parallelism* is based on the split of a training set associated to a ML algorithm with nonoverlapping subsets, each one assigned to a different worker. The latter performs learning (a) from its portion of data and (b) synchronizing with other workers through partial information exchange.

Such process is typically accomplished thanks to a coordinating entity (named master), which is also in charge of collecting the results of the training process based on data parallelism [56]. It is worth noting that the fragmentation of the training set could degrade the classification performance. Furthermore, a higher number of workers may also negatively impact the temporal gain, due to the burden imposed by the synchronization overhead at controller side [56].

Differently, *model parallelism* resorts to splitting the model associated to a ML algorithm (with no partitioning of the training set), with the aim of simplification of the learning task for each worker. In this peculiar case (viz. LCPN approach), model parallelism can leverage dependencies among traffic classes (i.e. hierarchical dependency) to *perfectly* parallelize the learning task over different workers, each one assigned to a subproblem. Indeed, the breakdown of the classification procedure along a tree-like architecture enables the training phase of *distinct* ML (sub-)models in parallel (e.g., the classifier nodes). Accordingly, there is no classification performance loss due to model partitioning (by construction) when considering LCPN. This is one of the peculiarities of hierarchical classification approaches like LCPN as opposed to flat counterparts, where model partitioning is not performed based on hierarchical class representation. Nevertheless, from a time-related perspective, the advantages against a flat approach are not guaranteed and need to be investigated. In other words, the advantages arising from the training of multiple but less complex classifiers, instead of the training of a single but more complex classifier, are not obvious.

Accordingly, the key idea is to *combine* the benefits of data parallelism enabled by BD paradigms with those deriving from the model parallelism granted by the adoption of the hierarchical classification architecture. In classification problems, since the duration of the learning phase for classification tasks is directly proportional to both the number of samples used for training and the number of classes to choose from, *data parallelism* can be employed to reduce the number of samples, whereas *model parallelism* to reduce the number of classes.

Description of BD Infrastructure

BD-enabled LCPN is supported by a BD infrastructure shown in Fig. 2.4 and detailed as follows. The BD infrastructure manages the computing resources in B units (*buckets*) that are instantiated on the BD framework (e.g., Apache Spark) and are handled by a *scheduler*. Workload assignment is based on a master-slave architecture, with a controller and several workers, and leverages

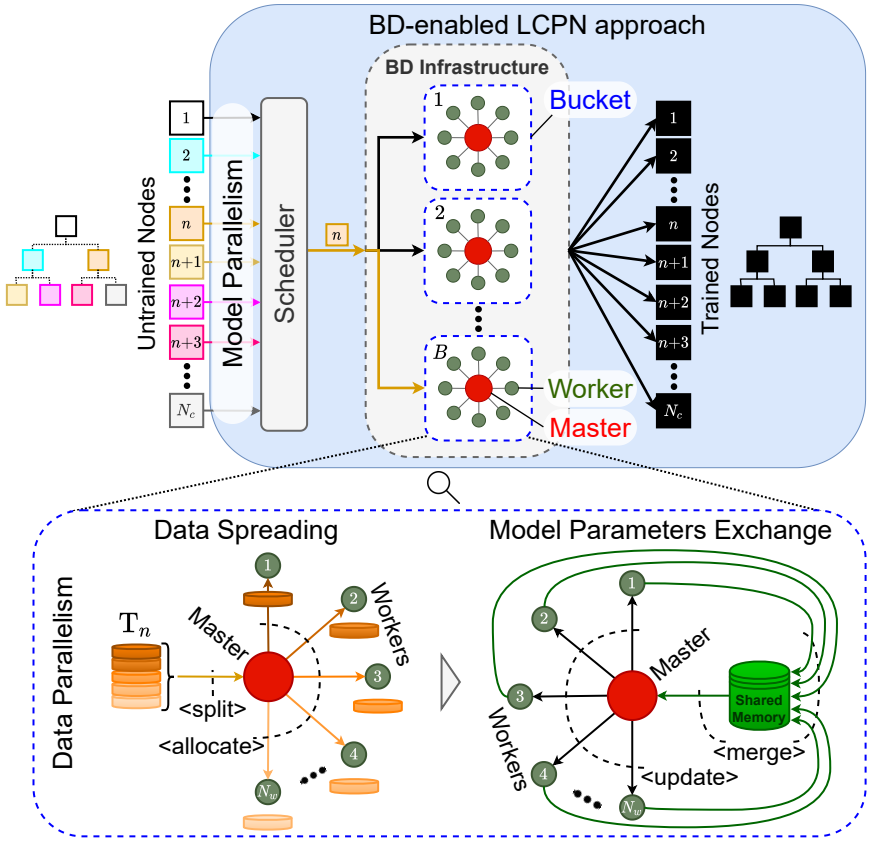


Figure 2.4: LCPN classifier training with focus on the BD infrastructure: scheduling of training tasks over buckets (top, model parallelism) and internal structure of a bucket (bottom, data parallelism).

a distributed file-system (e.g., Spark SQL, Hadoop Distributed File-System, etc.). Hence, within each bucket, there is one *master* node and a pool of worker nodes, with each master assigning and coordinating the work of the pool. Buckets are assumed to have the same number of workers (N_w), as the scheduler acts as a load balancer that distributes tasks uniformly across the buckets.

Regarding the Hierarchical Learning Engine, when LCPN is selected as hierarchical classification approach, the scheduler implements *model parallelism* by scheduling the training task of each node of the hierarchical classification architecture to one of the buckets and forwarding the corresponding

training set, along with ML model specifications. It is worth underlining that a preliminary phase is enforced by the Hierarchical Learning Engine to build the training sets $\{\mathrm{T}_1, \ldots, \mathrm{T}_{N_c}\}$ associated to the nodes of hierarchy (starting from T as a result of labelwise splitting operations). Hence, training tasks assigned to the same bucket are executed *sequentially*, whereas training tasks assigned to different buckets are run *in parallel*. Once a ML training task is assigned to a given bucket, our framework performs the latter job by implementing *data parallelism* through master-workers exchanges. In other terms, data parallelism is implemented within each bucket.

In detail, *data parallelism* for training each ML model (associated to a classifier node) is enforced by a BD framework (e.g., Spark) through two main phases, i.e. *data spreading* and *model parameters exchange* (Fig. 2.4, bottom). First, during *data spreading* phase, the training set (already a model-specific subset of the full training set) is further split into several (non-overlapping) portions (i.e. <*split*>). Accordingly, the master node acquires the required resources from the worker nodes and assigns each set portion to a given worker (i.e. <*allocate*>). Then, during *model parameters exchange* phase, status information is repeatedly exchanged between workers and the master to (i) enable the aggregation of model parameters from all worker instances (<*merge*>, workers→master) and (ii) synchronize the workers with the updated master status (<*update*>, master→workers).

Ideally, scheduling aims at minimizing the training completion time t^{tot} of the hierarchical classification architecture (i.e. the makespan). Since the latter requires all nodes to be trained in order to be put in the test phase, the above time corresponds to:

$$t^{\mathrm{tot}} \triangleq \max_{b=1,\ldots,B} t^b \tag{2.1}$$

i.e., the longest completion time among buckets $\{t^1, \ldots, t^B\}$, where B is the number of buckets. In detail, the completion time of b-th bucket can be written as $t^b = \sum_{n=1}^{N_c} \psi_{b,n} t_n$, where t_n is the training time required for n-th classifier node and $\psi_{b,n} \in \{0, 1\}$ is an indicator variable being one (resp. zero) when the training task of n-th node is assigned (resp. not assigned) to b-th bucket. For simplicity, the completion time of each node t_n is supposed constant over bucket assignment. Still, the BD-enabled LCPN training could be generalized to heterogeneous buckets, i.e. having different resource budgets.

Accordingly, the optimal scheduler provides the solution to the following

optimization:

$$\widehat{\mathbf{\Psi}} \triangleq \arg\min_{\mathbf{\Psi}} \max_{b=1,\dots,B} \left\{ t^b(\mathbf{\Psi}) = \sum_{n=1}^{N_c} \psi_{b,n} t_n \right\} \tag{2.2}$$

where $\mathbf{\Psi} \in \{0,1\}^{B \times N_c}$, whose (b,n)-th entry equals $\psi_{b,n}$. Hence, the optimization is carried out over the space of selection matrices $\mathbf{\Psi}$ (column sum is constrained to one, i.e. one task is assigned only to one bucket). It is worth noticing that the solution to the optimization in Eq. (2.2) is infeasible since in real scenarios the time required for each training task (namely $t_1, \dots, t_{N_c}$) is *unknown a priori* and due to *NP-completeness* of the optimization problem. The explaination of how these two technical issues are circumvented follows.

First, in the place of task completion time, a surrogate function $\rho(\cdot,\cdot)$ associated to the completion time has been defined. The need for defining a surrogate metric originates from no known general and explicit expressions of complexity of ML classifiers as a function of relevant parameters considered: (i) size of the training set and (ii) number of classes. In detail, for the n-th classifier waiting time t_n, such function depends on the number of samples of training set ($|\mathrm{T}_n|$) and classes of the node's TC task ($\bar{L}_n$), namely

$$\rho_n \triangleq \rho(|\mathrm{T}_n|,\, \bar{L}_n) \triangleq |\mathrm{T}_n|\, \bar{L}_n \;/\; (\sum_{m=1}^{N_c} |\mathrm{T}_m|\, \bar{L}_m) \tag{2.3}$$

Hence, t_n has been replaced with ρ_n to perform the optimization in Eq. (2.2). It is worth noticing that other complexity measures monotonically growing with both $|\mathrm{T}_n|$ and $\bar{L}_n$ could be considered as well.

Secondly, the used *priority scheduling* approach [1] has the advantage of being $\mathcal{O}(N_c)$. It is based on the aforementioned surrogate and aims at assigning training tasks so to balance the completion time among all buckets. In the following, all scheduling strategies have been assumed to deal with $N_c > B$ since they collapse into trivial assigments for $N_c \leq B$.

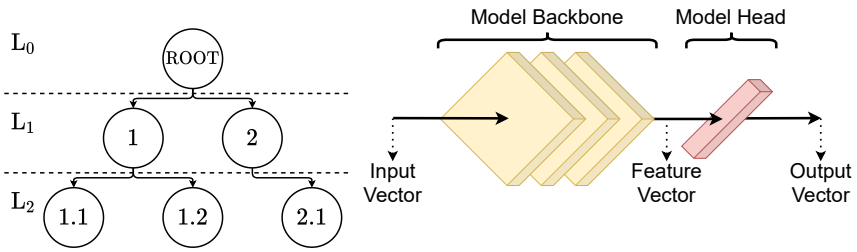
The Hierarchical Learning Engine supports *two* scheduling strategies, namely *offline* and *online scheduling*, to fulfill the LCPN model training. Both scheduling strategies sort the tasks by decreasing ρ_n first and assign one task per bucket accordingly. The former *statically* assigns each remaining task to the bucket having the lowest current sum of ρ_n's already assigned to it. The latter strategy *dynamically evaluates the state of the buckets* and, when a bucket

[1] This approach is also referred to as "longest processing time" approach in scheduling literature.

completes its currently assigned task, it is assigned the remaining task with the highest ρ_n. Differently from the offline scheduling, this strategy exploits time completion feedback at the cost of monitoring each bucket state. Indeed, feedback has been shown to "repair" the degrading effect of uncertainty on scheduling results (due to the unavailability of t_n's and their replacement with ρ_n's) and provide general beneficial effects (i.e. independently on the specific scheduling approach adopted) by monitoring workers' workload status [78].

2.3.3 Global Classifiers Design

Beyond the support for LCPN-based hierarchical learning approach, the proposed framework supports several Global Classifier (GC) (viz. big-bang) techniques for TC which are described in this section and are implemented by the Hierarchical Learning Engine.



(a) Class Dependencies (b) General DL Model

Figure 2.5: Examples for Global Classifier Techniques.

Subsequently, examples for each GC approach are carried out exploiting both the simple hierarchy of labels and the DL general model which are depicted in Fig. 2.5, by focusing on the mapping between neurons in the head(s) and classes in the hierarchy; between all the GC techniques only the naïve global classifier is applicable as-is to classical ML models. It is worth to underline that the global classifier approaches are only defined for Neural Network (NN)-based models which are composed by two parts (Fig. 2.5b): (i) the model backbone, which extract relevant features from the input vector resulting in the feature vector, and (ii) the model head, which performs classification by providing the class-wise probability distribution starting from the feature vector. At the end of each section the loss function each model optimizes has been reported.

Naïve Global Classifier

The Naïve Global Classifier (NGC) technique consists in a model that is trained on all the possible labels in the hierarchy, resulting in a model whose predictions could fall at each node (viz. label) of the hierarchy. Despite this approach is prone to predict non-leaf node, *post-processing operations* on the soft output could be enforced, e.g., by aggregating by hierarchical dependencies or by fixing the granularity (viz. level) at which one wants to predict by considering only related confidence values.

Therefore, this approach is based on the association of each training sample with a set of labels, i.e. each sample is associated with multiple labels, one for each level of the hierarchy. In this way the model is capable to acquire knowledge about hierarchical dependencies. As abovementioned, the soft outputs obtained through this procedure could be aggregated in several ways, by leveraging hierarchical dependencies. Accordingly, the Hierarchical Learning Engine implements two rules of aggregation, namely the *finest-grain* and the *hierarchical*, where the former is based on the filtering out all confidence values that are not related to the finest-grain level of the hierarchy, and the latter is based on the *addition* or *product* of the confidence values involved in the same path of the hierarchical dependencies tree. Both aggregation rules are followed by a normalization procedure in order to obtain a well-formed probability distribution vector.

The loss function this technique optimizes is:

$$L_{NGC} = -\sum_{t=1}^{T}\sum_{j=1}^{L_t}\sum_{i=1}^{\mathcal{S}_{tj}} y_i \times \log \hat{y}_i$$

It is worth to recall that T is the number of levels the hierarchy is composed by, L_t if the number of classes at level L_t, and $\mathcal{S}_{tj}$ is the number of training samples for the j^{th} node of the t^{th} level.

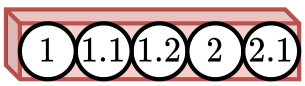


Figure 2.6: Example: NGC approach. The circles represent the possible classes from the output (head).

Combined Global Classifier

This technique is similar in essence to the NGC, but it is based on the modeling of all the possible combinations of labels across levels. This enlarged output space is thus formed by *valid* and *non-valid* combinations. Despite this technique does not predict nonleaf nodes, it could happen that the prediction is a non-valid combination: several postprocessing operations should be applied to the soft outputs of the model, e.g., by considering only valid confidence and by exploiting non-valid confidences in some ways.

In contrast to the NGC technique, with the Combined Global Classifier (CGC) each training sample is associated to a valid combination of labels, one per level, but the model's output space considers both valid and non-valid combinations: in this way each sample acts as a negative example for non-valid classes, potentially pushing the generalization and accuracy of the trained model. Indeed, when the underlying model is NN-based, the target one-hot vector positions related to non-valid combinations are always zero. Also for this technique, the Hierarchical Learning Engine applies an aggregation rule, named *only valid*, which considers only valid confidence values from the soft output. Also in this case the rule requires a normalization in post-processing.

The loss function this technique optimizes is:

$$L_{CGC} = -\sum_{j=1}^{L_T}\sum_{i=1}^{\mathcal{S}_{Tj}} y_i \times \log \hat{y}_i$$

where L_T is the number of leaf nodes, and $\mathcal{S}_{Tj}$ is the number for training samples for the j^{th} leaf (viz. level T) node.

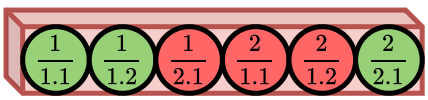


Figure 2.7: Example: CGC approach. The circles represent the possible classes from the output (head). Circles in red represent not valid combinations of labels, in green the valid ones.

Task-Incremental Global Classifier

Task-incremental learning is a paradigm of model training where, starting from a model trained on a specific task, an incremental training procedure is enforced in order to extend the capability of such a model to cope with other

tasks. This kind of training paradigm could fit the global classifier objective, i.e. training a single model that exploits hierarchical dependencies among labels, by considering each level of the hierarchy as an incrementally-learned task [79, 80]. In this thesis, we refer to a task-incremental trained GC model as Task-Incremental Global Classifier (TIGC).

Practically, most task-incremental learning solutions are based on the assumption that each (neural network) model is composed by two in-tandem components, namely the *backbone* and the *head*, where the backbone acts as a features extractor and the head performs classification (as shown in Fig. 2.5b). Given a couple of backbone–head trained on a particular task, task-incremental learning consists in adding a new head to the backbone, which head will be specialized for a new task, by leveraging the already trained backbone in some ways (e.g., by fine-tuning the couple backbone–new head on the new task). Therefore, for hierarchical traffic classification, each task corresponds to a level of the hierarchy, and we apply task-incremental learning by adding a new head, starting from the coarsest-granularity level/task to the finest, and by sequentially (fine-)tuning the model backbone on each task/level.

Finally, about the model's output management, the model obtained via task-incremental training is composed by three separated heads, each one predicting a level of the hierarchy, potentially falling in prediction inconsistency (viz. the labels predicted by each head are not hierarchically dependents). In this thesis we bypass this issue by considering only the *predictions of the finest-granularity head.*

The loss function for the t^{th} task (viz. level) that this technique optimizes is:

$$L^t_{TIGC} = -\sum_{j=1}^{L_t}\sum_{i=1}^{\mathcal{S}_{tj}} y_i \times \log \hat{y}_i$$

It is worth to recall that L_t is the number of classes at the L_t level (viz. task) of the hierarchy, and $\mathcal{S}_{tj}$ is the number for samples for the j^{th} node of the t^{th} task/level.

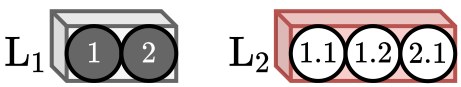


Figure 2.8: Example: TIGC approach. The circles represent the possible classes from the output (head). The gray head and circles are related to the old task (viz. predicting at L_1).

Multitask Global Classifier

Similarly to the TIGC technique, the Multitask Global Classifier (MGC) technique (viz. multi-task hierarchical learning) is based on a model that solves multiple tasks all together, but also training all together, avoiding the incremental fine-tuning which affects the TIGC. When the model is NN-based, the optimization (minimization) objective is a loss function composed by a term for each task/level. Despite it is possible to assign a different weight to the loss of each task/level, we adopt a uniform weighting rule. Noteworthy, the same considerations carried out for the outputs management of TIGC apply also to this case, considering only finest-grain head predictions.

The loss function this technique optimizes is:

$$L_{MGC} = -\sum_{t=1}^{T}\sum_{j=1}^{L_t}\sum_{i=1}^{\mathcal{S}_{tj}} y_i \times \log \hat{y}_i$$

where T is the number of levels (viz. tasks), L_t is the number of classes at the level L_t (viz. task) of the hierarchy, and $\mathcal{S}_{tj}$ is the number of samples for the j^{th} node of the t^{th} level.

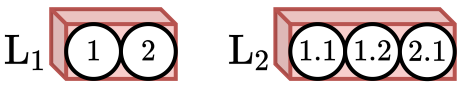


Figure 2.9: Example: MGC approach. The circles represent the possible classes from the output (head) of each task.

Hierarchical Loss Global Classifier

Among GC techniques, the Hierarchical Loss Global Classifier (HLGC) enforces the hierarchy without extending the model, as the previously presented GC approaches, but embedding the hierarchical dependencies in the loss function by defining a *hierarchical loss* [81]. The hierarchical loss is enforced by the mean of an Ultrametric Tree, which is applied to the softmax output of the model and to the one-hot-encoding of labels (viz. target) to enforce hierarchical dependencies in the loss function.

In detail, this procedure starts from the set of ancestors A (w/o the root node), one for each leaf label, and from a set of coefficients c, one per level, and transform the probability distribution P of the leaf level predictions by enforcing hierarchical dependencies. This procedure is described in Alg. 1.

Algorithm 1 Hierarchical Representation.

Require: A, c, P
  $\mathrm{D} \leftarrow get\_dependencies\_matrix(\mathrm{A})$
  $\mathrm{H} \leftarrow [\,]$
  $\mathrm{H}.append(\mathrm{P} \cdot \mathrm{c}[0])$
  for i in $[\,0, \mathrm{D}.length()\,)$ **do**
    $\mathrm{H}.append((\mathrm{P} \times \mathrm{D}[\mathrm{i}]) \cdot \mathrm{c}[i+1])$
  $\mathrm{H} \leftarrow reduce\_sum(\mathrm{H})$
  return $softmax(\mathrm{H})$

N.B. The function *get_dependencies_matrix* transforms the vector of ancestors (A) per leaf label in $T-1$ binary squared matrices (D), one per non-leaf level. Those matrices are of dimensions $L_T \times L_T$, where L_T is the number of leaf labels. The matrix for the level L_k contains 1 as i, j element if the i and j leaf nodes are siblings at the level L_k, otherwise 0. H is the hierarchical representation of vectors in P.

In detail, the coefficient c_i for the level L_i is $1/2^i$, otherwise for the leaf-nodes of level L_T it results $\mathrm{c}_T = 1/2^{T-1}$. Because L_0 (viz. the ROOT node) is not considered, its coefficient c_0 is practically zeroed.

Contextual Label Smoothing Global Classifier

Another widely used (in non networking-related literature) technique to obtain a GC is Contextual Label Smoothing Global Classifier (CLSGC). As the HLGC, this technique does not expand models, but enforces hierarchical dependencies via Contextual Label Smoothing (CLS), that is very similar to the Ultrametric Tree. CLS extends classical label smoothing by embedding hierarchical dependencies via a smoothed representation which considers sibling/ancestor relations among the leaf nodes' labels [82], i.e. fixed a label, "older" is the kinship with another label higher is the smoothed value related to this label.

Let B the list of branches defining the hierarchical dependencies, e.g., in Fig. 2.5a $\mathrm{b}_0 = 1 \rightarrow 1.1 = (1, 1.1)$ is the branch of the leaf node 1.1, each value of B matches a leaf label of the hierarchy. CLS representation of the labels' one-hot-encoding is obtained via the steps described in Alg. 2.

For example, looking at the hierarchy depicted in Fig.2.5, the one-hot encoding of the leaf node 1.1, that is $[1, 0, 0]$, results in the $[.9, .09, .01]$ CLS representation. In detail, we first compute the $\mathrm{com\_anc}$ (*common ancestors count*): between 1.1 and itself it is 2 (i.e. $\|(1, 1.1) \cap (1, 1.1)\| = 2$), between 1.1 and 1.2 it is 1 (i.e. $\|(1, 1.1) \cap (1, 1.2)\| = 1$), and between 1.1 and 2.1 it is

Algorithm 2 CLS Representation.

Require: B, x
  for each leaf ohe label (i) in the hierarchy **do**
    $\mathrm{weights} \leftarrow [\,]$
    for each branch (j) in B **do**
      $\mathrm{com\_anc} \leftarrow \|\mathrm{b}_i \cap \mathrm{b}_j\|$
      $\mathrm{weights}.append(x^{\mathrm{com\_anc}})$
    return $\frac{\mathrm{weights}}{|\mathrm{weights}|}$

0 (i.e. $\|(1, 1.1) \cap (2, 2.1)\| = 0$). Then, $x = 10$ fixed, we compute the $\mathrm{weights}$ vector starting from the $\mathrm{com\_anc}$ list (i.e. $[10^2, 10^1, 10^0] = [100, 10, 1]$). The last step is the normalization of such vector (i.e. $\frac{[100,10,1]}{111} = [.9, .09, .01]$).

Multi-Label Global Classifier

The Multi-Label Global Classifier (MLGC) is in essence similar to the NGC approach because (i) both predict all the labels which are in the hierarchy and (ii) each training sample is associated with a set of labels, one per level of the hierarchy. Otherwise, the main difference between the two approaches is that, while for the NCG approach each sample in the training set is passed (for each training epoch) one time per level each time with the level-specific label, for the MLGC we train a model with a multi-label target vector (viz. not one-hot-encoded as for NGC, but *multi-hot-encoded*). Multi-label classification is applied when an object could be associated to multiple labels at the same time (in our case to one label per level), and it consists in applying sigmoid activations with binary cross-entropy loss, instead of the softmax with categorical cross-entropy loss used by other approaches.

The loss function this technique optimizes is

$$L_{MLGC} = -\frac{\sum_{t=1}^{T}\sum_{j=1}^{L_t}\sum_{i=1}^{S_{tj}} y_i \times \log \hat{y}_i + (1 - y_i) \times \log (1 - \hat{y}_i)}{\sum_{t=1}^{T}\sum_{j=1}^{L_t} S_{tj}}$$

where t is the number of levels, L_t is the number of label nodes (viz. classes) at level L_t, and S_{tj} is the number for samples for the j^{th} node of the t^{th} level.

2.3.4 Classification and Detection Models

In this section, we report several Statistical and Neural Network-based models which were successfully applied for Traffic Analysis-related tasks. In particular, models are taxonomized in base of the nature of the prediction task they perform in Multi-Class Classifiers (MCCs) and One-Class Classifiers (OCCs). From a traffic analysis point of view, such classes adhere to one or more tasks. Namely, MCCs models could deal with Traffic Classification, Misuse Detection, or Attack Classification problems, otherwise OCCs models majorly deal with Anomaly Detection (viz. novelty or outlier detection). However, this associations should not be taken as absolute because several adaptations could be applied, e.g., the application of a reject option could enable MCCs models in discovering outliers.

Multi-Class Classifiers

Statistical-based Family:

- **Decision Tree (DT)** is a tree-based model used for classification based on the distribution entropy concept and trained via a top-down recursive and greedy procedure. C4.5 [83] is the most famous algorithm employed to generate a DT, which enables the previous formalization in dealing with categorical features.

- **Random Forest (RF)** [84] is a classifier based on an ensemble of different decision trees (viz. a forest of DT) built at training time exploiting the idea of *bagging* (viz. "bootstrap aggregating" and random feature selection) to features and samples. This model is practically introduced to overcome the easy overfitting the DT is susceptible to and the optimization is obtained by training multiple sub-performing DTs and by taking the majorly voted class as the final prediction.

- **eXtreme Gradient Boosting (XGB)** [85] is a tree-based model that improves the RF by applying *boosting* (plus regularization) in the training phase, which is referred as *additive training*: basically, the output of each trained DT is posed at the input of the subsequent DT in the ensemble.

- **Naïve Bayes (NB)** is a simple probabilistic classifier that assumes class conditional independence of the features, being not the case for real-world problems, but working well in practice and leading to low com-

plexity. The variant shown to perform better in [5] is the one uses supervised discretization for numerical features.

- **Bayesian Network (BN)**, mitigating the stringent assumption underlying naïve bayes, is a classifier which models the dependence relationships between features and classes, usually via a *direct acyclic graph*. The best performing variant from [5] is obtained when the dependence model is reduced to a simpler tree.

Neural Network-based Family:

- **MultiLayer Perceptron (MLP)** is the feedforward neural network of the beginnings, consisting of an output layer and at least one hidden layer, being able to learn non-linear mapping between inputs and outputs.

- **1-Dimensional Convolutional Neural Network (1DCNN)** is a convolutional-based DL model proposed by [22] for TC. As abovementioned, the authors exploited the raw payload (referred as PAY in the follow) of network packets in order to perform TC. Accordingly, the 1DCNN model takes as input a matrix of dimension $(n\_bytes, 1)$. The composition of this model is shown in Fig. 2.10a, with the model backbone constituted by a double stage of 1-D Convolutional plus Max Pooling layers, and a Fully Connected (viz. Dense) at the end; the model head is a fully connected layer.

- **Long Short-Term Memory (LSTM)** is a recurrent-based DL model proposed by [24] for TC. In this case, the authors exploited the time-series features from the IP and L4 headers (referred as HDR in the follow) of network packets in order to perform TC. Therefore, this model takes as input a matrix of dimension $(n\_packets, n\_features)$. The composition of this model is shown in Fig. 2.10b, with the model backbone constituted by a cascade of LSTM and Dense layers; also for this model the head is a Dense layer.

- **2-Dimensional Convolutional Neural Network (2DCNN)** is a convolutional-based DL model proposed by [24] for TC. As for the LSTM, the authors exploited the HDR features set, resulting in an input sized $(n\_packets, n\_features)$. The composition of this model is shown in Fig. 2.10c, with the model backbone and head that practically follow the structure of the 1DCNN, but using 2-D Convolutional layers.

- **2DCNN and LSTM cascade (2DCNN+LSTM)** is a hybridization of abovementioned 2DCNN and LSTM models proposed in [24], exploiting in the model backbone the CNN as feature extractor and an LSTM on top of these (Fig. 2.10d). The input is the same as the 2DCNN and LSTM models, as also the model head.

- **MultImodal DL-based MobilE TraffIc Classification (MIMETIC)** is an ensemble of DL models, enabling for multi-modality modeling, proposed in [16]. In detail, this model is composed of two branches (modalities): (i) a lighter version of the abovementioned 1DCNN, which takes the PAY input, and (ii) an enhanced version of the abovementioned LSTM exploiting a Bidirectional Gated Recurrent Unit (BiGRU) instead of the LSTM layer, which is fed with the HDR features set. The head of this model is composed by an information fusion component, required to merge the per-modality extracted features, and ends with the usual Dense layer.

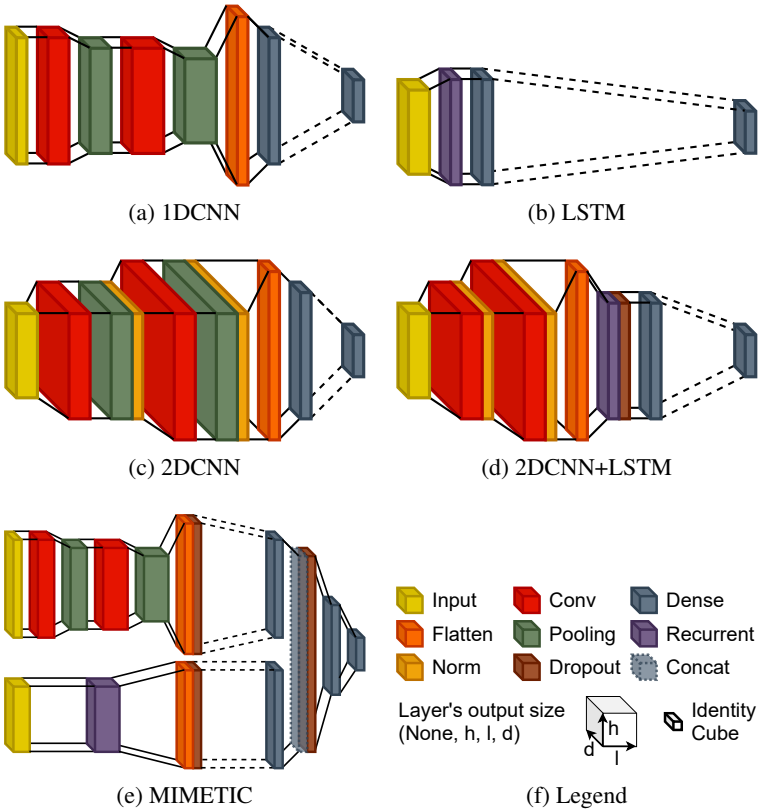
One-Class Classifiers

Statistical-based Family:

- **One-class Support Vector Machine (OC-SVM)** [86] is the natural extension of SVM algorithm to the case of unlabeled data. This problem has been approached by attempting to estimate a function f that is positive for regions with high density of points (assumed as normal), and negative for small densities (assumed as anomalous). The functional form of f is given by a *kernel expansion in terms* of a subset of the training data.

- **Isolation Forest (IF)** [87], unlike other model-based anomaly detection methods, does not create a profile for normal cases in order to discover anomalies, but instead explicitly isolates anomalies. Exploiting the "few and different" nature of anomalies, IF separates anomalies closer to the root of the tree than normal points. Because of this unique feature, IF can develop partial models (as opposed to full models in profiling) and use only a small percentage of training data to create effective models.

- **Local Outlier Factor (LOF)** [88] is based on the computation of a degree of being an outlier to each sample, instead of considering being an



(a) 1DCNN (b) LSTM

(c) 2DCNN (d) 2DCNN+LSTM

(e) MIMETIC (f) Legend

Figure 2.10: Adopted DL models. The *dashed lines* connect each model's *backbone* to the respective *head*. The legend (f) shows the color of each layer and the key of lecture of boxes dimensions (w.r.t. the output size of the respective represented layer). It is worth to underline that all the Recurrent layers are LSTM, but the one in (e), which is a bidirectional GRU.

outlier as a binary property. This degree is called the *local outlier factor* of a sample and it is local because this factor depends on how isolated the sample is with respect to its neighborhood.

Neural Network-based Family:

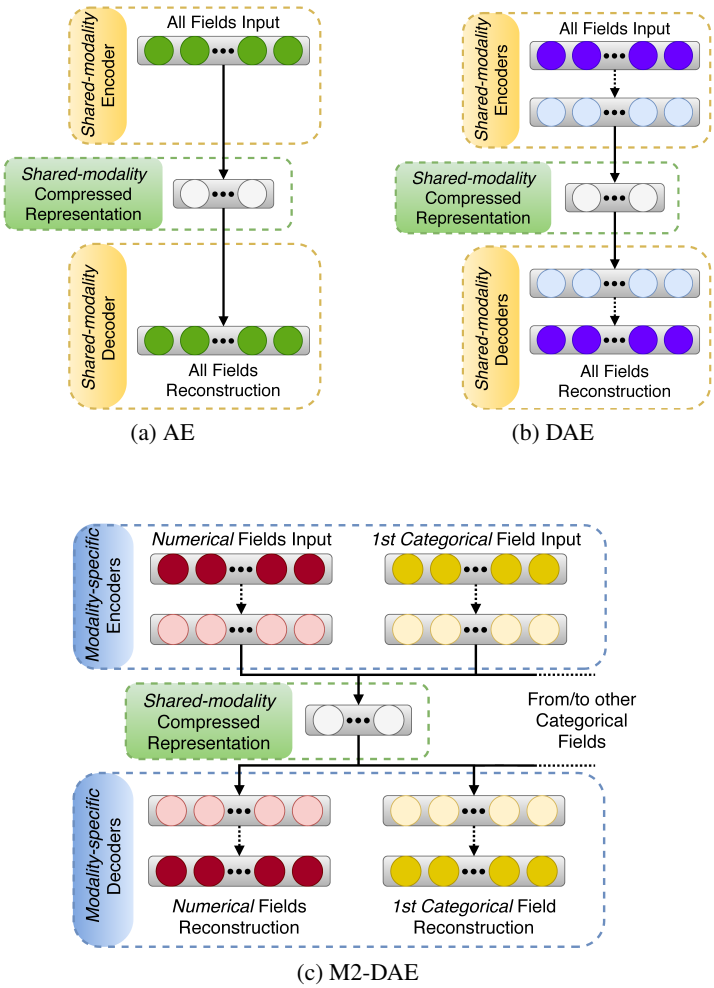
- **AutoEncoder (AE)** and **Deep AutoEncoder (DAE)** working principle is depicted in Figs. 2.11a and 2.11b. AEs are commonly employed as an unsupervised feature extractors, and their aim is to set the output

equal to the input $\hat{\boldsymbol{x}}(m) \approx \boldsymbol{x}(m), \forall m$ within the training set, by learning a compressed data representation of benign traffic which minimizes the loss $\geq(\hat{\boldsymbol{x}}(m), \boldsymbol{x}(m))$ (e.g. the *mean squared error*). In detail, the first AE layer (i.e. the encoder) provides a lower-dimensional data representation, whereas the second layer (i.e. the decoder) tries to reconstruct the data from the compressed representation. Differently from the AE, the DAE has several encoding and decoding layers. This increases the generalization capability of such a model, thus increasing the model complexity. Noteworthy, AE-based models could be used as anomaly detectors, i.e. by using (during testing phase) the loss metric between the observed input and the AE-based reconstruction as a measure of "anomaly-ness" [89].

- **Multi Modal-Deep AutoEncoder (M2-DAE)** is an evolution of DAE which handles different input types of the TO as separate "modalities", thus reducing the number of trainable parameters of the network. To accomplish this task, the M2-DAE network is made of both modality-specific and shared-modality layers (both encoding and decoding). The proposed M2-DAE has the appeal of learning a compressed representation keeping a more-efficient (viz. less parameters) neural network structure than DAE, which only uses shared (encoding/decoding) layers and processes different input types in a flattened form. In detail, a single modality deals with all the *numerical* input fields and one modality for each *categorical* field is applied. In the latter case, *TO-based* categorical fields (one instance per TO, e.g., port) are represented via the well-known one-hot-encoding format, whereas *packet-based* categorical fields (one instance per packet within the TO, e.g., TCP flags) are arranged herein in a multinomial-encoded format, e.g., the vector average of one-hot-encoding over the packets of the TO. The reconstruction loss of M2-DAE is a weighted sum of the *per-modality* losses and it is used as the "anomalyness" score to catch anomalies.

2.3.5 Performance Metrics

In this section, we report the most common performance metrics usually used to evaluate classification systems. In detail, the introduced metrics can be divided into two families, namely "classical" and "hierarchical", with the former covering the metrics with mandatory leaf prediction (viz. only fine-grained



(a) AE (b) DAE

(c) M2-DAE

Figure 2.11: Comparison among AE (a), DAE (b), and M2-DAE (c) architectures.

predictions are considered), and the last group, which is composed by hierarchical metrics, namely with non-mandatory leaf prediction (viz. the classifier could also predict labels at higher levels in the dependencies hierarchy). The hierarchical metrics are taken from the artificial intelligence literature that tackled hierarchical classification. Moreover, we designed new hierarchical metrics to perform the per-node evaluation of the LCPN, to detect bottleneck

performance, and the hierarchical reliability evaluation of models via hierarchical calibration analysis. Furthermore, classification systems should be evaluated from *four* complementary (and intertwined) performance aspects, which are essential for a complete evaluation of the proposed Hierarchical Framework: (i) classification performance; (ii) training completion time; (iii) cloud deployment cost; (iv) calibration analysis; It is worth underlying that (ii) and (iii) are introduced in order to properly evaluate the BD-enabled training of the LCPN classifier. Noteworthy, the BD-enabled training could also impact the classification performance.

According to the considerations mentioned above, this section is divided into three parts, namely *classification performance metrics*, which introduces both classical and hierarchical (binary-) multi-classification performance metrics, *BD-related metrics*, in which we explain issues could affect classification performance metrics, and introduce training completion time and cloud deployment cost aspects, and finally *model calibration metrics*, which regards the reliability of trained models and presents our proposal for hierarchical reliability metrics.

Classification Performance Metrics

The performance evaluation process is usually based on the following main performance metrics: accuracy, precision (prec), recall (rec), and specificity (spec). For conciseness, out of the latter three metrics, one can consider the *F-measure*, $\mathrm{F} \triangleq (2 \cdot \mathrm{prec} \cdot \mathrm{rec})/(\mathrm{prec} + \mathrm{rec})$, and the *G-mean*, $\mathrm{G} \triangleq \sqrt{\mathrm{rec} \cdot \mathrm{spec}}$. Since these two arise from three metrics defined on a per-class basis, usually their arithmetically averaged (viz. *macro*) versions are adopted. Also, *confusion matrices* (breaking the results down by class) can be leveraged to provide a representation of the results of the investigated classification approaches, also highlighting misclassification patterns at fine grain.

Focusing on hierarchical metrics, these are defined starting from the Precision and the Recall scores (Eq. 2.4).

$$P = \frac{TP}{TP + FP},\ R = \frac{TP}{TP + FN} \tag{2.4}$$

In detail, TP, FP, and FN are the number of *true positive*, *false positive*, and *false negative*, respectively. Accordingly, the hierarchical version of such metrics defined in [90] is formalized in Eq. 2.5.

$$hP = \frac{|\hat{Y}^b \cap Y^b|}{|\hat{Y}^b|},\ hR = \frac{|\hat{Y}^b \cap Y^b|}{|Y^b|} \tag{2.5}$$

with Y^b ($\hat{Y}^b$) being the actual (predicted) branch for a sample, where for *branch* is intended the list of nodes in the hierarchical dependency tree starting from the root to the actual (predicted) leaf, but discarding the root. Also these metrics can be summarized via their harmonic mean, i.e. the hierarchical F1 Score. Noteworthy, to detect *performance bottlenecks* of the LCPN approach, we also provide *per-node metrics* (i.e. not considering classification errors introduced by upper levels), deriving useful guidelines for system design and evaluation.

Moreover, in the context of intrusion detection, other performance measures could be adopted [10]. In particular, for Anomaly Detection (AD) the comparison among models is presented in terms of the *True Positive Rate* (TPR, i.e. the ratio of correctly detected anomalies) vs. the *False Positive Rate* (FPR, i.e. the ratio of benign samples incorrectly declared as anomalies), referred to as *Receiver Operating Characteristic* (ROC). To evaluate Attack Classification (AC) capabilities, again the the *macro F-measure* (*F1*, i.e. the harmonic mean of precision and recall) is considered along with confusion matrices to highlight fine-grained error patterns.

Finally, considering that in our design each classifier implements a *reject option*, we also deepen the impact of this design choice on performance. In more detail, we investigate the impact of varying the thresholds, whose tuning can be effective to improve the classification performance, trading it off with the reduction of the classified instances (viz. the ratio of classified instances, Classifier Ratio (CR)).

Big Data-related Metrics

Since BD technologies do not constitute a transparent accelerator for the training phase of ML-based traffic classifiers (can impact the resulting performance), we evaluate *classification performance* of our framework leveraging the well-known *F-measure* to assess classification effectiveness. As abovementioned, this metric represents the harmonic mean of per-class precision and recall, arithmetically averaged over all considered classes (viz. macro averaged).

Moreover, because reducing the processing time required for task completion is arguably the major driver in the adoption of BD architectures, we

provide a detailed evaluation of this key aspect. In detail, we focus on *training completion time* t^{tot}, because classification systems are expected to require frequent re-training operations, due to the aging of training data as a result of the quick evolution of network applications and their usage.

Finally, beyond the performance in terms of classification effectiveness and execution time, a key aspect when deploying applications on cloud is the *cost* of the analysis according to the pay-per-use model. This cost is proportional to the duration of the analysis (training phase in our case) and the configuration of the BD architecture (i.e. the degree of parallelism). In turn, the configuration cost is proportional to the number of master machines and the number of worker machines. Hence, the total cost c^{tot} scales according to:

$$c^{\text{tot}} \triangleq B\,(c^M + N_w\,c^W)\,t^{\text{tot}} \tag{2.6}$$

where c^M (resp. c^W) denotes the master (resp. worker) hourly cost. Noteworthy, AWS, like most public cloud providers, provides a per-second billing with a $60\,s$ minimum, but since all the training completion times observed in the use cases exceed $60\,s$, such constraint does not affect our cost evaluation.

Model Calibration Metrics

Moreover, we evaluate via reliability diagrams the Expected Calibration Error (ECE) and the Maximum Calibration Error (MCE) [91], which compare neural network output probabilities to the expected model accuracy in order to determine the calibration degree In general, a better calibrated model is less prone to produce overconfident misclassifications, thus promoting the application of threshold based censoring mechanisms (viz. reject option) with the result of improving classification performance by limiting the cost of non-classified (underconfident) samples.

In particular, to evaluate the expected model accuracy, the predictions are grouped into M equally sized bins with respect to the output probabilities. Let be B_m the set of samples whose confidence fall within $(\frac{m-1}{M}, \frac{m}{M}]$, the expected accuracy and the average confidence are defined in Eqs. 2.7 and 2.8, respectively.

$$A(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} 1(\hat{y}_i = y_i) \tag{2.7}$$

$$C(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \hat{p}_i \tag{2.8}$$

Accordingly, the ECE and MCE are defined in Eqs. 2.9 and 2.10, where n is the number of samples.

$$\mathrm{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} |A(B_m) - C(B_m)| \tag{2.9}$$

$$\mathrm{MCE} = \underset{m \in \{1,\ldots,M\}}{max} |A(B_m) - C(B_m)| \tag{2.10}$$

To a similar extent to [90], we formulate the hierarchical version of ECE and MCE, namely, hECE and hMCE, which follow the definition of classical ECE and MCE (Eqs. 2.9 and 2.10) but leveraging hA (described in Eq. 2.11) instead of A.

$$hA(B_m) = \frac{|\hat{Y}_m^b \cap Y_m^b|}{|B_m|} \tag{2.11}$$

2.4 Implementation Details

In this section, we first report implementation details related to the tools and technologies we leveraged for the scripting of the proposed framework. Then, the package view of the Hierarchical Learning Engine, with respect to the relation among principal packages, is explored.

2.4.1 Leveraged Tools & Technologies

The proposed Hierarchical Learning Framework for Network Traffic Analysis has been prototyped by majorly leveraging the Python programming language. Each component, to accomplish its task, shares with others a set of common Python libraries and leverages component-specific Python libraries.

In detail, each component takes advantage of `numpy` and `pandas` modules, which are used to manipulate numerical data and to properly manage both the input/output data to/by the Hierarchical Learning Engine. Moreover, both the Traffic Segmentation and the Features Extraction components leverages `scapy` (and optionally `pyshark`) in order to manipulate network packets. Regarding the sole Features Extraction Component, this exploits also the `scipy` module to compute several statistics about traffic objects. Finally, the Hierarchical Learning Engine exploits `keras` (with functional APIs), `scikit-learn`, and `python-keras-wrapper` libraries for Python to instantiate models, and we selected Apache Spark (with `PySpark` module) as

the most suitable BD technology, in terms of performance and ease of deployment.

2.4.2 Hierarchical Learning Engine Package View

In this section we report the package view of the prototyped Hierarchical Learning Engine in Fig. 2.12. It is worth to underline that both the Traffic Segmentation and the Features Extraction components presented above are not shown because strictly dependent on the way the used labeling information are provided, e.g., in tabular format such as CSV files, in form of raw PCAPs with labeling information apart, and so on.

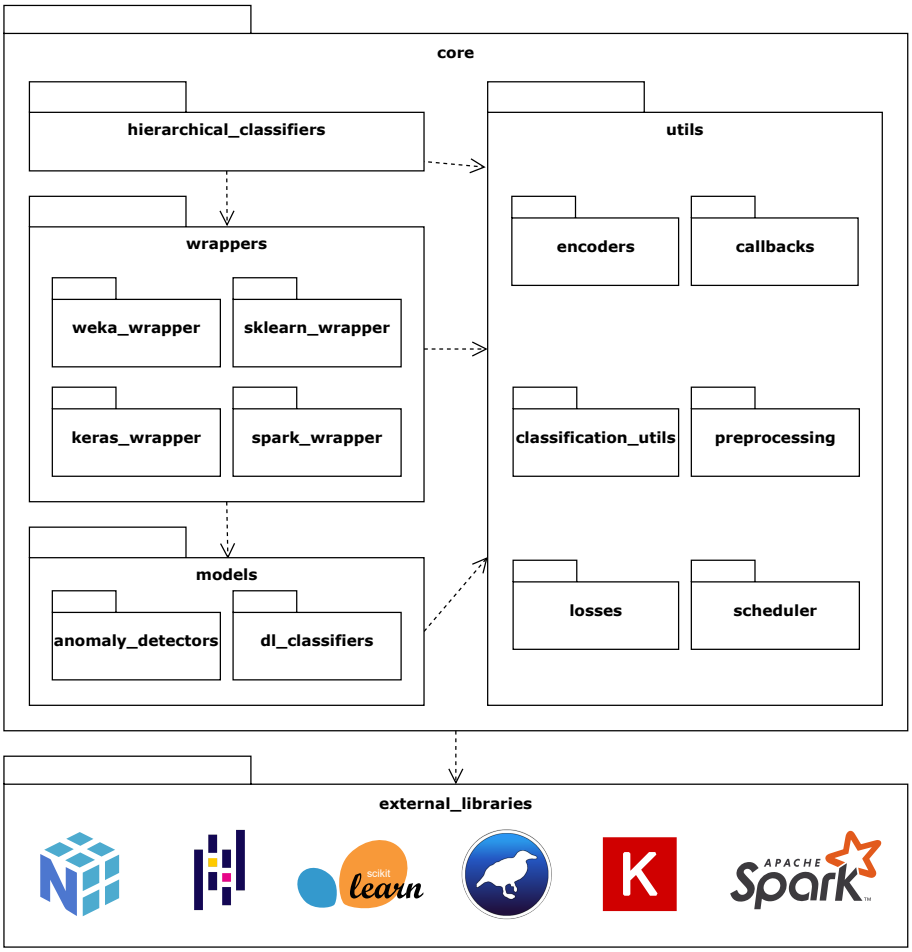


Figure 2.12: Package view for the prototyped Hierarchical Learning Engine.

About Fig. 2.12, the Hierarchical Learning Engine is composed of four main packages in the *core*, namely: the *hierarchical_classifiers* package, that implements the designed LCPN approach and deals with the input dataset in order to properly feed underlying classification models; the *wrappers* package, that contains subpackages each one dealing with a specific external library which implements ML- and DL-based models and implementing a common interface in order to easy the interaction with other packages; the *models* package, that contains subpackages which defines DL-based one-class-classifiers (viz. *anomaly_detectors*) and binary/multi-class-classifiers (viz. *dl_classifiers*); and the *utils* package, that contains subpackages dealing with

generic functionalities, like the label *encoders* used for the LCPN, the input *preprocessing*, the definition of *losses* and *callbacks* for the training of DL-based models, and several *classification_utils*, like the definition of the `argmaxrand` function which deals with tie-breaking events.

Noteworthy, the logic behind the global classifier approaches is implemented by the *keras_wrapper* and the *losses* packages, the former providing training procedures like multitask learning, task-incremental learning, and multiclass learning, and the latter implementing losses functions like hierarchical loss.

Finally, the source code for the last publicly accessible version of the Hierarchical Learning Engine, which covers the design of the BD-enabled LCPN approach and the intrusion detection solutions, is available at the following link: `https://github.com/jmpr0/hierarchical-framework`. The last version of the Hierarchical Learning Engine, whose package diagram is the one depicted in Fig. 2.12 and which includes the implementation of global classifier approaches, will be released with the related paper.

Part II

Use Cases

Chapter 3

Privacy: Classification of Anonymity Tools

In this chapter the Local Classifier per Parent Node (LCPN) model (Big Data-enabled for training) instance of our framework has been configured to solve the traffic classification of several privacy-preserving tools, i.e. Anonymity Tools.

3.1 Context of Classification of Anomymity Tools

Since its creation, the Internet has grown in importance (as the importance of the activities people do online), putting privacy and security assurance in the limelight. The preservation of user anonymity, in particular, has piqued the interest of the academic community, which has spent the last several years creating and developing a variety of privacy-preserving technologies,including proxy sites, virtual private networks, and Anonymity Tools (ATs). The latter function as intermediaries between Internet users and eavesdropping entities, allowing them to obscure the communication as well as the nature of the transferred materials, and as a result, a variety of AT are now publicly accessible, capable of hiding the identities of the people (source and destination) participating in the communication, in addition to the content of the communication itself, via encryption. The *most popular* ATs developed in recent years are The Onion Router (Tor)[1], the Invisible Internet Project (I2P)[2], and JonDonym[3]

[1] Tor Project. `https://www.torproject.org`.

[2] The Invisible Internet Project. `https://geti2p.net`.

[3] JonDonym. `http://anonymous-proxy-servers.net`.

(formerly known as Java Anon Proxy or Web-Mix). These solutions let users maintain their anonymity by encrypting data numerous times and passing it via many stations, each receiving only a portion of the data. From the user's perspective, ATs allow them to browse the web and operate applications while keeping their identity and location hidden from any intermediate entity viewing the traffic (also bypassing limitations imposed by providers or governments). As a result, it's difficult to track people and their activity throughout various networks. On the one hand, ATs aid authorities in the investigation of cybercrime, such as the sale of copyrighted or harmful software, narcotics, firearms, child pornography, and stolen digital identities, as well as the concealment of online frauds, extremism, hacking, and abuses. On the other side, they are critical for disseminating critical information through the Internet, such as when non-democratic actors impose censorship or for the single right to privacy, as cited by [13]. This latter point underpins their initial goal of maintaining the Internet as a fully accessible public utility. As a result, ATs like Tor are now widely used, with a user base of up to 2 million.[4] On the one hand, looking into the Traffic Classification (TC) of ATs is beneficial to designers since it puts their efficacy to the test, identifies flaws, and points the way to making them more robust. These studies, on the other hand, are of importance to both providers and government agencies, since they give knowledge that may be used to enforce informed engineering regulations or prohibit users from conducting undesired acts. As a result, categorizing ATs traffic is a highly intriguing and hard research subject, with current methods that may be improved.

Up to our knowledge, there are no studies focused on classification of different anonymity services at various levels of granularity via *hierarchical learning*. Accordingly, literature on TC of ATs via a "flat" (viz. non-hierarchical) approach is reviewed, including the conceptually-related Website Fingerprinting (WF) problem.

The analysis of ATs has been initially carried in *private networks*, e.g., with the aim of discriminating between HTTPS and Tor traffic [92]. In detail, by leveraging a dataset made of (i) regular HTTPS traffic, (ii) HTTP and (iii) HTTPS over a private Tor network, authors show that HTTP/HTTPS traffic over Tor can be detected with $\geq 93\%$ accuracy, employing Random Forest (Random Forest (RF)), Decision Tree (DT), and AdaBoost classifiers.

On the other hand, a few works focus on TC analyzing *real traffic* from anonymity networks, as most of the "experimental" literature explores anonymous WF, whose aim is to identify a webpage accessed by a client with en-

[4] `https://tinyurl.com/y5cucfno`.

crypted and anonymized connections by observing the patterns of data flows (e.g., packet size and direction). [93] propose a Multinomial Naïve Bayes (MNB) that relies on the normalized frequency distribution of IP packet sizes to tackle the WF problem in the context of different privacy-enhancing technologies (in a closed-world scenario) including Tor and JonDonym. Although Tor and JonDonym guarantee a better protection than other privacy-enhancing technologies (i.e., OpenSSL, OpenVPN), they prove to be not perfect (3% and 20% average accuracy, respectively). In [94] the same problem is tackled via a Support Vector Classifier, obtaining a gain of detection rate over [93] from 3% to 55% (resp. from 20% to 80%) in Tor (resp. JonDonym) network. On the other hand, in an open-world case, the detection rate drops with a maximum of 73% and 0.05% false-positive rate. More recently, in [95] a WF approach aimed to overcome limitation of previously devised alternatives is proposed and tested on a huge real-world representative dataset, exploring the limits of WF at Internet scale. Specifically, these are highlighted by a precision/recall drop with the size of the background sites which the monitored pages need to be distinguished from. Finally, we mention that the adoption of Deep Learning (Deep Learning (DL)) to WF is also a currently investigated topic. A novel DL-based method to deanonymize Tor traffic is proposed in [96] and tested on a dataset made of $\geq 3 \cdot 10^6$ network traces, with the best-performing DL model being $+2\%$ accurate than state-of-the-art attacks. In [97] a WF attack against Tor is developed, leveraging a Convolutional Neural Network and evaluated against state-of-the-art defenses (i.e. WTF-PAD and Walkie-Talkie). Results report an accuracy $> 98\%$ on undefended Tor traffic, while reaching $> 90\%$ accuracy (resp. 49.7%) when WTF-PAD (resp. Walkie-Talkie) is employed, with the attack remaining effective also in an open-world setting.

Moving to pure TC of ATs (based on real-data), [98] devise an approach based on Hidden Markov Models (HMMs) to classify four categories of Tor traffic (Peer-to-Peer (P2P), FTP, IM, and Web). HMMs are employed to build inbound and output models of the application types considered, and are fed with features based on burst volumes and direction of Tor flows, obtaining an accuracy up to 92%. [99] present a Machine Learning (ML)-based method employing Naïve Bayes (Naïve Bayes (NB)), Bayesian Network (Bayesian Network (BN)), functional and logistic model trees to recognize applications used by Tor users. Both *circuit-level* and *cell-level* information is leveraged for offline and offline/online classification, respectively. The highest accuracy achieved in the online (resp. offline) case equals 97.8% (resp. 91%). A similar setup is proposed in [100] for user activity recognition by means of four clas-

sifiers (NB, BN, RF, and DT) fed with *traffic-flow* and *circuit-level* features. Both approaches reach $\approx 100\%$ accuracy, with flow-based TC being *less demanding* and based on data that could be captured anywhere between the user and the Tor's relay. Along the same lines, [101] employ flow-based (statistical) traffic analysis to prove whether Tor Pluggable Transports (PTs) can evade censorship systems. Adopting a DT classifier (and based on a thorough analysis), the authors show that Tor PTs usage is recognizable, as PT-based obfuscation changes the content shape in a distinct way w.r.t. Tor (i.e. conferring to flows distinctive fingerprints). The effects of bandwidth sharing on I2P were analyzed by the same authors in [102] considering both application and user profiling achievable by an attacker. Using a DT classifier fed with flow-based features, the results show that users and applications on I2P *can* be profiled, with a harmful (resp. beneficial) effect of the shared bandwidth increase on application (resp. user) profiling accuracy.

Recently, [77] describe Anon17 public dataset comprising directional traffic-flows obtained by collecting data from three ATs (i.e. Tor, I2P, and JonDonym). Besides, detailed information about the traffic types and applications running on Tor and I2P is provided in the form of *three-level labels* for each flow. Up to our knowledge, the sole public dataset similar to Anon17 is that described in [103], containing however *only* Tor traffic of eight applications (browsing, audio, chat, mail, P2P, FT, VoIP, and video). Anon17 dataset is leveraged in [5], where three *flat classifiers* (that is, one per level, i.e. Anonymity Network, Traffic Type, and Application), are employed to perform TC, also in its "early" variant [18]. In detail, at each level the performance of five ML classifiers (NB, MNB, BN, DT, and RF) is compared, highlighting insights about the number and nature of relevant features needed for an accurate TC. Results show that the anonymity networks (i.e. Tor, I2P, and JonDonym) can be easily distinguished (up to 99% F-measure). Differently, the specific application generating traffic can be classified with up to 69% F-measure. At both levels, early TC achieves worse results.

3.2 Hierarchical Framework Instances for Anonymity Tools

To provide a clear example of how the proposed framework performs on a valued practical application, we refer to the traffic generated by ATs. Still, we remark that the proposed framework *is not limited* to ATs, albeit being motivated also by this practical application. In fact, our framework is designed

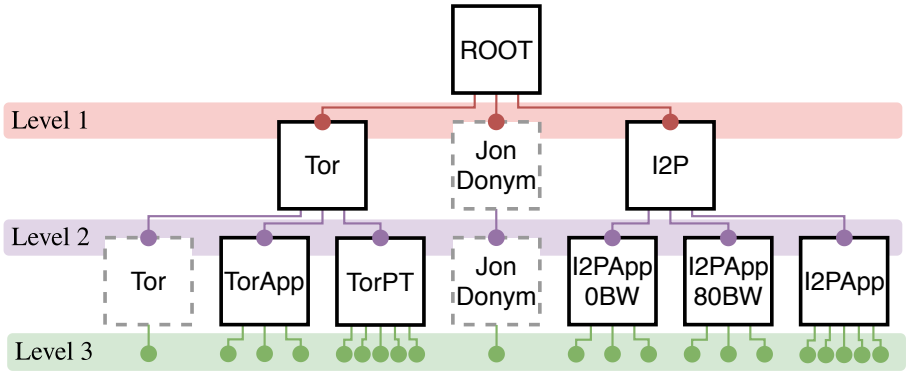


Figure 3.1: Traffic classifier for AT based on the LCPN approach. Classifiers (solid black squares) distinguish among several classes (dots). Dashed grey squares correspond to degenerate (i.e. single class) classifiers.

to benefit from any hierarchical classification model of network traffic, e.g., mobile apps (arranged as categories, apps, and versions) [14], also including flat models as simpler cases.

For example, in our considered scenario (where we leverage the subsequently introduced Anon17 dataset) we face $T = 3$ granularity levels for TC. The first level focuses on identifying the AT used to transport traffic, i.e. assigning $\hat{\ell}_1$ from $L_1 = 3$ classes. The second delves into classification of the type of transport service offered by the specific AT, i.e. assigning $\hat{\ell}_2$ from $L_2 = 7$ classes. Finally, the finest granularity is associated with labeling the specific application tunneled in the AT with a given transport service, i.e. assigning $\hat{\ell}_3$ from $L_3 = 21$ classes. Noteworthy, the training set size of each classifier node $|\mathrm{T}_n|$ (compared with $|\mathrm{T}|$, required when training a flat TC approach), is shown in Fig. 3.9d. Moreover, model parallelism (induced by LCPN) leads to the decomposition of the original classifier with $L_3 = 21$ classes, supported by the whole training set T, into $N_c = 8$ independently trainable simpler nodes considering at most five classes (Fig. 3.1) with smaller training sets T_n. The impact on the training time of each T_n (viz. complexity) is further minimized by data parallelism, which allows splitting each by the number of workers considered. Our experimental validation (Sec. 3.3) shows how, in the application of the proposed framework, there actually is an improvement in terms of both training time/cost and classification performance. To this end, henceforth we leverage the Big Data (BD) infrastructure supporting the LCPN classifier.

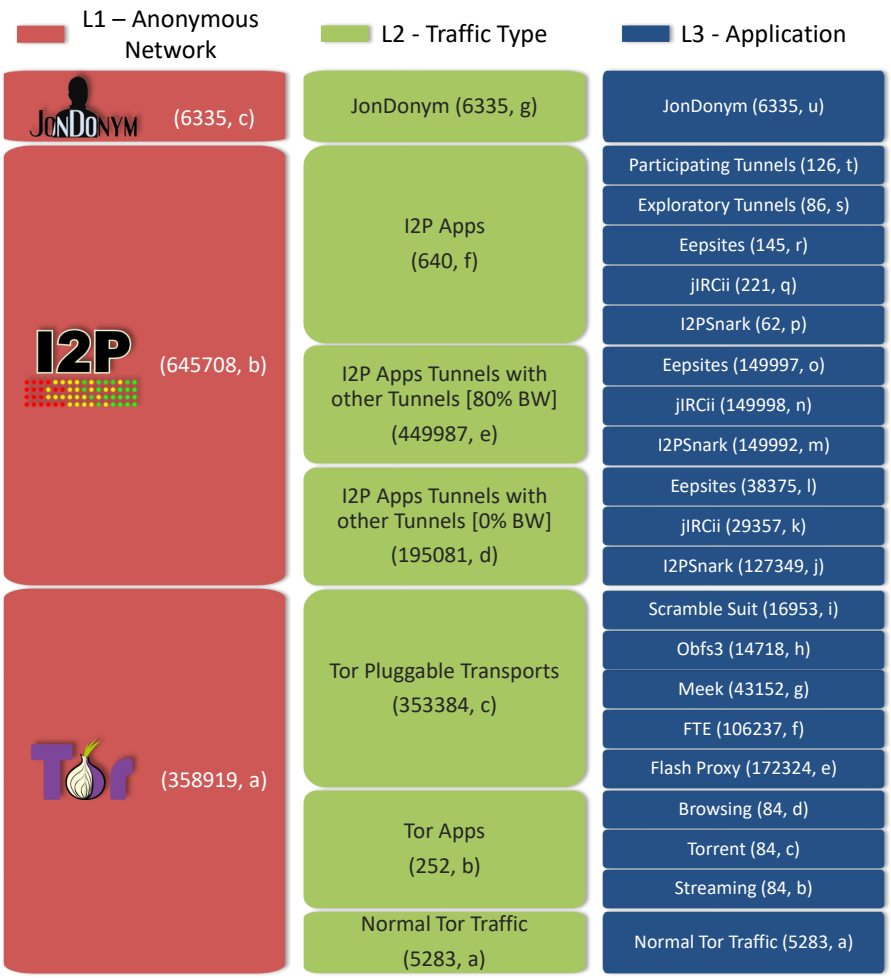


Figure 3.2: Anon17 Classification Levels: Anonymous network (L1), Traffic Type (L2) and Application (L3), with total no. of samples per class and class label at each level.

3.2.1 Dataset Description

As mentioned above, the dataset used to validate our proposal is Anon17, which was collected in a *real-network environment* at the NIMS Lab [77] during 2014–17 and gathers traffic from Tor, I2P, and JonDonym. The dataset has been labeled leveraging the information provided by the anonymity services themselves. Indeed, it provides labels at different levels: (i) *Anonymous Network Level* ($L_1 = 3$ classes); (ii) *Traffic Type Level* ($L_2 = 7$ classes); (iii) and *Application Level* ($L_3 = 21$ classes). This label organization promotes analyses of ATs traffic at different levels of granularity, as well as the implementation of hierarchical approaches, as remarked in Fig. 3.2. Precisely, the dataset contains $\approx 1.46M$ flows for a total of $\approx 430M$ packets. We point to [77] for obtaining exhaustive information on Anon17 dataset. In this study, we down-sample to 5% the most-populated traffic-type classes of the original dataset, adopting a pre-processing strategy similar to [5], so as to mitigate class imbalance. This represents a safe choice due to the high number of samples of "majority" classes.[5]

The dataset is released in the form of 74 statistics per flow (that can be extracted also from encrypted traffic). Thus, the Traffic Object (TO) we refer to in our study is the *flow*. We leverage the *full set of features in our experimentation for all nodes*. These choices are dictated by the type of analysis to perform, which could be either batch (also termed post-mortem), or streaming (also termed real-time). Our choices are suitable for the batch TC task we aim to address. This notwithstanding, our framework can accommodate alternative design choices.

According to the above dataset description, we tackle the problem of classification of ATs traffic assuming that we are in the presence of AT traffic only, similarly to [5]. The depicted scenario refers to a context where an *upstream classifier* has been able to separate AT traffic from a clear or standard-encrypted one (e.g., as shown by [92] for Tor network). Hence, the aim of the proposed approach is to assess discrimination of anonymity services and related applications *once this AT traffic has been separated from other traffic*. More generally, the results of our analysis can be intended as an upper bound on the ATs classification performance in the case of an *open-world* assumption. Indeed, the use of an upstream classifier would perfectly fit within the hierarchical approach evaluated in this chapter (viz. LCPN), with design and evaluation of the whole AT-TC system (truly operating in an open-world

[5]Experimental results, not shown, have highlighted negligible difference with the (additional) use of oversampling of "minority" classes.

scenario) left as future work.

Based on the nature of the traffic in Anon17 dataset, the TC here considered is arranged in three levels, corresponding to anonymity networks, traffic types, and specific applications. As a whole, the LCPN instance of our framework resorts on *one* classifier at L1, *two* classifiers at L2 and *five* classifiers at L3.

3.2.2 Traffic Object and Features

Anonymous traffic contained in Anon17 has been split into different *flows* (see Sec. 2.1), by means of the flow-exporting tool *Tranalyzer2* [104], constituting the TC object here employed. We highlight that the direction of each flow (considered as a feature) is indicated as "A" or "B" for client-to-server and vice versa, respectively. We note that the proposed hierarchical classification approach could even operate at the *packet-level*, in principle (viz. the TC object could be the *single packet*[6]). For each traffic flow, Anon17 provides different sets of features extracted via Tranalyzer2 [104]. In this work, we consider *two* different feature sets, referred hereinafter to as `TC_set` and `EarlyTC_set`. In brief, `TC_set` capitalizes complete traffic flows, while `EarlyTC_set` only relies on the first K packets of each flow, thus enabling early TC.[7]

In detail, `TC_set` originally refers to 81 per-flow statistical features. Repeated fields (such as those related to packet length and packet size) and (initial/final) timestamps have been removed to avoid overestimated results.[8] As a result, the employed feature set consists of 74 statistics comprising:

1. flow direction and duration,

2. *packet length* (`PL`) and *inter-arrival time* (`IAT`) statistics (mean, min, max, median, quartiles, etc.),

3. TCP[9] and IP header-related features (window size, sequence number, TCP and IP options, etc.),

[6] Up to our knowledge, there is no work in literature tackling anonymous TC at this granularity.

[7] We highlight that other feature sets (e.g., histograms), leading to lower performance, have been considered in [5].

[8] Timestamp values depend upon the process adopted for collecting traces, potentially introducing classification artifacts.

[9] TCP-related features have zero-value if the flow leverages UDP as transport protocol (e.g., I2P network works on both TCP and UDP).

4. number of Tx/Rx bytes and packets,

5. number of distinct hosts connected to flow source or destination IP (during its lifetime),

6. number of concurrent flows sharing the same (source IP, destination IP) pair (regardless of source & destination ports).[10]

Differently, `EarlyTC_set` is made of the sequence of pairs `(PL, IAT)` of the first K packets of each flow. In the rest of this chapter, we will employ `TC_set` when referring to standard TC, whereas adoption of `EarlyTC_set` will be assumed just for early TC.

Finally, for `TC_set` features, we consider feature selection, based on a *filtering approach*, ranking the elements of the set based on the relative importance of each feature (so as to skim the more informative ones), in terms of *mutual information* with the class (random) variable, whereas for `EarlyTC_set` features, ranking is performed according to a time constraint (i.e. only the first K packets are employed). We remark that, for the `TC_set`, feature extraction techniques, such as PCA, could be easily adopted in the proposed hierarchical classification framework without any substantial change.

3.2.3 Models

Herein, we consider as potential nodes *four* different ML-based classifiers, i.e. the DT, the RF, the NB, and the BN. Indeed, these classifiers have been successfully employed in several works tackling TC of anonymous traffic [99, 100, 102]. Nonetheless, as the proposed hierarchical classification framework is general, other ML (e.g., Support Vector Machine (SVM), Gradient Boosting, etc.) or even DL classifiers could be adopted with no substantial change.

Specifically, based on the related literature [101, 105], we have identified four relevant ML models: two tree-based models, (i.e. Random Forest (RF) and Decision Tree) and two Bayesian-based models, (i.e. Naïve Bayes and Bayesian Network). Deepening about these algorithms are reported in the dedicated Sec. 2.3.4. In this use case, for the latter two algorithms we will adopt the variants shown to perform better in [5], i.e. NB_SD (supervised discretization is used for numerical features) and BN_TAN (the dependence model is reduced to a simpler tree).

[10] We point to the user manual of Tranalyzer2 (`https://tranalyzer.com`) for further details about these features.

3.3 Experimental Results of Classification of Anomymity Tools

In this section, we show experimental results aimed at investigating anonymous TC performance via the proposed hierarchical framework, also when considering early TC, by leveraging the LCPN approach. First, the performance of the proposed framework is analyzed and compared with the best flat classifier, showing that the usage of a "naïve" LCPN architecture is able to introduce non-negligible improvements, being the result of decomposition in *simpler TC sub-tasks*. Then, the results of design improvement are shown, deepening the impact of both rough- and fine-grained optimization choices—the former consisting in varying the number of features of the classifiers (keeping the classifier type fixed) in the hierarchy with the same increment, while the latter involving changes to both features and classifier types. Concerning the early-TC scenario, only final results pertaining to fine-grained optimization are reported for brevity. Also, the evaluation contains finer-grained analyses of the error patterns of these two different classification "philosophies" along with the severity of errors (thus leading to interesting conclusions on the ATs considered). Then, a first investigation of classification performance obtained by resorting to progressive censoring in the hierarchical case is reported, and compared with the effects of censoring on a flat classifier baseline.

Moreover, results regarding the BD-enabled training of the LCPN classifier are provided. In detail, we first show the performance achievable with pure data parallelism. Then, we evaluate the impact of adopting different scheduling strategies to assign tasks to buckets, assessing pure model parallelism. Finally, we experimentally evaluate the benefits originated by the capitalization of data and model *double-parallelism* in our framework (Sec. 2.3.2).

3.3.1 Naïve Hierarchical vs. Best Flat Classifier

In Tab. 3.1 we report the performance of the best flat classifiers (e.g., the configurations with an optimal number of features in terms of F-measure) as derived from [5] and resulting in a RF fed with 50, 35, and 65 features at L1, L2, and L3, respectively. Performance is reported in terms of accuracy, F-measure, and G-mean at each classification level.[11] Such optimal setup is compared with a first naïve implementation of our hierarchical classification approach,

[11] "n.d." points out unavailable performance for flat classifiers at levels deeper than that considered for classification.

Table 3.1: Accuracy, F-measure, and G-mean (%) of the best flat classifier (RF) with {optimal number of features} at each level compared to naive hierarchical configuration. Results are in the format *avg. (± std.)* over 10-folds.

Classifier	Metric	L1	L2	L3
Best Flat L1 {50}	*Accuracy*	99.80±0.03%	n.d.	n.d.
	F-measure	99.80±0.04%	n.d.	n.d.
	G-mean	99.83±0.03% ♢	n.d.	n.d.
Best Flat L2 {35}	*Accuracy*	99.75±0.06%	97.01±0.24%	n.d.
	F-measure	99.73±0.06%	94.30±0.35%	n.d.
	G-mean	99.80±0.05%	96.19±0.29%	n.d.
Best Flat L3 {65}	*Accuracy*	99.70±0.06%	96.77±0.24%	73.52±0.40%
	F-measure	99.71±0.06%	93.51±0.58%	71.14±1.05%
	G-mean	99.79±0.04%	95.71±0.34%	82.73±0.57%
Naïve Hierarchical {65}	*Accuracy*	99.81±0.06% ⋆	97.17±0.24% ⋆	74.60±0.48% ⋆
	F-measure	99.81±0.06% †	94.43±0.75% †	73.82±1.42% †
	G-mean	99.83±0.05%	96.23±0.39% ♢	84.35±0.74% ♢

Legend: Best Accuracy (⋆), F-measure (†), and G-mean (♢) per level.

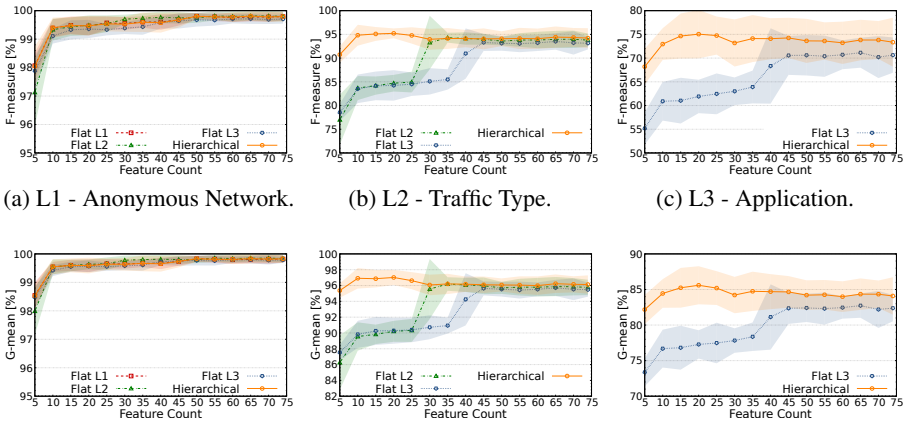
obtained by using the best L3 flat configuration (RF + 65 features) in *all* the classifier nodes of the hierarchy. Unsurprisingly, L1 performance metrics report a score $\geq 99.7\%$: traffic generated through different anonymity networks is easily distinguishable from each other, confirming that these tools are designed to provide anonymity but not to hide the usage of the tool itself. Also, results show that even a naïve hierarchical solution improves L3 performance (up to $+2.68\%$ F-measure)[12] w.r.t. the best L3 flat classifier, and performs on a par in terms of L1–L2 levels when compared to level-optimized best flat classifiers. This is due to split of the original TC task (21 classes at L3) into smaller tasks (at most, 5 classes).

3.3.2 Optimization Results

Impact of Feature Selection

As a first step towards the optimization of the proposed approach, we investigate here the performance of the framework when varying the number of features used by each classifier but *considering a common number of features (here denoted M) for each node in the tree*, and *keeping the classification algorithm fixed to RF* (that is, the best-performing one in the flat case [5]). We remark that although there is a common M for all nodes, the specific set of

[12] Also, from a statistical significance viewpoint, we observed a gain of hierarchical classification approach in 100% of the cases over the considered folds.



(a) L1 - Anonymous Network. (b) L2 - Traffic Type. (c) L3 - Application.

(d) L1 - Anonymous Network. (e) L2 - Traffic Type. (f) L3 - Application.

Figure 3.3: F-measure (a–c) and G-mean (d–f) [%] of hierarchical and flat classifiers: RF fed with different subsets of features in `TC_set` (from 5 to 74 with increments of 5). Average on 10-folds and corresponding $\pm 3\sigma$ confidence interval are shown.

features *may differ*.

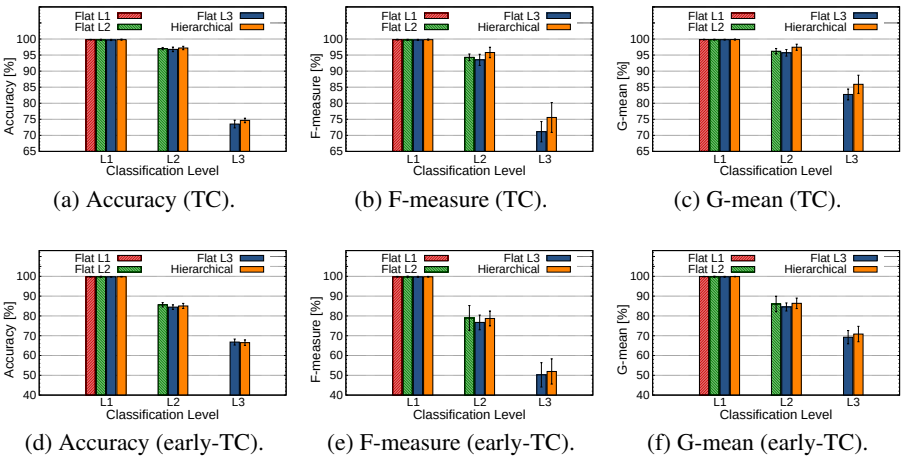
Fig. 3.3 summarizes the obtained results—with a view to finding the optimal value M *for the entire hierarchy*—also providing a comparison against the three flat classifier counterparts (in terms of F-measure and G-mean, as accuracy showed similar trends). First, the results highlight that there is no appreciable performance difference among L1–L3 flat classifiers and the hierarchical approach by looking at L1 metrics, and only a slight performance improvement is observed with a high number of features (performance saturation is observed with at least 10 features) is achieved. On the other hand, at L2 slightly higher performance are obtained by the hierarchical approach with a lower number of employed features per node (approximately 10–20), whereas at L3 (being the harder task) the following key observations can be made:

1. the hierarchical approach provides a non-negligible improvement over the L3 flat classifier for all the range considered,

2. the best performance of the hierarchical classification is attained with a smaller number of features.

These considerations apply to both F-measure and G-mean.

Fine-grained Optimization



(a) Accuracy (TC). (b) F-measure (TC). (c) G-mean (TC).

(d) Accuracy (early-TC). (e) F-measure (early-TC). (f) G-mean (early-TC).

Figure 3.4: Accuracy (a), F-measure (b) and G-mean (c) of the best classifiers and of their early-TC counterparts (d, e, and f). Average on 10-folds and corresponding $\pm 3\sigma$ confidence interval are shown.

Herein the fine-grained framework optimization, in case of features from `TC_set`, is discussed. In detail, with the aim of trading off design complexity with performance, *we remove the constraints previously introduced, and allow each node in the hierarchy to be optimized in terms of both the number of features and the classifier type*. We consider *four* different ML-based classifiers (DT, RF, NB_SD and BN_TAN). We remark that, to avoid a combinatorial explosion of the optimization problem, we resort to the *per-node* optimization rationale. Based on the above rationale, in Fig. 3.4 we report the classification performance in terms of accuracy (Fig. 3.4a), F-measure (Fig. 3.4b) and G-mean (Fig. 3.4c), by comparing the flat classification approaches with the per-node optimized hierarchical classifier. The proposed (optimized) hierarchical classifier is able, at least, to perform at t^{th} ($t = 1, 2, 3$) level as well as the corresponding flat classifier *explicitly designed* to solve the classification task of the same level. In detail, such optimized hierarchical approach is able to achieve 99.81% (resp. 99.83%), 95.81% (resp. 97.44%) and 75.56% (resp. 85.89%) F-measure (resp. G-mean) score at L1, L2, and L3, respectively. These results (almost) represent a tie at L1, whereas $+1.51\%$ (resp. $+1.73\%$) and $+4.42\%$ (resp. $+3.16\%$) gains are experienced at L2 and

L3, respectively.[13] The details of the optimized hierarchical classification are reported in Fig. 3.5a, where for each classifier node the employed classifier and the number of features are reported. Remarkably, RF denotes the best classifier for each node-specific classification task, while only for the classifier of *Tor App* applications BN_TAN provides higher performance. Instead, the variability of the optimal number of features underlines no clear trend, except that usually I2P-related node classifiers require a lower number of features, at least when leveraging those in `TC_set`.

Optimization for Early TC

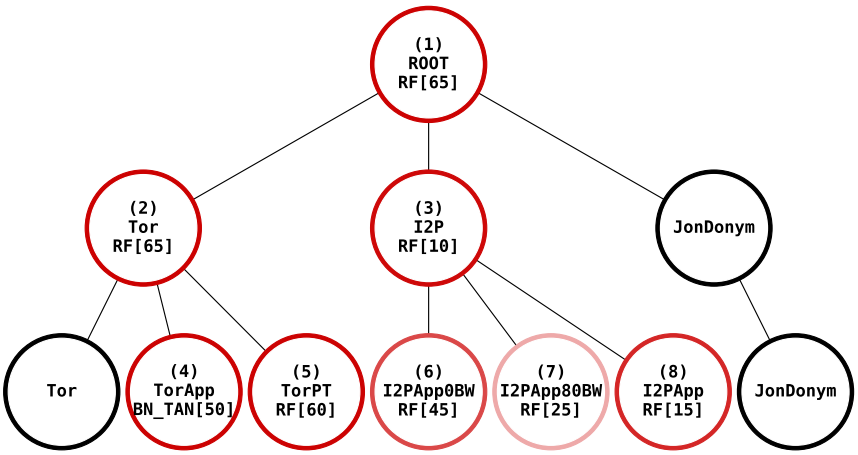
Herein we evaluate the hierarchical framework when the classifiers are fed with features in `EarlyTC_set`, i.e. considering `PL`s and `IAT`s of the first $K = 1, \ldots, 16$ (non-zero payload) packets. This analysis helps assessing the framework capability in supporting early TC, i.e. to evaluate how soon and to which degree ATs and related services can be identified. To this end, we have paralleled the previous investigations in the early-TC scenario. Nonetheless, herein we omit the details for brevity, and comment only the final results. We remark that the *main difference* with the previous analysis concerns the *feature selection* process, herein performed on a *time-basis* (i.e. only the features drawn from the first K packets are considered).

First, results show that a naïve hierarchical "extension" of best flat L3 classifier (in this case a `BN_TAN` with $K = 11$ packets) is *not* able to provide improved performance (e.g., 48.80% F-measure at L3, as opposed to 50.23% in the flat case). This result highlights a key difference with respect to the scenario leveraging `TC_set` and emphasizes the *need for hierarchical-specific optimization* in such case. Secondly, we investigate (as in Fig. 3.3) how varying the features corresponding to the first K packets could improve the performance of the naïve hierarchical extension, and compare it with the best flat counterpart. Our investigations reveal that a performance saturation is observed after ≈ 10 packets, and that the hierarchical classification approach is able to improve best flat L3 approach in terms of G-mean, whereas an F-measure drop is observed for all values of K considered. Finally, fine-grained optimization (see Figs. 3.4d, 3.4e, and 3.4f) provides higher performance with respect to the optimized flat case, e.g., $+1.71\%$ F-measure and $+1.59\%$ G-mean at L3.[14]

[13] Also, from a statistical significance viewpoint, we observed a gain of hierarchical classification approach in 97.5% of the cases over the considered folds.

[14] Also, from a statistical significance viewpoint, we observed a gain of hierarchical classifi-

The corresponding optimized hierarchical structure is shown in Fig. 3.6a and—when compared to Fig. 3.5a—clearly shows that per-node optimized classifiers significantly differ in type and number of features, thus motivating the need for fine-tuned optimization of the proposed hierarchical classification. Although the hierarchical classification gain is not significant, its operating principle allows to delve into the "information structure" of the TC problem and highlight critical points by excluding potential performance drops due to the size of the classification task, as shown by the following *per-node* performance analysis.



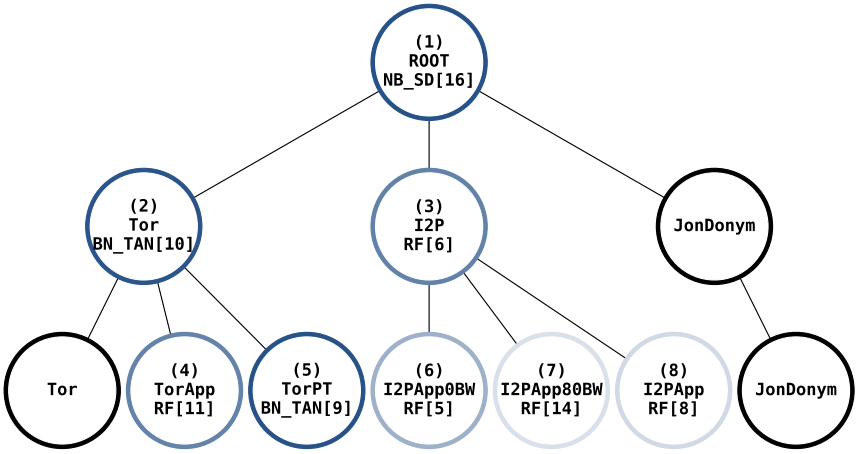
(a) Fine-grained optimized hierarchical structure **with `TC_set`**.

Classifier	#Classes	Accuracy	F-measure	G-mean
$C_0 \rightarrow$ (1)	$L_1 = 3$	99.81 ($\pm$ 0.06)	99.81 ($\pm$ 0.06)	99.83 ($\pm$ 0.05)
$C_1 \rightarrow$ (2)	$\overline{L}_1 = 3$	99.97 ($\pm$ 0.04)	99.91 ($\pm$ 0.20)	99.97 ($\pm$ 0.04)
$C_2 \rightarrow$ (3)	$\overline{L}_2 = 3$	95.05 ($\pm$ 0.29)	91.53 ($\pm$ 1.06)	93.29 ($\pm$ 0.79)
$C_{11} \rightarrow$ (4)	$\overline{L}_{11} = 3$	99.58 ($\pm$ 1.25)	99.58 ($\pm$ 1.25)	99.69 ($\pm$ 0.94)
$C_{12} \rightarrow$ (5)	$\overline{L}_{12} = 5$	99.72 ($\pm$ 0.12)	99.44 ($\pm$ 0.21)	99.63 ($\pm$ 0.14)
$C_{21} \rightarrow$ (6)	$\overline{L}_{21} = 3$	72.42 ($\pm$ 1.32)	58.43 ($\pm$ 1.63)	69.00 ($\pm$ 1.26)
$C_{22} \rightarrow$ (7)	$\overline{L}_{22} = 3$	48.94 ($\pm$ 0.51)	48.90 ($\pm$ 0.52)	60.37 ($\pm$ 0.42)
$C_{23} \rightarrow$ (8)	$\overline{L}_{23} = 5$	75.12 ($\pm$ 5.95)	72.50 ($\pm$ 6.99)	81.68 ($\pm$ 4.58)

(b) Performance metrics **with `TC_set`**.

Figure 3.5: Fine-grained optimized hierarchical structure with `TC_set` (a). Optimal number of features for each classifier (node) is shown in square brackets. Lighter colors point to worse performance. Related per-node metrics are shown in (c) (with $< 60\%$ F-measure nodes highlighted in gray).

cation approach in 90.0% of the cases over the considered folds.



(a) Fine-grained optimized hierarchical structure **with `EarlyTC_set`**.

Classifier	#Classes	Accuracy	F-measure	G-mean
$C_0 \rightarrow (1)$	$L_1 = 3$	99.80 ($\pm$ 0.05)	99.78 ($\pm$ 0.06)	99.87 ($\pm$ 0.04)
$C_1 \rightarrow (2)$	$\overline{L}_1 = 3$	99.20 ($\pm$ 0.12)	90.48 ($\pm$ 1.61)	94.82 ($\pm$ 1.27)
$C_2 \rightarrow (3)$	$\overline{L}_2 = 3$	72.20 ($\pm$ 0.81)	60.61 ($\pm$ 1.70)	66.85 ($\pm$ 1.02)
$C_{11} \rightarrow (4)$	$\overline{L}_{11} = 3$	63.42 ($\pm$ 4.92)	63.27 ($\pm$ 4.92)	72.01 ($\pm$ 3.85)
$C_{12} \rightarrow (5)$	$\overline{L}_{12} = 5$	99.69 ($\pm$ 0.07)	99.55 ($\pm$ 0.18)	99.63 ($\pm$ 0.13)
$C_{21} \rightarrow (6)$	$\overline{L}_{21} = 3$	67.37 ($\pm$ 0.81)	44.56 ($\pm$ 1.45)	56.38 ($\pm$ 0.97)
$C_{22} \rightarrow (7)$	$\overline{L}_{22} = 3$	43.47 ($\pm$ 0.75)	43.13 ($\pm$ 0.72)	55.76 ($\pm$ 0.63)
$C_{23} \rightarrow (8)$	$\overline{L}_{23} = 5$	50.67 ($\pm$ 9.21)	37.75 ($\pm$ 11.24)	58.04 ($\pm$ 7.81)

(b) Performance metrics with **with `EarlyTC_set`**.

Figure 3.6: Fine-grained optimized hierarchical structure with `EarlyTC_set` (a). Optimal number of features for each classifier (node) is shown in square brackets. Lighter colors point to worse performance. Related per-node metrics are shown in (b) (with $< 60\%$ F-measure nodes highlighted in gray).

3.3.3 Fine-grained Results

Per-node Detailed Classification Performance

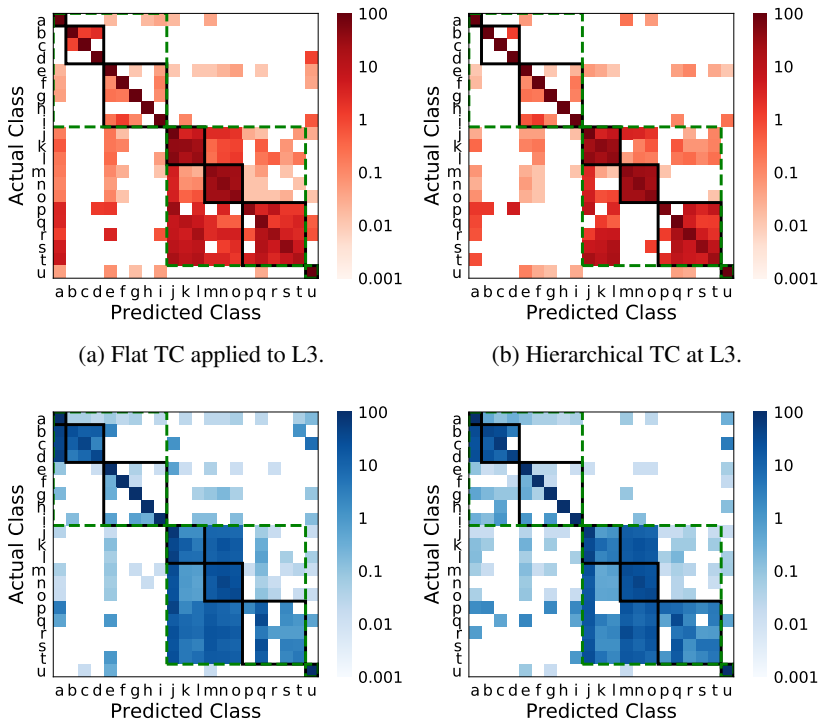
For completeness, we report in Figs. 3.5b and 3.6b the detailed per-node classification performance, corresponding to `TC_set` and `EarlyTC_set`, respectively. The following interesting observations can be made on the reported tree representation. First, with respect to *Anonymous Network Level* classification (L1) nodes present near-ideal performance both when relying on `TC_set` and `EarlyTC_set`, thus showing *(almost) no errors propagating from hierarchical classification of ATs*. Secondly, at *Traffic Type Level* (L2) both

approaches based on `TC_set` and `EarlyTC_set` present near-ideal performance in classifying *Tor* traffic types, whereas some performance degradation is observed in classification of I2P traffic types; this phenomenon is more penalizing in the early-TC case, e.g., with *I2P* F-measure drops down to 60.61%. This represents one of the main causes of performance difference among the two scenarios, as all these *errors are propagated downwards*. Finally, at *Application Level* (L3), the behavior of the classifier nodes is more varied. Indeed, referring to approach based on `TC_set`, it is apparent that Tor nodes (`TorApp` and `TorPT`) work well, whereas on I2P nodes significant degradations (higher than those at L2) are observed, with *I2PApp80BW* the *most significant*. Differently, discrimination within each I2P traffic type and *TorApp* is only possible with $< 65\%$ F-measure with features in `EarlyTC_set`. Therefore, classification within *I2PApp80BW* cannot be accurately attained with neither of the considered feature sets (confirming the intuition that *I2PApp80BW* represents the traffic obtained by mixing different apps), and the advantage of the classification task split into subproblems cannot exceed a certain threshold in this case, due to the impossibility of discerning applications within this traffic type, resorting to the set of the available features.

Per-Class Performance Breakdown

Since classification at L3 is a challenging task (but also the most interesting from a user's privacy perspective), we report in Fig. 3.7 the per-class performance breakdown of hierarchical classification results in terms of confusion matrices at L3, comparing them against results obtained with flat approach. We recall that for these matrices the higher the concentration toward the main diagonal, the better the overall performance. Furthermore, to highlight how errors at L3 which do not imply misclassification at L1/L2 are *less severe* and should be *promoted* as opposed to the others, in the same figures we also highlight the error patterns corresponding to the same traffic type (solid boxes) and the same AT (dashed boxes).

First, comparing flat and hierarchical approaches (Figs. 3.7a and 3.7b, respectively), we can observe a *reduction of error-patterns* in the latter case, highlighting the beneficial "divide-et-impera" principle of hierarchical classification. Secondly, confusion matrices at L1 (dashed boxes) show that classifiers based on both approaches (with different quantitative outcomes) present error patterns which almost entirely lead to a misclassification of the traffic type *within the same* anonymous network. Differently, referring to L2 standpoint (solid boxes), hierarchical classification provides *improved capabilities*



(a) Flat TC applied to L3.

(b) Hierarchical TC at L3.

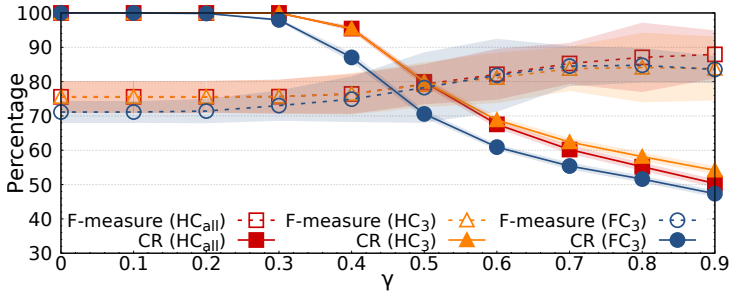
(c) Flat early TC applied to L3.

(d) Hierarchical early TC at L3.

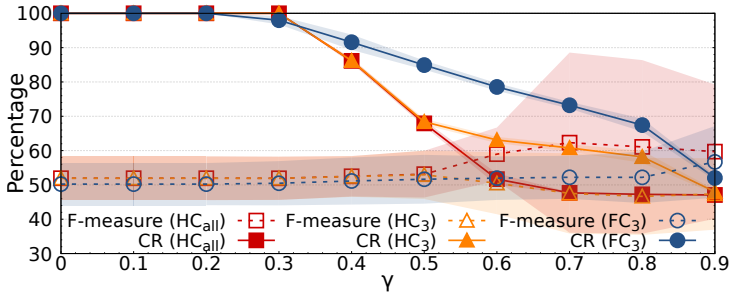
Figure 3.7: L3 Confusion matrices ([%] in log scale) of best flat and hierarchical classifiers in standard TC (red) and early TC (blue).

in confining errors to the same traffic type (especially in the case of I2P traffic). A similar consideration applies to early TC results (Figs. 3.7c and 3.7d) and, in particular, to errors concerning the applications of *I2P Apps* and *Tor Apps*. Hence hierarchical classification approach performance clearly highlights the inability, *not depending on the size of the classification task*, in satisfactorily discriminating (with an early TC setup in mind) among I2P traffic types and within their corresponding applications, along with those in `Tor Apps`. On the other hand, results confirm the outcomes of [105], witnessing that the obfuscation implemented by `Tor Pluggable Transports` induces a class fingerprint easily distinguishable ($\geq$ 99% accuracy, see C_2 classifier in Fig. 3.6) from both `Normal Tor Traffic` and `Tor Apps`.

3.3.4 Performance with Reject Option



(a) TC.



(b) Early TC.

Figure 3.8: F-measures and Classified Ratios (CRs) of best classifiers vs. γ.

Finally, in Fig. 3.8 we focus on the adoption of censoring threshold(s) for flat classification and hierarchical classification of ATs. Specifically, we report L3 performance, being the hardest task considered. We recall that, although in the flat case there is only a single tunable γ (referred to as FC_3), hierarchical classification allows to set a different threshold value at each node (e.g., for C_{ij} the threshold γ_{ij} can be adjusted independently from the others. Nonetheless, as a preliminary investigation on the censored behavior of the hierarchical classification approach—to avoid cumbersome analyses—we consider two simplified options: considering only a common γ value shared by (i) *all the nodes* in the hierarchy ($hierarchicalclassification_{all}$) and ($ii$) *solely* by the classifier nodes concurring to L3 classification ($hierarchicalclassification_3$). Accordingly, we report the F-measure vs. γ (similar trends have been observed for other metrics) along with the corresponding trend of the ratio of classified samples, viz. CR.

Although using such thresholds cannot be intended as a *panacea* able to

cope with high-confidence wrong predictions, results pertaining to the adoption of `TC_set` show performance improvement with increasing γ. In detail, both hierarchical variants offer performance gain with higher CR (less discarded flows) w.r.t. FC_3 (e.g., $\approx 80\%$ F-measure is attained with $\approx 80\%$ CR), with $hierarchicalclassification_{all}$ having slightly improved performance (while incurring in a slightly higher CR, due to the presence of non-zero thresholds for nodes at higher levels in the hierarchy). On the other hand, in the early-TC scenario, the CR is lower for hierarchical approaches, while $hierarchicalclassification_{all}$ provides slightly improved performance than FC_3, whose performance do not benefit from a γ increase. Nonetheless, such results underline how progressive censoring via per-node thresholds may be a viable option for further performance improvement.

3.3.5 Big Data-enabled training Results

Pure Data Parallelism.

In this campaign, we inspect the impact of sole *data parallelism*, hence we do not take into account model parallelism, i.e. we assign all the nodes to a single bucket ($B = 1$, no scheduler needed). In detail, we perform the training of the N_c classifiers of the hierarchy *sequentially*, submitting each training task at a time to a single master. Accordingly, we analyze the trend of the metrics varying the number of workers N_w linked to the master, namely we perform per-classifier data parallelism. We compare the adoption of data parallelism on hierarchical classification (curve "`Hierarchical`") against the distributed *flat* approach (curve "`Flat`"), in terms of training completion time (Fig. 3.9a) and related costs (Fig. 3.9b). By distributed flat approach, we mean a single RF classifier trained at granularity level $g = G = 3$ using data-parallelism (e.g., splitting the whole training set T among the workers N_w, which are then coordinated by the master for performing the learning task. For completeness, we also report the respective *theoretical baselines* ("`Hierarchical BL`" and "`Flat BL`"), defined as the completion time and cost for the centralized version divided by the number of workers N_w. In other words, the latter are *ideal curves* which do not take into account the synchronization overhead and allow to appreciate its impact (by comparison) on realistic data-parallel approaches.

As shown in Fig. 3.9a, the training completion time shows an asymptotic behavior, almost flattening at around 8–10 workers. The comparison between hierarchical and flat training completion time shows an almost constant gain

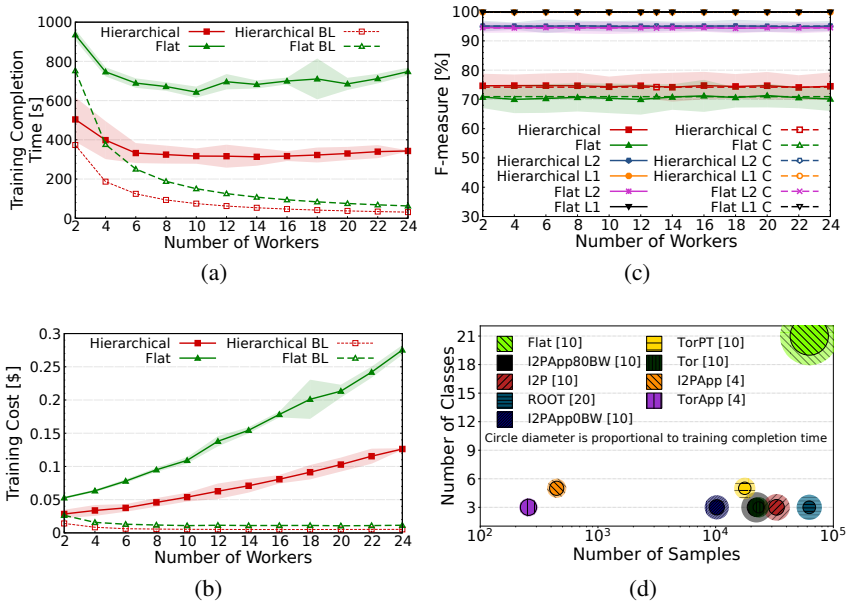


Figure 3.9: **On the left**: Completion time **(a)** and cost **(b)** of training phase for hierarchical and flat approaches against their respective theoretical baselines (— BL), varying the number of workers.
On the right: Classification performance (F-measure) **(c)**, at all 3 granularity levels, hierarchical and flat approaches against their respective centralized version (— C), varying number of worker machines. Dashed lines refer to configuration with only one worker machine, i.e. no data parallelism. Complexity map **(d)**, where each circle represents a classifier. For each, inner and outer borders refer to the training completion time with optimized data-parallel configuration (the optimal number of workers is shown in squared brackets) and centralized configuration (with one worker), respectively.
Graph values are provided as $\mu \pm 3\sigma$, corresponding to a confidence interval of 99.75%.

of $\approx 50\%$ for the former approach, e.g., with 10 workers the training phase is completed by the hierarchical classification and flat approach in $316.97 \pm 19.83\,s$ and $643.01 \pm 28.32\,s$, respectively. The above gain originates from the simpler classification tasks associated to the nodes in the hierarchy and the smaller training sets T_n (Fig. 3.9d). Notably, all training completion time curves do not follow the respective theoretical baselines, as the actual speedup

is burdened by the overhead imposed by the master–worker communication. Accordingly, as shown in Fig. 3.9b, the reduction of the training completion time does not result in cost savings, with cost curves showing a monotonically increasing trend.

In Fig. 3.9c we compare the F-measure vs. the number of workers of the hierarchical and flat classifiers against their centralized counterparts ($N_w = 1$). F-measure is shown for all the three levels of TC granularity. Beyond the effective gain of hierarchical classification (e.g., red vs. green curves), it is apparent that the performance remains stable when N_w grows. This result may seem counterintuitive at first, since the fragmentation of the training set could negatively impact the performance of ML classifiers [56]. Still, such weak dependence could be explained by the parallel and ensemble nature of the RF classifier. Indeed, the aforementioned peculiarity leads to a federated RF version (on Spark) which suffers less from compressed intermediate exchanges among workers (via the master), as opposed to other algorithms. Hence, this makes it more resilient to the fragmentation of the training set. We remark that, in general, this is not the case for ML approaches, e.g., see neural networks in [56] for a mobile TC task.

Finally, in Fig. 3.9d we provide the fine-grain analysis of the training completion time for the flat-approach as well as for each node of the hierarchical classification hierarchy (as shown in Fig. 3.1). Therein, the diameter of each circle is proportional to the training completion time of the related classifier (t_n). In detail, for each classifier the inner circle refers to the optimized data parallel configuration (the optimal number of workers—related to the minimum number of workers that allow to achieve the lowest completion time observed—is reported in square brackets close to node names), the outer one refers to the centralized (single worker) execution. The difference between the outer and the inner diameter shows the gain achieved with data parallelism. Circles are scattered according to the number of samples of the related training sets (x-axis) and classes that must be told apart (y-axis).

From this plot, we infer that a higher complexity corresponds to a higher reduction in training time when data parallelism is leveraged. This positive effect is balanced by the unavoidable saturation due to the synchronization bottleneck between master-workers.

Pure Model Parallelism.

In this campaign we evaluate, in the case of pure model parallelism (i.e. $N_w = 1$, resulting in each bucket with one master and one slave), how different

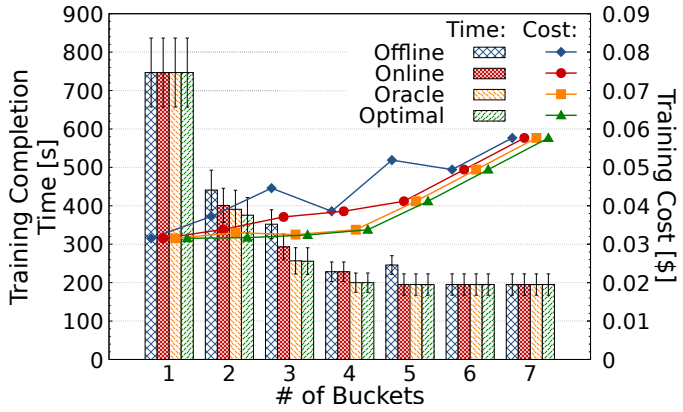


Figure 3.10: Training completion time (left y axis) and cost (right y axis) achievable with different scheduling strategies in case of pure model parallelism (bars report average and standard deviation over 10 folds, markers report the sole average).

scheduling strategies available impact the training completion time (t^{tot}) and the training cost (c^{tot}) achieved by the architecture. Notably, for this experimental analysis we only consider t^{tot} and c^{tot}, since TC effectiveness is not impacted by scheduling selection (viz. model parallelism).

In detail, we evaluate the two proposed variants of the priority scheduling, i.e. its *offline and online versions*, against *two* baselines. The former is an *oracle baseline* which realizes the same scheduling with clairvoyant knowledge of training completion time for each node $\{t_1, \ldots, t_{N_c}\}$, which is used in the place of $\{\rho_1, \ldots, \rho_{N_c}\}$. The latter is the *optimal baseline* which derives from the full exploration of all scheduling choices, namely the solution to Eq. (2.2). Figure 3.10 reports the results vs. $B = 1, \ldots, 7$ buckets. The general trend witnesses the benefits of adopting pure model parallelism, which allows to improve the average performance from a t^{tot} perspective. For instance, when considering $B = 5$ with online scheduler, there is a relative t^{tot} improvement of up to $\approx +74\%$ with respect to $B = 1$ (i.e. the centralized case), at expenses of a slight c^{tot} increase (i.e. $\approx +33\%$). Moreover, the proposed priority scheduling strategies achieve results not far from the oracle and the optimal baselines, i.e. $\approx -27.1\%$ and $\approx -27.3\%$ relative loss of training time, respectively, at most. In detail, the online variant outperforms the offline counterpart in most of the cases, i.e. $\approx +26\%$ relative gain in terms of training time, at most, and $\approx -20\%$ relative loss of training cost, at most. This gain

is due to the side knowledge of nodes completion time originating from the online feedback.

Finally, regardless of the scheduling strategy adopted, the obtained results show a *saturation*, with completion time settling at $\approx$ 200s for deployments with $B \geq 4$. This highlights the inherent limitations of pure model parallelism that need to be overcome by leveraging data and model parallelism *together*, as shown in the following analysis. More specifically, we focus on the *online variant* of the proposed priority scheduling since it guarantees better performance and feasible implementation.

Data+Model Double-Parallelism.

Based on the outcomes of previous experimentations, we now evaluate the performance of our framework. In detail, we *jointly* consider data ($N_w > 1$ workers) and model ($B > 1$ buckets) parallelism.

Accordingly, to measure performance we consider the training completion time and the resulting cost: TC effectiveness is not considered since the effect of data parallelism on RF has been shown to be negligible. Indeed, increasing B only affects the degree of model parallelism (i.e. there is a higher number of buckets which can process the training tasks associated to the classifier nodes) and does not alter TC performance of HC approaches. Conversely, increasing N_w incurs in the same TC performance insensitivity observed in Fig. 3.9c.

The above two metrics are explored by varying the number of (a) available buckets B and (b) workers per bucket N_w. For completeness, in Fig. 3.11 we again report, in dotted horizontal and dashed vertical boxes, the configurations pertaining to *pure data parallelism* ($N_w > 1$ and $B = 1$) and *pure model parallelism* ($N_w = 1$ and $B > 1$), respectively. Notably, the intersection of the two boxes reports also the time-cost performance of the *centralized HC approach* ($N_w = 1$ and $B = 1$). It is worth to underline that time-cost performance was not previously addressed, given the focus on TC effectiveness and the lack of a BD infrastructure.

By looking at the results, the minimum training completion time (Fig. 3.11 top) is obtained with $N_w = 11$ workers and $B = 5$ buckets ($\boxtimes$ symbol), corresponding to $74.54 \pm 13.44\,s$, on average. This configuration leads to $+77.58\%$, $+61.79\%$, and $+90.02\%$ relative reduction of the training completion time with respect to optimized pure data parallelism (with $N_w = 11$) and pure model parallelism (with $B = 5$), and centralized approaches, respectively. Differently, considering the cost (Fig. 3.11 middle) the cheapest configuration is obtained with $N_w = 2$ and $B = 2$ (\$ symbol), corresponding to a cost

of $0.028\$$. This configuration leads to $+24.42\%$, $+31.02\%$, $+9.9\%$ relative cost reduction w.r.t. optimized pure data parallelism (with $N_w = 2$) and pure model parallelism (with $B = 2$), and centralized approaches, respectively.

Finally, we also consider a *time-cost score* ($s(t^{\text{tot}}, c^{\text{tot}})$) taking into account both training completion time (t^{tot}) and cost (c^{tot}). In detail, we evaluate the weighted average of the two normalized metrics which ranges from 0 to 1 (the lower, the better). More specifically, the formula is:

$$s(t^{\text{tot}}, c^{\text{tot}}) = \lambda_1 \widetilde{t}^{\,\text{tot}} + \lambda_2 \widetilde{c}^{\,\text{tot}} \tag{3.1}$$

where $\widetilde{t}^{\,\text{tot}}$ and $\widetilde{c}^{\,\text{tot}}$ represent the corresponding min-max normalized counterparts (i.e. $\widetilde{x} \triangleq \frac{x - x_{\min}}{x_{\max} - x_{\min}}$), with weights $\lambda_1 = \lambda_2 = 0.5$. It is worth remarking that these weights could be configured to get the desired trade-off.

Thus, the best balanced configuration is attained with $N_w = 6$ workers and $B = 5$ buckets (⚖ symbol) and shows a training completion time of $82.81 \pm 16.89\, s$ and a cost of $0.047\$$, corresponding to a score of 0.05. This configuration leads to $+76.89\%$, $+58.78\%$, $+90.28\%$ relative reduction considering pure data parallelism (with $N_w = 6$), pure model parallelism (with $B = 5$) and centralized approaches, respectively.

Summarizing, in the light of the results reported and by comparison with existing literature, some important *take-home messages* can be drawn. First, our validation on the sole data parallelism highlights the need for a complete investigation which includes the related cost analysis. Indeed, we have found that a naïve increase of the number of workers incurs in a saturation of the time gain (due to synchronization overhead) which negatively impacts the cost of the training architecture. Such complementary and close-to-deployment investigation has been previously addressed only by our recent work [56] which was limited to a nonhierarchical scenario. Secondly, from a time-cost perspective (even) pure model parallelism is beneficial for hierarchical approaches to TC: nonetheless, this requires the careful design of a *feasible* scheduling strategy for its effective capitalization. We believe the importance of this aspect is likely to increase due to the rising trend toward large-scale TC problems [14]. Hence, our proposal complements the lack of this analysis in similar studies on hierarchical classification [45, 51]. Finally, results concerning the integration of both the model (via the designed scheduler) and data parallelism granted by our framework have highlighted the high-performance (in terms of cost, time and TC effectiveness) achieved, as well as its flexibility, in comparison to existing hierarchical TC implementations (e.g., [45]).

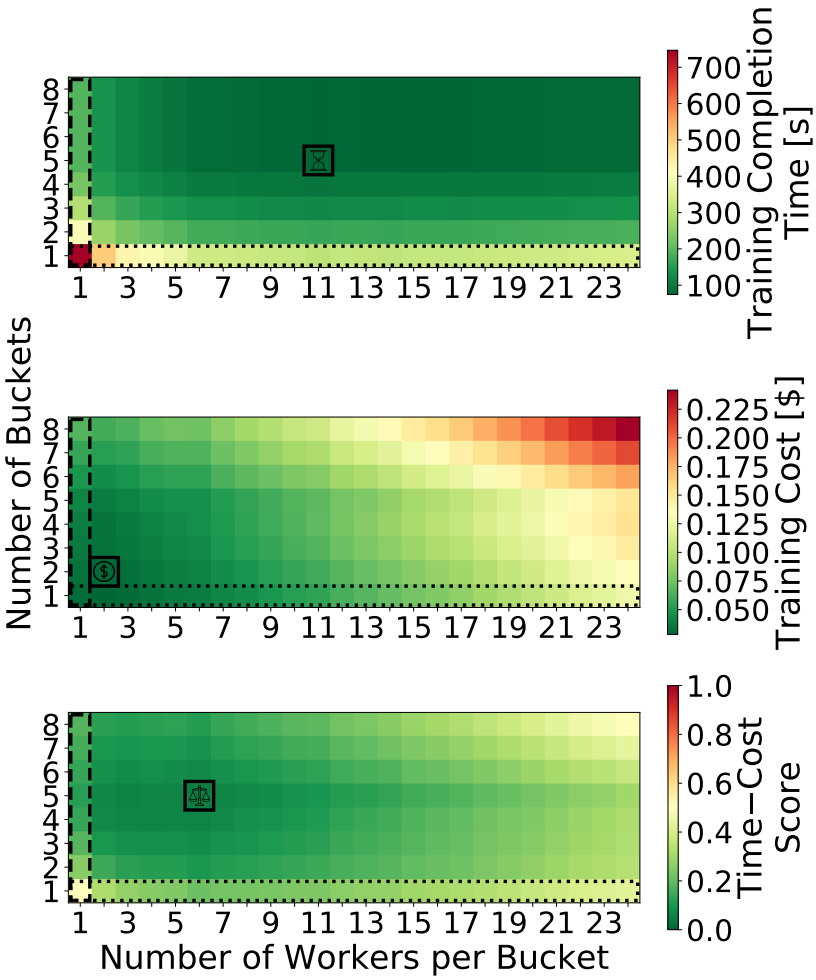


Figure 3.11: *Completion time* (top), *cost* (center), and *time-cost score* (bottom) of training task, varying the number of buckets and worker machines. Results for *pure data parallelism* and *pure model parallelism* correspond to row 1 and column 1, respectively; *centralized hierarchical classification* corresponds to cell (1,1). For each metric the optimum combination is marked.

Chapter 4

Security: Intrusion Detection for IoT devices

In this chapter the proposed framework is configured in order to tackle the Intrusion Detection task in Internet of Things (IoT) context by using the LCPN model.

4.1 Context of Intrusion Detection for IoT Devices

Nowadays, the number of Internet of Things (IoT) devices was anticipated to be over 7 billion in 2018, with a 3-fold increase expected by 2025, taking into account both consumer and industrial uses [1]. Unfortunately, IoT devices are frequently characterized by a low-cost manufacturing process (including hardware and software design decisions, such as unsecured network services, dangerous update mechanisms, and obsolete components) and insufficient user attention to setup, raising serious security issues and multiple vulnerabilities [2]. As a result of many and major flaws in IoT devices, the primary goal of daily-released malware has changed to infecting IoT services. Once infiltrated, IoT devices may be used to create a botnet that takes advantage of their "massive" and "always-on" nature to exploit newly exploitable flaws. Despite this, IoT botnets vary from traditional botnets due to the large number of bots engaged and their variety. These cyberweapons are commonly used to launch Distributed Denial of Service (DDoS) attacks, but they might also be used

[1] https://tinyurl.com/iot-dev-2018

[2] https://owasp.org/www-project-internet-of-things/

for social engineering, collecting sensitive information from targets, and even spreading malware to do unlawful cryptocurrency mining. Indeed, a number of IoT botnets have recently been documented, including the well-known Mirai malware [106] and its numerous variants [107], proof-of-concept worms designed to infect high-density IoT devices, such as smartbulbs [108], or other notable IoT malware instances like Bashlite, Hajime, BrickerBot, NewAidra, and VPNFilter. It is worth noting that hostile actions (such as DDoS and phishing) always include a reconnaissance phase (scan) and a data exfiltration phase (steal), both of which are likely to leave a trace in network traffic [4, 106, 109, 110]. Accordingly, Network Intrusion Detection Systems (NIDSs) are meant to monitor network traffic to determine when a system is being targeted by a network attack, or is a source of it.

Thereby, the interest of research community has been recently focused on attacks related to IoT context. To fulfill their goals, a NIDS could implement two main approaches: Anomaly Detection (AD) or Misuse Detection (MD), aiming at capturing any deviation from the profiles of normal activities, or identifying the patterns of known attacks, respectively. Indeed, Machine Learning-based intrusion detection has been widely adopted in last years, with researches investigating both AD [40, 49, 52], trained only on benign traffic (i.e. anomalies are identified as outliers), and MD [41, 43, 50], trained on both benign and malicious traffic [10]. MD could be binary (benign vs. malicious) or multi-class (benign vs. specific attacks). Further, several approaches fall within Attack Classification (AC), where a preliminary phase, skimming benign events, is assumed. Also, a number of proposals for network intrusion detection in IoT environments can be found in literature [71, 72, 73, 74, 75]. Differently from the scenario analyzed in this section, IoT-aware Intrusion Detection Systems (IDSs) in literature do not all use an IoT dataset for validation, or target AD or MD separately. Also, the most related proposal [73] discriminates only among known attacks (i.e. no ability to detect unknown attacks).

In this section, we leverage the proposed framework to instantiate a hybrid approach for ID, which is evaluated targeting both AD and MD at the same time: the Hierarchical Classification Framework instance, which is subsequently described, results in a Hierarchical Hybrid Intrusion Detector (H2ID)—which is an LCPN-based solution—tailored for the demands of IoT scenarios.

As outlined in Chapt. 1, several works regarding *hierarchical network Intrusion Detection (ID) approaches* where proposed, distinguishing them between AD and MD based on the nature of the first level of the hierarchy. To

better frame the problem on the specific scenario, subsequently several works performing network ID in IoT environments are presented.

In [71] authors proposed MSML, a Multilevel Semi-supervised Machine Learning-based ID. The aim of this framework is to separate unknown traffic from benign and known attacks related and it is composed by four modules, i.e. pure clustering, pattern modeling, fine-grained classification, and model updating. MSML is tested on KDD'99 dataset, showing an accuracy up to 96.6%.

Li et al. [72] propose a two-stage IDS enhanced by software-defined IoT. They perform a preliminary feature selection stage, followed by a cost-sensitive classification using a Random Forest (RF). The entire workflow guarantees the detection of novel intrusions and adds a self-learning capability. The proposal is tested on KDD'99 dataset, showing an improved detection with no impact on time complexity w.r.t. existing solutions.

Lately, in [73] authors propose a two-level ID, where the first level implements a flow-based binary MD task, whereas the second addresses packet-based Attack Classification (AC). The model is tested on both CICIDS2017 and UNSW-NB15, resulting in an ideal F1 score on the former, and 100% at the first level (resp. 97% at second level) on the latter.

In [74], a Deep AutoEncoder (DAE) is proposed for AD and compared against some outlier detectors, achieving lower False Positive Rate (FPR) (0.007% as opposed to 0.02%) with similar True Positive Rate (TPR) on their own dataset N-BaIoT. Along the same lines, in [75] authors develop Kitsune, an online neural network-based NIDS, using an ensemble of AutoEncoders (AEs) (KitNET). They show that the proposed approach performs equally as offline algorithms. They show that proposed approach balances offline algorithms. Several works leveraged it to evaluate their ID models. Then, several works have leveraged N-BaIoT; for example, [111] have compared several classifiers, i.e. ZeroR and OneR, as baseline, and JRip, PART, J48, and RF, in an AC task. Results show a $\approx 80\%$ ($\approx 10\%$) gain in accuracy of all these against ZeroR (OneR).

Finally, [112] propose a Denial of Service (DoS) detection framework composed by three modules, i.e. Dataset Preparation and Labelling, Feature Selection, and Classification & Comparison. Proposal falls in multi-class MD fitted with and without a moving average windows method. Classifier authors compare are Naïve Bayes, and evolution, i.e. Bayesian Network, A1DE, and A2DE, and other models, obtaining a near ideal performance.

Another dataset containing traffic related to IDS in the IoT ecosystem is

Bot-IoT [113], being the object of the study presented in this section. Bot-Iot was capitalized by the same authors to perform both binary and multi-class MD analyses leveraging SVM, RNN, and LSTM-RNN, resulting in a high accuracy. Moreover, the first application of AD techniques to the dataset Bot-IoT is performed by [114] through a novel system named Mixture Localization-based Outliers (MLO). MLO is fed with the top-10 features of both UNSW-NB15 and Bot-IoT datasets (unfortunately, in the latter case, these features comprise the Argus sequence number, possibly implying a performance bias). Authors compare their proposal against others AD techniques, with a $94.8\% - 98.3\%$ average TPR on Bot-IoT (resp. $95\% - 99.5\%$ on UNSW-NB15).

4.2 Hierarchical Framework Instances for Intrusion Detection for IoT Devices

In this section we describe the instance of the Hierarchical Learning Framework designed to perform the Intrusion Detection task, resorting in an LCPN-based approach, whose structure is depicted in Fig. 4.1: first, the overall LCPN architecture is shown and then the *two main levels* composing it are detailed, via the respective design choices. It is worth to underline that this LCPN solution requires a $T = 2$ levels hierarchy of labels, with L_0 being the root, L_1 being the "Type of Traffic" (viz. *benign* or *malicious*) level, and L_2 being the "Type of Attack". Accordingly, the first level of the solution corresponds to an **Anomaly Detector** (L_1), whose output may activate an **Attack Classifier** implemented as the second level (L_2). Indeed, if a TO is flagged as anomalous (ANM) by L_1, deviations from normal activities are identified and are further inspected by L_2 in order to search for known attacks. L_2 is in charge of classifying the TO according to a set of known attacks $\{\mathrm{ATK}_1, \ldots, \mathrm{ATK}_n\}$ or detect an unknown attack (UNK) possibly modeling *zero-day* attacks. This task is based on open-set classification methodology (i.e. unknown attacks are not assumed to be seen during the training phase). Differently, no other computation is required if L_1 declares the TO as benign (BNG). Notably, the specific instance of the LCPN classifier benefits from the Reject Option (Sec. 2.3.1) to enforce *a double-censoring mechanism* based on two independent thresholds γ_{B} and γ_{U} that allow the architecture to be adapted according to the required performance trade-off. Looking at the architecture in its entirety, L_1 is meant to provide a pre-filtering (aimed at identifying benign TOs) that can be conducted with low overhead also on limited hardware, thus *being suitable for IoT contexts*. Accordingly, L_2 is activated *on-request* (based on the verdict of L_1): this

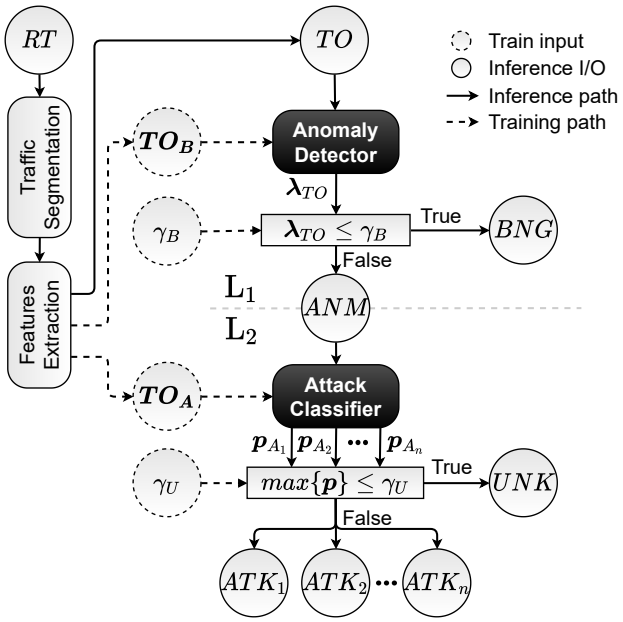


Figure 4.1: Instance of the Hierarchical Classification Framework for IDS. Solid arrows represent operational dataflow (inferring), while dashed arrows refer to training phase only.
Legend. Raw trace (RT), traffic object (TO), benign (B, BNG), anomaly (ANM), attack (A, ATK), thresholds (γ_{B}, γ_{U}), loss (λ), and probability (p).

results in a lightweight path for benign traffic that is not subjected to a second-level analysis. While the LCPN classifier is generalizable (i.e. different choices can be made to implement either level), in what follows our specific instance is detailed, consisting of a *MultiModal Deep AutoEncoder* (Multi Modal-Deep AutoEncoder (M2-DAE)) and an *Machine Learning-based classifier* at levels L_1 and L_2, respectively.

4.2.1 Dataset Description

Results have been obtained leveraging the Bot-IoT dataset, collected by authors of [113] in an emulated IoT environment, which is (when experiments where performed) the sole dataset releasing PCAP traces of (IoT) attacks and benign behaviors, enabling the extraction of engineered sets of features. Attack traffic refers to four attack categories, namely (i) *information gathering* (Scan), *denial of service*, both (ii) *single sourced* (DoS) and (iii) *distributed*

(DDoS), and (iv) *information theft* (Theft). For our study, we took advantage of the traces (in *pcap* format) composing the full dataset to extract specific features. Each sample in the dataset is labeled at multiple levels: (a) attack, (b) category, and (c) subcategory. For each sample in the dataset, we consider only the first two levels. The first one (binary-labelled, i.e. tells whether the sample is part of either a benign communication or an attack) is used for training and evaluating the first detection level (L_1), whereas the second is used to perform attack classification at the second level (L_2). In detail, L_1 is trained only with benign traffic traces, whereas L_2 is trained via traffic exclusively related to known attacks. These two training phases are completely independent (L_1 and L_2 are trained on nonoverlapping sets of data).

4.2.2 Traffic Object and Features

When performing the analysis, raw traces are first segmented in *biflows* by the Traffic Segmentation Component. In addition to TCP and UDP, also ARP is considered, due to its importance in LANs (i.e. MAC addresses to index this traffic). In detail, MAC addresses can be used to extend the 5-tuple identifying each biflows—obtaining a *7-tuple*. This traffic object allows to catch mismatches between layer-2 and layer-3 addresses, enabling the framework to naturally manage other kinds of anomalous (e.g. spoofed) traffic. However, since in the considered dataset the is a perfect match between 5-tuples and 7-tuples, this specification could be omitted without prejudice for the generality of the approach or the correctness of the reported results. For each biflow, an increasing number of packets (i.e. 5, 10, 15, 20, and 25) is considered, thus enabling the evaluation of the proposal in several scenarios (including *early-detection*, in which the very initial sequence of packets is taken into account). From the raw traces the Feature Extraction Component extracts the input fields reported in Tab. 4.1. They are differentiated according to the layer of the TCP/IP stack they belong to (i.e. network or transport), their type (i.e. numerical or categorical), and granularity (i.e. TO- or packet-based). To reduce the number of malicious biflows and obtain a balanced dataset, a random undersampling of extracted TOs is enforced by selecting all biflows related to benign traffic (i.e. $\approx$ 7k) and at most 500 biflows (selected at random) per attack subcategory, for a total of 4.5k anomalous samples.

Table 4.1: Input data extracted from Bot-IoT dataset.

TCP/IP Stack Layer	Field Name	Stat(s)	Type	Granularity	#
Transport	`n. wrong fragments`	+	N	B	3
	`destination port`	n.d.	C	B	1
	`payload bytes seq.`	A	C	P	3
	`payload length`	A, S, m, M	N	P	12
	`TCP flags combin.`	+	C	P	3
	`TCP window size`	A, S, m, M	N	P	12
Network	`n. packets`	+	N	B	3
	`byte rate`	n.d.	N	B	3
	`duration`	n.d.	N	B	3
	`inter-arrival-time`	A, S, m, M	N	P	12
	`protocol`	n.d.	C	B	1
	`time-to-live`	A, S, m, M	N	P	12

Legend:
Input Dimensionality (#);
Stats: sum (+), average (A), std dev (S), minimum (m), maximum (M);
Type: Numerical (N), Categorical (C);
Granularity: Biflow-based (B), and Packet-based (P).
Fields present bidirectional, upstream, and downstream representation.
`destination port` and `protocol` are only bidirectional.

4.2.3 Models

L_1: Anomaly Detection through M2-DAE

In this section, the proposed approach for AD is described, leveraging a particular class of Deep Learning models, namely the DAEs, in an innovative way. In this context, the DAE is used as an anomaly detector, i.e. by using (during testing phase) the loss metric between the observed input and the DAE-based reconstruction as a measure of "anomaly-ness" [89]. The reason for this choice is the successful application of AEs (and recently DAEs) in several AD works [42, 89, 115]. The *autoencoder*-based solutions has been designed as the model for L_1 is an Multi Modal-DAE (M2-DAE) neural network (detailed in Sec. 2.3.4), which handles different input types of the TO as separate "modalities", thus reducing the number of trainable parameters of the network.

L_2: Machine Learning-based Attack Classification

In this section, the design of the L_2 level is discussed. Its goal is to classify each anomalous TO received from the L_1 level, associating it to either one of the known attacks $\{ATK_1, \ldots, ATK_n\}$ or the *unknown attack class* (UNK). The generality of the proposed hierarchical framework in its LCPN mode allows for any n-class classifier to be employed at the L_2 level; hence, any Machine/Deep Learning-based supervised classifier can be adopted. The sole requirement for each classifier is to be able to provide its soft-output vector $\mathbf{p} = [p_1, p_2, \ldots, p_n]$, p_i being the confidence probability associated to the (known) class ATK_i, needed by the censoring mechanism we adopt (see the following section). Given the presence of the unknown class, L_2 naturally fits the problem of *open-set classification* via reject option: during the operational phase, the TOs declared as anomalous may reveal attacks never observed in the training phase, when not matching any known attack. This choice also supports loop-based mechanisms [40] that can adaptively identify new classes for initially unforeseen attacks. Based on the existing literature on AC, several options for implementing the L_2 level are considered. In particular, three different options are investigated: RF, NB, and MultiLayer Perceptron (MLP).

Double-censoring mechanism

In the abovementioned design *both levels* implement a *threshold-based mechanism*, whose effect is to censor the output of each level in case of low-confidence. This provides the architecture with adaptability, with minimal impact on complexity. With regards to L_1, an anomaly is detected if $\lambda > \gamma_B$ (in such a case the TO is passed to L_2). Herein the threshold γ_B is designed to balance the **anomaly TPR-FPR tradeoff**. Differently, with reference to L_2, the soft-output vector $\boldsymbol{p} = [p_1, p_2, \ldots, p_n]$ of the L_2 classifier (gathering the predicted class probabilities of $ATK_1, \ldots, ATK_n$) is used to label the TO with the attack $a \triangleq \arg\max\{\boldsymbol{p}\}$, *only* when $\max\{\boldsymbol{p}\} > \gamma_U$. Differently, when $\max\{\boldsymbol{p}\} \leq \gamma_U$, the TO is associated to the unknown class UNK. Hence, γ_U represents the threshold balancing **discrimination of known attacks against unknown attack detection**. Notably, the two thresholds are *independently* set, allowing to separately tune the sensitivity to anomalies and unknown attacks. Ultimately, this allows to adhere to different performance trade-off requirements.

Deployment Scenarios

The proposed two-level architecture perfectly fits typical IoT deployment scenarios. In fact, it results in practical benefits achieved by means of deployment modularity and flexibility, as discussed hereinafter. Since IoT infrastructures are intrinsically hierarchical (e.g., made of devices, gateways, and edge/cloud layers), the two constitutive levels can be deployed at different layers and trained separately. The latter aspect limits the need for retraining the whole architecture. Indeed, L_1-*Anomaly Detection* can be deployed *locally*, i.e. on each IoT device (or on the local gateways). On the other hand, L_2-*Attack Classification* can be implemented in a centralized fashion (e.g., at a remote gateway or at the edge/cloud layer). As IoT devices are often characterized by limited computing, storage, memory, and energy resources, it is paramount that the locally-deployed functionalities do not conflict with these constraints. In our proposal this is guaranteed by the lightweight nature of the L_1 level (implementing M2-DAE as opposed to common DAEs). Moreover, for both the training and operation phases, the proposed framework architecture adheres to the privacy requirements characterizing IoT networks, where operational (benign) traffic is naturally subjected to privacy concerns (e.g., smart-home or health-related applications, to name a few). More specifically, the proposed two-level architecture allows both *training and operation phases* to be performed without sharing the benign traffic with the remote nodes implementing AC. Indeed, the AD level is the only component to be fed with the benign traffic and is locally deployed. Only the traffic marked as anomalous is sent to the (remote) AC level. Accordingly, no exchange of benign privacy-concerned traffic is put in practice neither in training nor in operation. This notwithstanding, the training process is not negatively impacted since: i) L_1-*Anomaly Detection*—requiring benign traffic only—is still able to learn patterns based on local observations; ii) L_2-*Attack Classification* is based on anomalous traffic only, and does not need the—privacy concerning—local traffic for training. Thus, L_1 can be pre-trained by using already known traffic patterns of IoT devices connected to the user's network and fine-tuned to the specific traffic behavior of each locally-connected device. Since the IoT-generated traffic patterns could be influenced by the deployment position (e.g., distance from the cloud/edge) and by user behavior and needs, the fine-tuning phase should take this heterogeneity into account by balancing the importance of both contributions (i.e. location- and behavior-dependent features). Instead, L_2 can be more effectively trained by using samples observed from multiple distributed devices. This is in line with the fact that the benign traffic is expected to be

characteristic of the specific IoT devices (and of their position in the network) while the attacks are expected to assume their own typical patterns which are dictated by the attacking strategies rather than the attacked network. Accordingly, the designed architecture naturally allows IoT devices to concur in training L_2-*Attack Classification* by exchanging traffic at any granularity via a wide class of existing federated learning approaches [116, 117], without incurring in loss of privacy.

4.3 Experimental Results

In this section, the experimental evaluation of the Hierarchical Classification Framework instance for IDS is provided. First, evaluation metrics are introduced and then the evaluation setup is described. Finally, experimental results are shown and discussed.

Performed performance assessment consists of *two* phases: L_1 **analysis** and **IDS analysis**. In L_1 **analysis** the ROC analysis is exploited to evaluate M2-DAE against three One-Class Classifiers (OCCs) commonly used in AD (i.e. One-class Support Vector Machine (OC-SVM), Isolation Forest (IF), and Local Outlier Factor (LOF)), and a standard DAE, which is inspired by [74]. OCCs are fed with the same engineered input as (M2-)DAE, which consists of 4+4 encoding/decoding layers with *relu* activations. In **IDS analysis** the hierarchical model instance is compared with a multi-class MD (Multi-MD), designing both for open-set classification. It is worth to underline that the Multi-MD model is instantiated by leveraging the flat approach with reject option available in the hierarchical framework. The two approaches are compared based on the F1 score of the open-set problem ($\{\mathrm{ATK}\}_{i=1}^{n} \cup \mathrm{UNK}$) vs. the unknown threshold (γ_{U}). A former comparison of RF, NB, and MLP ability in performing AC and MD tasks aims to select the best model for L_2 and Multi-MD, respectively.

4.3.1 Results of Anomaly Detection Analysis

In Fig. 4.2, the partial ROC (i.e. with $\mathrm{TPR} \in (90, 99)\,\%$) compares the experimented models at L_1. Notably, M2-DAE and DAE outperform OCCs models, showing an almost constant FPR $\leq 1\%$ on varying number of packets, against $> 40\%$ FPR reached by OCCs. Although DAE and M2-DAE report similar results (with the former resulting in a lower FPR in most of the cases), the latter boasts a less complex model, with a reduction of trainable parameters

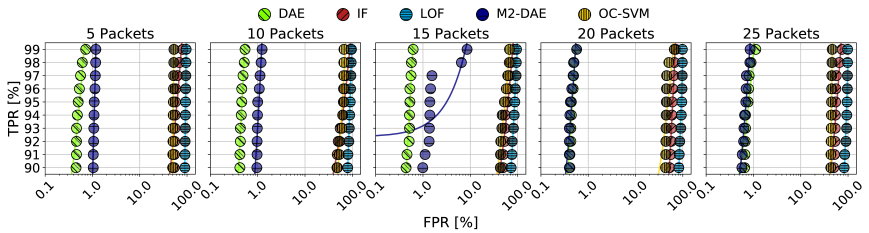


Figure 4.2: Partial ROC for baselines and proposed M2-DAE, varying the number of packets.

(i.e. weights and bias) up to a factor of $4\times$ (i.e. from $\approx 12M$ of DAE to $\approx 3M$ of M2-DAE). Therefore, it provides a better trade-off, considering that the reduction of complexity is a desired property for ML approaches in IoT context, when these algorithms are designed to run on hardware with limitations. Accordingly, the M2-DAE as model has been selected for L_1. Similarly, 5 packets per TO has been considered, to support early detection of the anomalies with low overhead.

4.3.2 Results of Intrusion Detection Approach Analysis

To select the best models for L_2 and Multi-MD, the aforementioned Machine Learning models (i.e. RF, NB, and MLP) have been preliminarily evaluated tackling AC and MD, respectively. The outcomes of the comparison are summarized in Fig. 4.3, reporting the results for MD task via RF, NB, and MLP on the overall dataset in terms of F1 score. Results pertaining to AC are omitted for brevity. Accordingly, the RF is selected a model for both MD and AC task, because it reported the best performance and the lowest variability. Then, for both the LCPN-based IDS and Multi-MD models, the open-set analysis is performed by removing one of the known attack classes from the training set and considering it as *unknown* (UNK class). This procedure is iterated for all the attack classes. It is worth to underline that DoS and DDoS classes are merged and referred as DoS. Then, a threshold γ_{U} is applied to the soft-output of the classifiers: TOs whose highest predicted class probability is $\leq \gamma_{\mathrm{U}}$ are declared as UNK.

In Fig. 4.4, the F1 score vs. γ_{U} is used to investigate their ability to recognize UNK class, for both the LCPN-based IDS and Multi-MD; for the proposed IDS, γ_{B} is fixed to obtain a $\approx 1\%$ FPR (the TPR corresponds to $\approx 99\%$ (Fig. 4.2)). Results highlight no significant difference in recognizing Theft

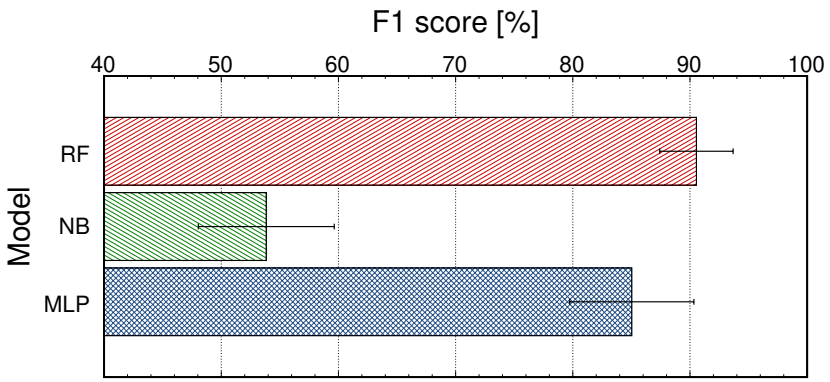


Figure 4.3: Comparison among F1 score of ML models for MD task. Variation values are provided for a C.I. of 99.7%.

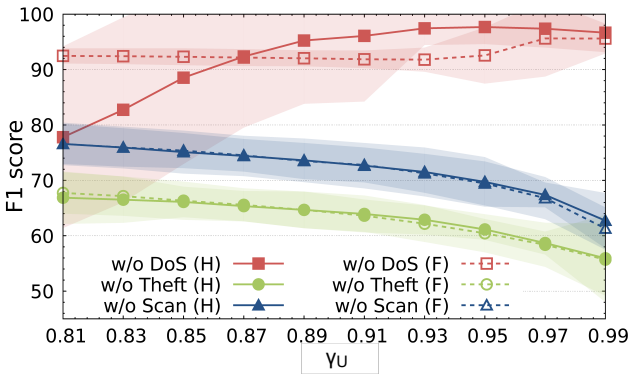
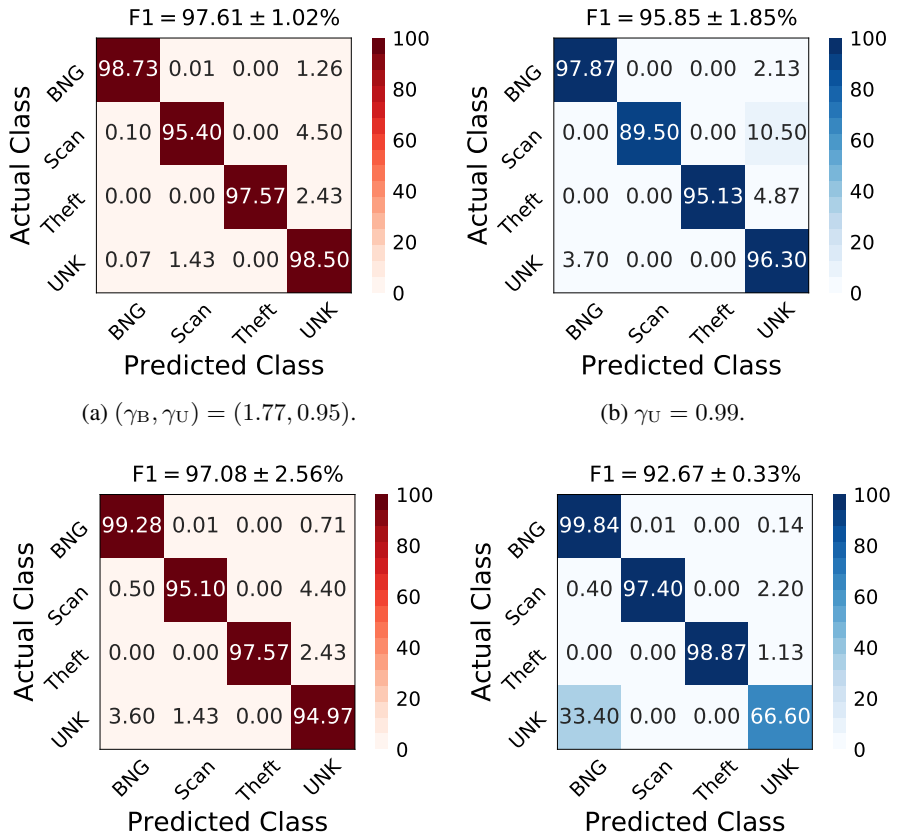


Figure 4.4: F1 score vs. γ_U for LCPN-based IDS (H) and Multi-MD (F) in an open-set approach. Variation values are provided for a C.I. of 99.7%.

and Scan as the UNK class, whereas for DoS class the proposed IDS solution can potentially provide significant gains over Multi-MD in terms of F1 score. Given the trade-off introduced by tuning γ_U for Multi-MD and (γ_B,γ_U) for the LCPN-based IDS, we inspect fine-grained classification results (w/o DoS) considering (i) the configurations achieving the best F1 score (Figs. 4.5a–4.5b), (ii) those ensuring a FPR $\leq 1\%$ (Figs. 4.5c–4.5d): the LCPN-based IDS achieves $+1.76\%$ ($+4.41\%$) F1 score in the former (latter) case. Confusion matrices confirm the effectiveness of the LCPN-based IDS in capturing unknown attacks, as opposed to Multi-MD. This recognition capacity reduces with the



(a) $(\gamma_B, \gamma_U) = (1.77, 0.95)$.

(b) $\gamma_U = 0.99$.

(c) $(\gamma_B, \gamma_U) = (2.57, 0.96)$.

(d) $\gamma_U = 0.72$.

Figure 4.5: Comparison between confusion matrices of LCPN-based IDS (left) and Multi-MD (right). (a) and (b) report the results of the configurations resulting in the best F1 score achievable. (c) and (d) report the results of the configurations resulting in the lowest FPR ($\leq 1\%$).

decreasing of FPR, but it is always better than Multi-MD. Finally, the use of a less complex classifier (no benign samples) gives to the LCPN-based IDS a gain in capturing attack classes.

Chapter 5

Traffic Management: Classification of Mobile Applications

In this chapter a wider evaluation of the framework instances is provided by using both LCPN and global classifiers solutions against the traffic classification of the traffic generated by mobile applications, to enhance fine-grained QoS enforcement via Traffic Management solutions.

5.1 Context of Classification of Mobile Applications

When no accurate information on the application generating the traffic is available, the use of security and QoS enforcement facilities, as well as network monitors, is decreased or qualitatively compromised. TC is the process of associating network traffic with certain applications, and has a long history of use in numerous domains. Nevertheless, the growing popularity of mobile devices, which is changing the type and composition of traffic traversing residential and business networks and connecting content and services over the Internet, is posing a serious challenge to TC. For instance, according to the latest Ericsson mobility report [76], between Q3 2019 and Q3 2020 (a time frame including the spread of the current pandemic situation), mobile data traffic grew 50%, driven by both the rising number of smartphone subscriptions and the increasing average data volume per subscription fueled primarily by video content. In more detail, it is forecasted that the share of *video traffic*, currently accounting

for 66% of all mobile data traffic, will increase to 77% in 2026. As a result, in recent years the popularity of mobile TC has grown, driven (in addition to the normal drivers of TC, such as service differentiation) by valuable profile data (for advertisers, security agencies, and insurance companies), while also pointing to privacy risks (e.g., recognition of context-sensitive apps, such as dating and health apps, and bring-your-own-devices policies). The increasing use of encrypted protocols has been spurred by the attempt to safeguard privacy and security (TLS).

Accordingly, several recent studies have examined mobile TC, primarily in the presence of encrypted traffic and using both standard machine and deep learning approaches. To this end, we will review the best-known studies that use basic ML and DL-based algorithms to handle encrypted traffic in order to conduct TC in the mobile context.

Among the first works tackled TC for mobile traffic, Stöber et al. [118] proposed a user fingerprinting technique for devices that learns their traffic patterns by monitoring background activity (about 70% of smartphone traffic). Data bursts are used to extract statistical information from 3G transmissions, which is then used to infer, by mean of ML-based classifiers, the individual user who generated it. In detail a 90% accuracy is achieved by considering 20 users with various groups of installed applications, focusing only on Android OS.

On the other hand, authors of [119] use a RF classifier to classify app usage across 13 iOS apps into 8 distinct categories by using data retrieved from encrypted Wi-Fi traffic. As the training time increases the results demonstrate inconsistent behavior, showing the influence of imprecise ground truth, which clearly compromises the accuracy of the classification.

Alan and Kaur [120] use website fingerprinting approaches to see if Android applications can be recognized from their launch-time traffic using only TCP/IP header information (i.e. the payload size of the first 64 packets). Applications can be recognized with 88% accuracy in the best case, i.e. when the training and test samples are collected on the same device. However, when the operating system/vendor is different, there is a significant drop, down to 26% for the top classifier. The effects of app updates on training data (viz. aging) are also considered.

Noteworthy, Aceto et al. [23] recently proposed a systematic framework for tackling the problem of encrypted (mobile) TC using DL models on three datasets of real human user activities, highlighting pitfalls, design guidelines, and challenges of works relied bot-generated datasets. Several factors are con-

sidered, including (i) the chosen TC object, (ii) the type and amount of input data, (iii) the DL architecture used, and (iv) the set of performance indicators needed for a thorough evaluation. The study of existing DL-based traffic classifiers reveals the need for unbiased, informative, and diverse input data, as well as a rigorous performance evaluation workbench.

Taylor et al. [121] present AppScanner, a machine learning (ML)-based (i.e. RF) system for smartphone app identification based on packet size and direction (also accessible from encrypted traffic). Results show app reidentification with up to 96% accuracy in the best case, exceeding baselines taken from website fingerprinting, with good tolerance to fingerprint aging, based on traffic generated by bots of the 110 most popular Android apps and accounting for variation in app versions, devices used, and fingerprint aging. To a similar extent, the same authors in [17] conducts further in-depth analysis of AppScanner (i.e. leveraging a larger dataset) to evaluate the loss of classification performance caused by the variation/aging of app fingerprints due to the different versions of the devices/apps used.

Moreover, in [122] authors present CUMMA, a technique based on RF, hidden Markov models, and clustering, with the goal of classification (and detect anomalous uses) of services in mobile messaging apps. Based on data collected from 15 participants using Whatsapp and WeChat, the results show that both apps are 96% accurate.

Finally, in [3] a multiclassification technique that leverages state-of-the-art classifiers provided for mobile (encrypted) TC was provided, by considering 4 classes of combinations that vary in the classifier results employed, learning philosophy, and training requirements. Using real user action datasets from iOS and Android, the combined results outperform the best state-of-the-art ML classifier (up to $+9.5\%$ recall).

5.2 Hierarchical Framework Instances

In this section, we describe our framework instances we evaluated in order to evaluate hierarchical TC solutions for the traffic generated by mobile applications. First, we decided to enforce hierarchical dependencies arranging the mobile applications into two main groups, namely *video* and *generic*. This first division, which defines the first two branches of the hierarchical dependency tree, is driven by the deeply different service requirements of these kinds of traffic, with the video that is huge demanding in terms of network resources. Furthermore, both video and generic branches are split into traffic categories

with a major difference: the video traffic is arranged into categories matching the *traffic type*, whereas the generic traffic is divided into *service-specific categories*. The definition of these two different categories is justified by the fact that video traffic has different behaviors with respect to the reproduced video stream. On the contrary, non-video (viz. generic) traffic usually follows the activities that users can perform, pushing for a service-specific categorization. Finally, the class dependencies tree ends with application at the leafs level.

Subsequently, enforcing of the aforementioned class dependencies tree in the training phase of designed hierarchical approaches, a broader evaluation of our framework is performed. Detailing, we compare the entire set of approaches provided by the Hierarchical Learning Engine, in order to assess (performance) advantages against a flat approach (viz. non-hierarchical). This analysis is performed by using different sets of features feeding both ML and DL state of the art models for mobile TC. Beyond the evaluation of (hierarchical) classification metrics, the analysis is also performed with respect to the calibration of the obtained models and in the presence of reject option.

5.2.1 Dataset Description

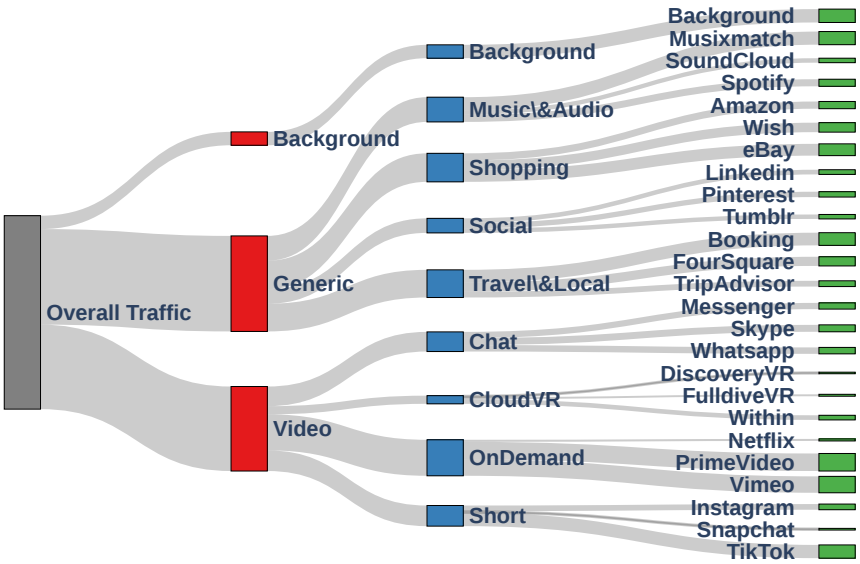
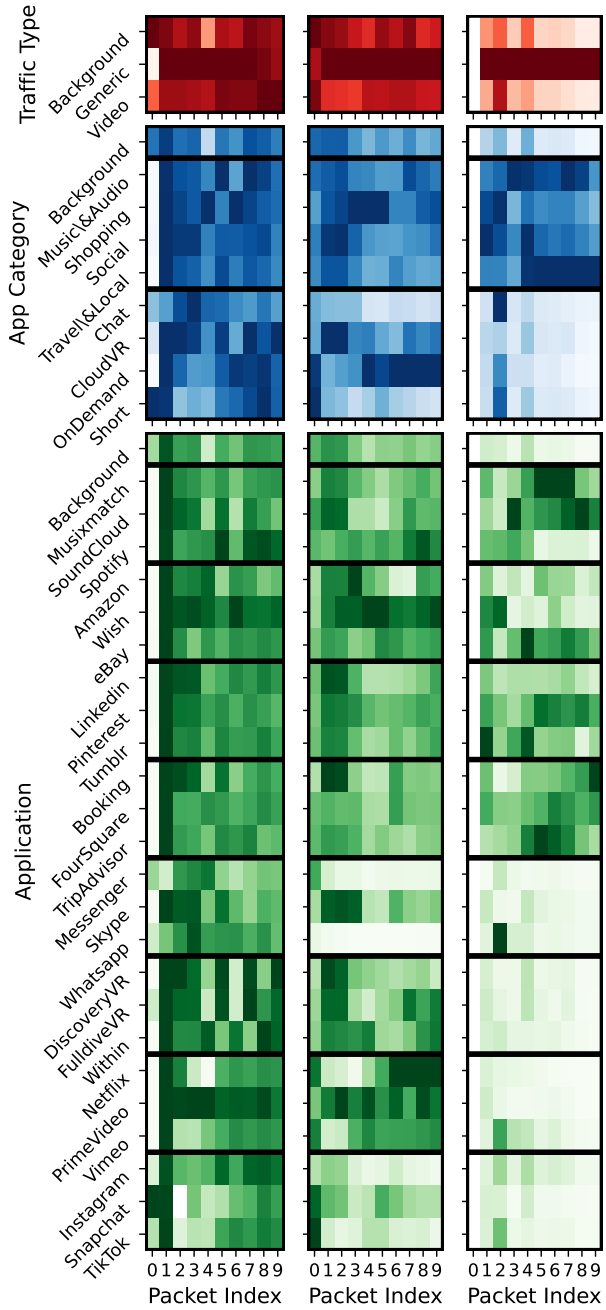


Figure 5.1: Number of biflows for the MIRAGE-2019 (Generic) and MIRAGE-video (Video and Background) datasets. The boxes' sizes are proportional to the number of biflows.

In this use case, we leverage both the datasets MIRAGE-2019 and MIRAGE-video, whose number of biflows is sketched in Fig. 5.1. Both datasets are collected leveraging the MIRAGE traffic capture system [14] at University of Napoli "Federico II". From these datasets we extract the traffic generated by 24 popular android applications, plus spurious network traffic referred as *background*, resulting in 25 classes at application granularity (in green in Fig. 5.1), 9 classes at service level (in blue in Fig. 5.1), and 3 macro classes, namely *video*, *generic*, and *background*. In detail, both not background macro classes are divided into 4 services and each non background service is further divided into 3 application classes, resulting in a balanced hierarchy we named MIRAGE-ext.

Therefore, in Fig. 5.2 is provided the hierarchical characterization of mobile applications by means of three main packet-level features, namely Direction (DIR) (where 0 stands for upstream and 1 for downstream), Payload Length (PL), and Inter-Arrival Time (IAT). In detail, Fig. 5.2 reports three

heatmaps representing the average per-level behavior of the considered classes w.r.t. abovementioned features for the first 10 packets with payload of biflows, essentially discarding signaling packets and considering, if HTTPS is used, about the first 5 packets belonging to the TLS handshake and the reminder to application data. Noteworthy, by focusing on the "Traffic Type" level (L1) clear differences arise. In particular, the generic traffic consistently shows the highest feature values for all packets (i.e. packets are usually downstream, large, and with high inter-arrival-times), but the first present an opposite behavior (i.e. the first packet carries a small payload, w.r.t. background and generic classes, and it is almost always sent in upstream). These outcomes denote the nature of generic traffic being sporadic and mainly generated by the user initiative (viz. request). Going deeper in the hierarchy, at "App Category" level (L2) we can note more patterns, with the Short video category being prone to present the first packet in downstream, and the OnDemand video showing a consistent increasing of payload size. A more confused behavior characterizes generic application categories. Finally, at "Application" level (L3) a clearer match of categories behavior to the considered applications is shown, like Netflix that strongly impacts the data payload length of the OnDemand category, and Whatsapp and Messenger that clearly push down the payload size for the Chat category.



(a) DIR (b) PL (c) IAT

Figure 5.2: Per-packet features composition of DIR, PL, and IAT.

5.2.2 Traffic Object and Features

In this section, we describe the input we extract from datasets in order to feed specific models. In particular, we extract three commonly used sets of features, namely (i) statistical, (ii) first-packets payload-bytes, and (iii) per-packet IP/L4-header-info. It is worth to underline that we select the *biflow* (bidirectional flow) as Traffic Object (TO) because capable of catch the request/response communication patterns characterizing the majority of mobile applications.

Statistical Features. This category of features is widely used in literature addressing TC when classical ML models (like DT, RF, SVM) are adopted [14]. In detail, for each TO, we extract a set of 40 features, namely (i) *Service*, (ii) *Protocol*, (iii) *Number of Packets*, (iv) *IP Packet Length*, (v) *L4 Payload Bytes*, (vi) *Duration*, (vii) *(L4) Payload Length* (PL), and $(viii)$ *inter-arrival time* (IAT), of which the time-series features PL and IAT are reported in terms of 17 stats, namely *minimum*, *maximum*, *mean*, *standard deviation*, *variance*, *mad*, *skewness*, *kurtosis*, and *percentiles* from 10 to 90 with step 10.

Payload Bytes Features. This set of features is composed by the sequence of the first bytes composing the L4 payload, for each TO [22]. In particular we extract up to 784 bytes for each biflow, encoding each byte in the range $[1\text{-}256]$ with a zero-right-padding for biflows do not cover the entire sequence. Noteworthy, when considering TCP biflows, this features will match part of the TLS handshake, in particular covering the total of both *client hello* (sized in 160-170 bytes) and *server hello* (sized in 70-75 bytes), and a part of the *certificates/keys* (size varies with negotiated parameters, e.g., the minimum recommended dimensions of 256 bytes for RSA and 32 bytes for ECDSA), up to cover the entire TLS handshake when certificates/keys are small enough, i.e. including also handshake closing info.

IP/L4 Header Time-Series Features. Finally, this group of features is composed by characteristics extracted from the IP and L4 headers of each packet (w/ L4 payload) [24]. These characteristics are the per-packet (i) *(L4) Payload Length*, (ii) *inter-arrival-time*, (iii) *Direction*, and (iv) *TCP Window Size*. The maximum number of packets is set to 100, resulting in a input of size $(100, 4)$. It is worth noting that the first IAT is always zero. A zero-right padding is applied when a biflows does not contains enough packets.

We refer to these three groups as STAT, PAY, and HDR, respectively. We recall that the modeling based on STAT input translates into a post-mortem analysis because features are computed once biflows are closed (viz. on the entire set of packets). Otherwise, PAY and HDR result in an online (viz. early)

classification, resorting to the first packets of each biflow.

5.2.3 Models

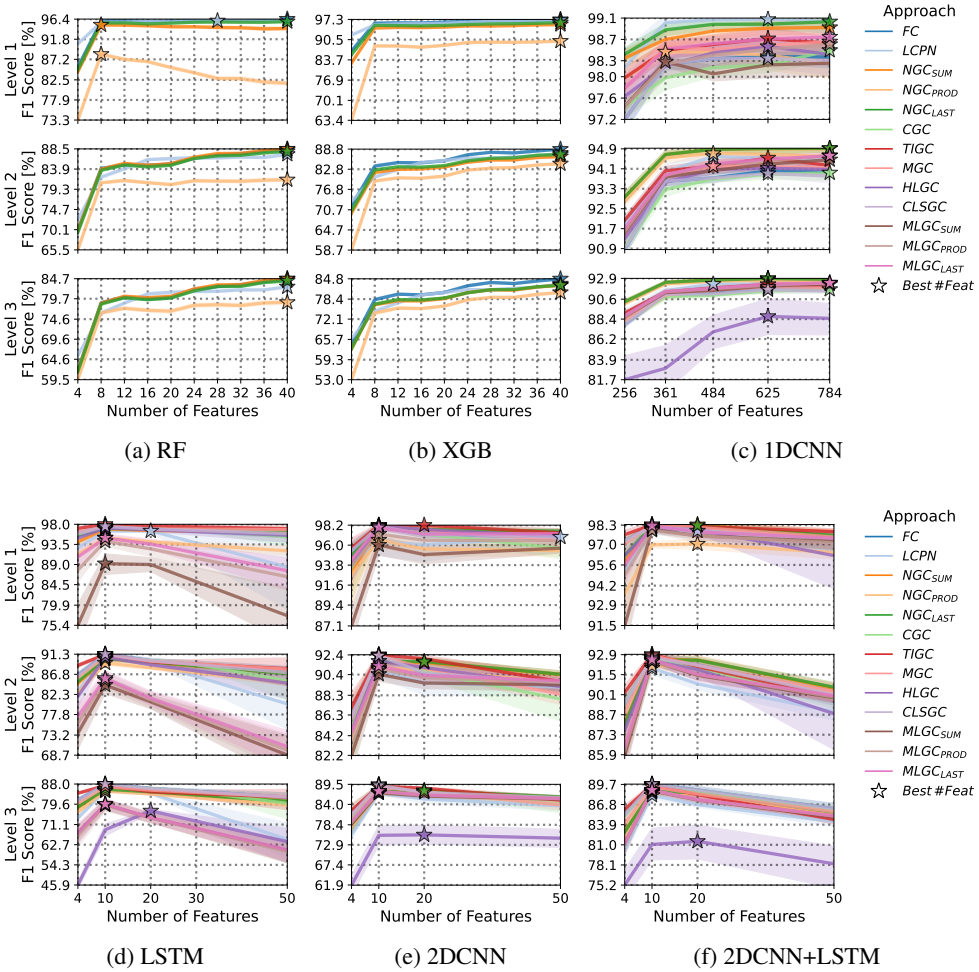
In this section, evaluated ML and DL models are listed. In detail, we select 2 classical ML models, i.e. RF [17] and eXtreme Gradient Boosting (XGB) [123], and 4 DL models, i.e. 1-Dimensional Convolutional Neural Network (1DCNN) [22], Long Short-Term Memory (LSTM) [24], 2-Dimensional Convolutional Neural Network (2DCNN) [24], 2DCNN+LSTM [24], and MultImodal DL-based MobilE TraffIc Classification (MIMETIC) [16], which are carefully described in Sec. 2.3.4. It is worth to underline that each model takes its own set of features from Sec. 5.2.2: (i) STAT is used for RF and XGB, (ii) PAY is used for 1DCNN and for the *first modality* of MIMETIC, and (iii) HDR is used for LSTM, 2DCNN, 2DCNN+LSTM, and for the *second modality* of MIMETIC.

All these models are trained by the Hierarchical Learning Engine by enforcing class dependencies through the hierarchical learning approaches described in Sec. 2.3.

5.3 Experimental Results

In this section we report results derived from the systematical application of hierarchical learning approaches for the task of TC on the dataset MIRAGE-ext. In detail, in this section we (i) assess the sensitivity of ML/DL models used to enforce hierarchical classification approaches against the selected input (viz. features selection); then (ii) we select the best input configuration for each model-hierarchical approach combination, providing a fair comparison among the best hierarchical classification approaches from the perspective of (hierarchical) classification performance, i.e. (h)F1 Score; subsequently (iii) we perform a calibration analysis via (h)ECE and (h)MCE to compare the reliability of all hierarchical approaches by selecting the best model for each aforementioned set of features; finally, (iv) we compare performance of the best performing ML/DL models and hierarchical classification approaches combinations, with respect to the flat counterpart, when the reject option is enforced. We underline that for the LCPN classification approach we resort in a naïve configuration, opting for the selection of the same model at each node, by selecting the same number of features and the same reject option thresholds for all the hierarchically arranged classifiers.

5.3.1 Sensitivity to the Number of Features



(a) RF (b) XGB (c) 1DCNN

(d) LSTM (e) 2DCNN (f) 2DCNN+LSTM

Figure 5.3: Sensitivity to number of features for single-modal models which are fed with the respective feature set. Results are reported as $avg. \pm std.$ over the 10-folds.

As abovementioned, in this section we perform the optimization of the input size (viz. number of features) for each ML and DL model, by separately considering hierarchical classification approaches. The F1 Score is reported for each level (viz. from L1 to L3) by coarsening the prediction at

the finest granularity L3, i.e. starting from leaf nodes predictions we ascend the hierarchical branches to obtain coarser predictions. It is worth noting that the MIMETIC model is not considered for the sensitivity analysis because the multimodal nature of the input causes a combinatorial explosion of the search space, opting for the per-modality optimization as shown in the paper the model is proposed [16]. Accordingly, we selected for the MIMETIC model 12 packets for the HDR set of features (viz. first modality) and 512 bytes for the PAY ones (viz. second modality).

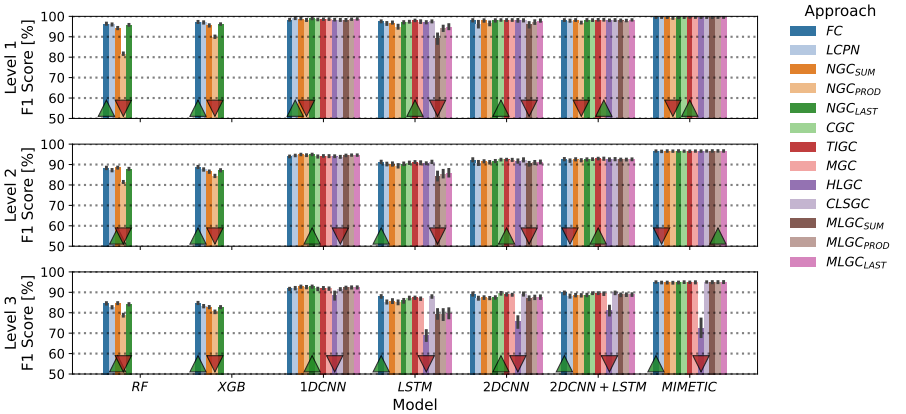
In detail, in Figs. 5.3a and 5.3b we report performance varying the number of STAT features from 4 to 40, with step 4, for RF and XGB, respectively. We recall that these features are selected based on their ranking, which is obtained via mutual information [124]. Noteworthy, the best number of features (star markers) is always 40 for both models and for all the approaches. The few exceptions regard the first level prediction of RF, where NGC_{SUM} and NGC_{PROD} saturate at 8 and the $LCPN$ at 28 features. From this analysis it is clear that the best results on the STAT features are obtained by selecting the entire set of features, i.e. 40.

Focusing on the PAY set of features, and thus to the 1DCNN model shown in Fig. 5.3c, it is clear that for the last level the optimal number of features (viz. the optimal number of payload bytes) is 625 for the majority of approaches with few exceptions falling around this value. Noteworthy, this trend is confirmed considering coarsest level of aggregation, with performance on L1 that reaches a plateau for all the approaches at 625 bytes.

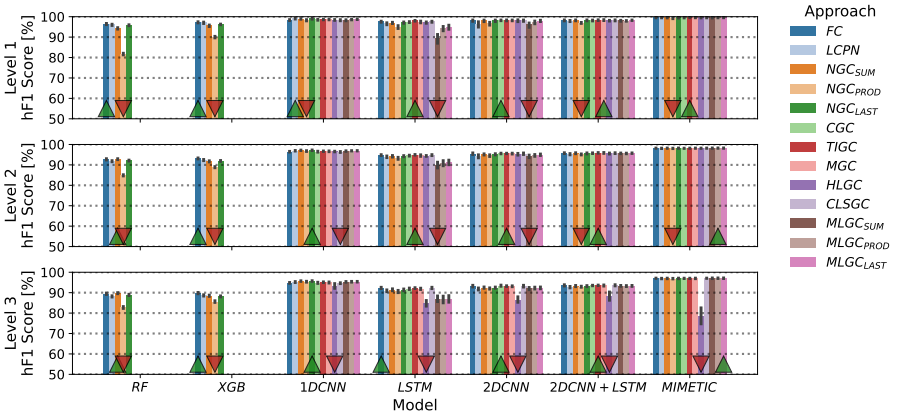
Considering the F1 score trend for the HDR set of features, namely Figs. 5.3d, 5.3e, and 5.3f, we remark that 10 packets (viz. features) are enough to obtain the best performance. In addition, it is worth noting the negative effect on performance caused by the selection of more than 10 packets. Exceptionally, the $HLGC$ is the sole approach consistently showing the best performance at 20 packets for the three models, namely LSTM, 2DCNN, and 2DCNN+LSTM. Ascending the hierarchy, 10 packets is consistently selected as the best number of features across all approach/model combinations.

5.3.2 Comparing Best Configurations

In Fig. 5.4 the comparison of the (hierarchical) F1 score for each combinations model/approach we validated is shown by selecting the best number of features for each configuration. The figures show, for each model, the best (i.e. the green triangle pointing up) and the worst (i.e. the red triangle pointing down) approach, in terms of F1 score and hierarchical F1 score. Noteworthy, looking



(a) F1 Score



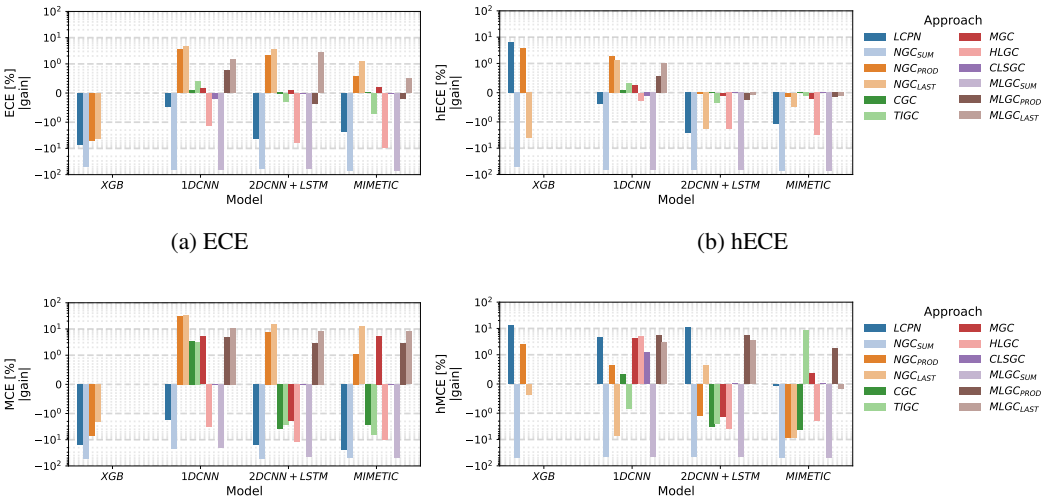
(b) hF1 Score

Figure 5.4: Comparison of classification performance, i.e. F1 (a) and hF1 (b) Scores, by selecting the optimal number of features for each model-approach combination, using MIRAGE-ext dataset. Fixed the model, the facing up (down) green (red) triangle indicates the best (worst) approach. Results are reported as $avg. \pm std.$ over the 10-folds.

at Fig. 5.4a by considering the finest granularity (L3), the approaches which are selected as worst are the $HLGC$ for DL-based models and the NGC_{PROD} for the ML-based ones. On the other hand, the best approach is the FC, which shows the highest F1 score for 4 out of 7 models, followed by the NGC and the CGC. Despite these findings elect the *flat classification* as the best per-

forming, going up through the hierarchy, the results show a different story. In fact, at both app category and traffic type classification (viz. L2 and L1, respectively) the *global classifiers* take the throne, showing better performance for 5 out of 7 models at both L2 and L1. As a general comment, the $MLGC$ approach is hard to train when using the $LSTM$ model, and the product-based post-processing aggregation applied to the NGC (viz. NGC_{PROD}) is always worse than the sum-based one (viz. NGC_{SUM}). Moreover, in Fig. 5.4b the hierarchical F1 score is shown. In this case, the FC is no more the best approach for the application level classification (L3), leaving the floor for solutions based on *global classifiers* which show higher performance for 5 out of 7 models. In detail, the hierarchical version of the F1 score metric is more complete with respect to the classical one because it considers the classification behavior through the entire hierarchy of labels, catching the behavior at coarser granularities, i.e. when non-mandatory leaf prediction is enforced.

5.3.3 Reliability Analysis



(a) ECE (b) hECE

(c) MCE (d) hMCE

Figure 5.5: Drop from the FC technique in terms of ECE (hECE) and MCE (hMCE).

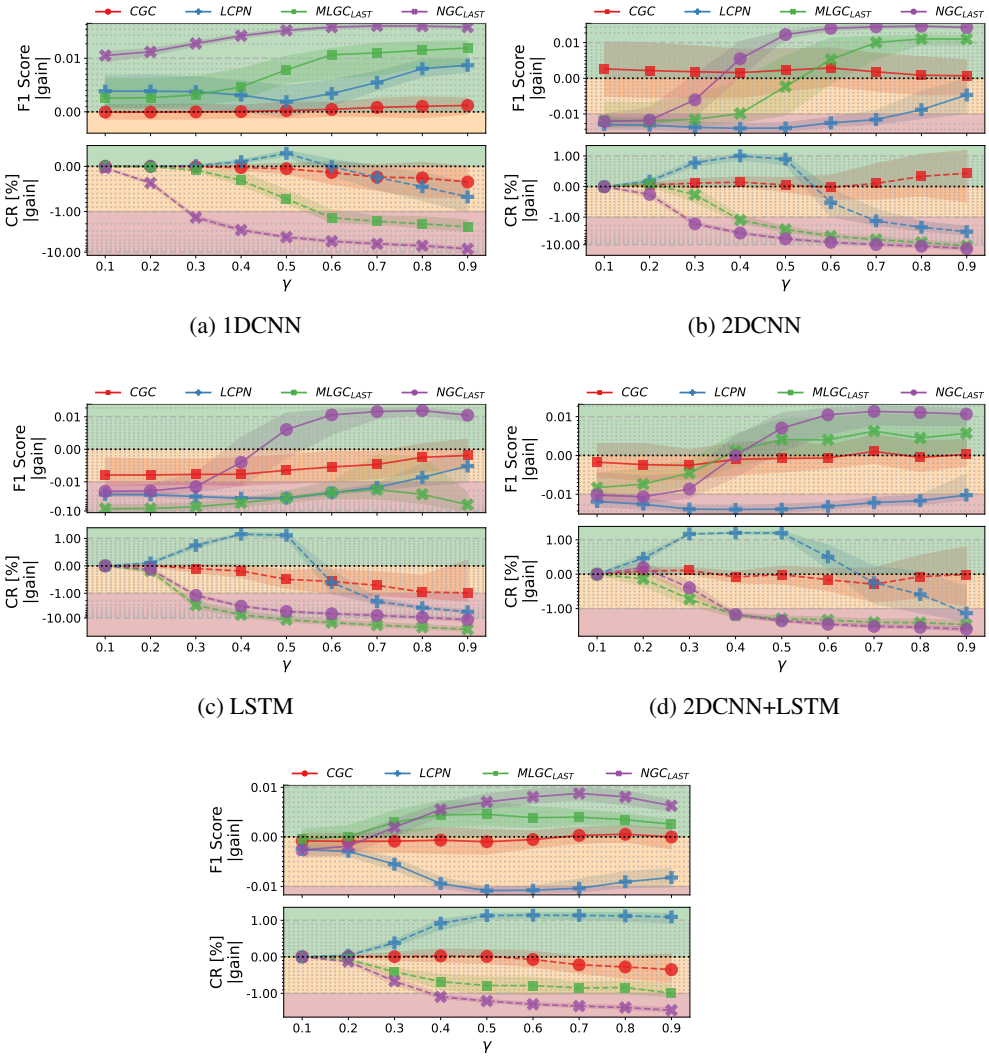
In this section we present the model calibration analysis by comparing the *absolute gain* from the FC approach of reliability metrics introduced in

Sec. 2.3.5, namely (hierarhical) ECE and MCE, (Fig. 5.5) for all the evaluated hierarchical learning solutions. For sake of brevity and uniformity of results among the three defined feature sets, we report calibration diagrams for the best model for each set of features, showing XGB for the $STAT$, the $1DCNN$ for the PAY, the $2DCNN + LSTM$ for the HDR, and $MIMETIC$ because of the multimodal input (viz. PAY+HDR). We underline that the metrics are shown for the finest level of classification (viz. application).

What emerged from this analysis is that, focusing on the nonhierarchical ECE and MCE NGC_{PROD}, NGC_{LAST}, and $MLGC_{LAST}$ consistently show positive gains considering DL-based models, reaching a $\approx +5\%$ ECE gain and a $\approx +13\%$ MCE gain for the NGC_{LAST} leveraging the $1DCNN$. Moreover, focusing on the MCE metric for the $1DCNN$ model 5 out of 8 hierarchical approaches show a positive gain ranging from $\approx +3\%$ to $\approx +13\%$ MCE. Then, focusing on the hierarchical version of ECE and MCE, which are based on the hierarchical accuracy thus including non-mandatory leaf prediction impact, both $LCPN$ and NGC_{PROD} show an opposite behavior with respect to nonhierarchical metrics by showing positive gains. As a general comment to Fig. 5.5, the NGC_{SUM}, $HLGC$, and $MLGC_{SUM}$ approaches systematically obtain a negative gain considering the four reliability metrics.

5.3.4 Reject Option Analysis

Finally, in this section we show the impact of the reject option with respect to approaches show a positive gain in terms of reliability metrics against the FC, namely the CGC, $LCPN$, $MLGC_{LAST}$, and NGC_{LAST} hierarchical learning approaches, focusing on the best approaches, i.e. DL-based. In detail, figures in Fig. 5.6 show the absolute gains of F1 Score and the percentage of Classified Ratio by varying the reject option threshold in the range $(0, 1)$ with respect to the flat classifiers. It is worth noting that these figures have different colored backgrounds, namely *green* (positive gain), *orange* (negative but limited gain, i.e. > -0.1 for the F1 score and $> -1\%$ for the CR), and *red* (negative gain), which aid in visualize the trade-off between classification performance and classified ratio. Accordingly, in all the figures in Fig. 5.6 the trends of F1 Score gain are opposed to the CR because these two metrics are in trade-off. Noteworthy, the CGC is the approach behave more similarly to the FC, consistently following the zero-gain dashed line for each considered model. Moreover, no couple of F1 Score and CR points falls in the green areas. In other words, hierarchical learning approaches does not beat the trade-off



(a) 1DCNN

(b) 2DCNN

(c) LSTM

(d) 2DCNN+LSTM

(e) MIMETIC

Figure 5.6: Best hierarchical classification approaches vs. FC when reject option is enforced.

offered by the flat approach. The sole exception is given by the $LCPN$ with $1DCNN$ at $\gamma = 0.5$, but the gain w.r.t. FC is very limited. To this extent, we consider as successful the F1 Score/CR trade-offs points with a metric on

the orange background (or very near to the boundary with the red one) and the other on the green. Detailing, looking at Fig. 5.6a (viz. $1DCNN$ model) the NGC_{LAST} obtains a positive gain of $\approx +2\%$ F1 Score with a limited CR gain of $\approx -1\%$ for the threshold $\gamma = 0.2$, up to reach a maximum of F1 Score with $\gamma = 0.7$, namely scoring $\approx +5\%$ with a huge negative gain of $\approx -9\%$ for the CR. On a similar fashion, the $MLGC_{LAST}$ approach obtain $\approx +2\%$ F1 Score with a small negative gain for the CR, i.e. $\approx -1\%$, when $\gamma = 0.9$. Furthermore, from Fig. 5.6b the main finding is related to the NGC_{LAST}, which obtain (at $\gamma = 0.7$) a F1 Score gain of up to $\approx +3\%$ with a CR negative gain of $\approx -10\%$. Finally, the last outcome that is worth to mention is shown in Fig. 5.6e, where the $LCPN$ obtain $\approx +1\%$ CR with a F1 Score reduction of $\approx -1\%$.

Conclusions

In this thesis a Hierarchical Learning Framework for Network Traffic Analysis is proposed, in order to obtain advantages in terms of fine-grain classification performance by exploiting (hierarchical) dependencies among network traffic classes which are included in the training phase of machine- and deep-learning models. Thus, we provide the design choices which results in a framework composed of three fundamental components, namely the traffic segmentation and the features extraction components, and the hierarchical learning engine. Whereas the first two modules enforce data preprocessing procedures in order to group raw network traffic into traffic object, each characterized by a set of features, in order to feed models managed by the hierarchical learning engine. In detail, we designed a top-down approach named *local classifier per parent node*, and 7 global classifier approaches by exploiting well known modeling techniques, such as *multitask learning*, *task-incremental learning*, *multi-label learning*, and *contextual label smoothing*. Moreover, among designed approaches for hierarchical learning, we capitalize the purely hierarchical nature of the local classifier per parent node approach (composed of models arranged in a tree by following the hierarchical dependencies among network traffic classes) to design a Big Data-enabled training procedure, with the objective of speeding up the training phase of such approach by capitalizing both model and data parallelism. Along this direction, we deeply explored the enforcement of a hierarchical reject option, obtained by censoring unconfident classification verdicts, which can result in advantages in terms of classification performance (at the expense of some not classified traffic) and which enables an open-set classification for detecting unknown traffic classes.

Finally, the proposed hierarchical learning framework is evaluated against three use cases, each one related to a particular application of network traffic analysis, namely: (i) the classification of anonymity tools, falling in privacy-related applications; (ii) the detection of intrusions in IoT environments, thus dealing with security related issues; and (iii) the classification of traffic gen-

erated by mobile applications, to facilitate the fine-grain enforcement of traffic management solutions. From these evaluations we (i) assessed the superiority of the local classifier per parent node against a flat (viz. non hierarchical) approach in terms of classification performance, training time, and deployment cost, when dealing with the classification of anonymity tools; (ii) clearly obtained advantages in terms of anomaly detection and unknown attack classification performance by adopting a two levels solution for intrusion detection instead of a single level multi-class misuse detector; and (iii) broadly compared 8 hierarchical learning approaches in performing mobile application traffic classification, showing pros and cons from various perspectives, i.e. classification performance, models reliability, and impact of reject option, both with and without the mandatory leaf node prediction constraint.

Bibliography

[1] Eva Papadogiannaki and Sotiris Ioannidis. A survey on encrypted network traffic analysis applications, techniques, and countermeasures. *ACM Computing Surveys (CSUR)*, 54(6):1–35, 2021.

[2] Alberto Dainotti, Antonio Pescape, and Kimberly C Claffy. Issues and future directions in traffic classification. *IEEE network*, 26(1):35–40, 2012.

[3] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. Multi-classification approaches for classifying mobile app traffic. *Journal of Network and Computer Applications*, 103:131–145, 2018.

[4] Alberto Dainotti, Antonio Pescapé, and Giorgio Ventre. Worm traffic analysis and characterization. In *IEEE international conference on communications*, pages 1435–1442, 2007.

[5] Antonio Montieri, Domenico Ciuonzo, Giuseppe Aceto, and Antonio Pescapé. Anonymity services Tor, I2P, JonDonym: Classifying in the dark. In *IEEE International Teletraffic Congress (ITC 29)*, volume 1, pages 81–89, 2017.

[6] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. Toward effective mobile encrypted traffic classification through deep learning. *Neurocomputing*, 409:306–315, 2020.

[7] Thuy TT Nguyen and Grenville J Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys and Tutorials*, 10(1-4):56–76, 2008.

[8] Petr Velan, Milan Čermák, Pavel Čeleda, and Martin Drašar. A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management*, 25(5):355–374, 2015.

[9] Fannia Pacheco, Ernesto Exposito, Mathieu Gineste, Cedric Baudoin, and Jose Aguilar. Towards the deployment of machine learning solutions in network traffic classification: A systematic survey. *IEEE Communications Surveys & Tutorials*, 21(2):1988–2014, 2018.

[10] Preeti Mishra, Vijay Varadharajan, Uday Tupakula, and Emmanuel S. Pilli. A detailed investigation and analysis of using machine learning techniques for intrusion detection. *IEEE Commun. Surveys Tuts.*, 2018.

[11] Nadia Chaabouni, Mohamed Mosbah, Akka Zemmari, Cyrille Sauvignac, and Parvez Faruki. Network intrusion detection for iot security based on learning techniques. *IEEE Communications Surveys & Tutorials*, 21(3):2671–2701, 2019.

[12] Yanfang Ye, Tao Li, Donald Adjeroh, and S Sitharama Iyengar. A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR)*, 50(3):1–40, 2017.

[13] Giuseppe Aceto and Antonio Pescapé. Internet censorship detection: A survey. *Computer Networks*, 83:381–421, 2015.

[14] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, Valerio Persico, and Antonio Pescapé. Mirage: Mobile-app traffic capture and ground-truth creation. In *International Conference on Computing, Communications and Security (ICCCS)*, pages 1–8. IEEE, 2019.

[15] Alberto Dainotti, Francesco Gargiulo, Ludmila I Kuncheva, Antonio Pescapè, and Carlo Sansone. Identification of traffic flows hiding behind tcp port 80. In *IEEE International Conference on Communications*, pages 1–6, 2010.

[16] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapè. Mimetic: Mobile encrypted traffic classification using multimodal deep learning. *Computer Networks*, 165:106944, 2019.

[17] Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. Robust smartphone app identification via encrypted network traffic analysis. *IEEE Transactions on Information Forensics and Security*, 13(1):63–78, 2017.

[18] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26, 2006.

[19] Manuel Crotti, Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review*, 37(1):5–16, 2007.

[20] Pedro M Santiago del Rio, Dario Rossi, Francesco Gringoli, Lorenzo Nava, Luca Salgarelli, and Javier Aracil. Wire-speed statistical classification of network traffic on commodity hardware. In *Proceedings of the Internet Measurement Conference*, pages 65–72, 2012.

[21] Zhanyi Wang. The applications of deep learning on traffic identification. *BlackHat USA*, 2015.

[22] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 43–48. IEEE, 2017.

[23] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. Mobile encrypted traffic classification using deep learning. In *Proc. IEEE TMA*, 2018.

[24] Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. Network traffic classifier with convolutional and recurrent neural networks for internet of things. *IEEE Access*, 5:18042–18050, 2017.

[25] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, 24(3), 2020.

[26] Carlos N Silla and Alex A Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1):31–72, 2011.

[27] Hong Zhao, Qinghua Hu, Pengfei Zhu, Yu Wang, and Ping Wang. A recursive regularization based feature selection framework for hierarchical classification. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[28] Xinxin Liu and Hong Zhao. Hierarchical feature extraction based on discriminant analysis. *Applied Intelligence*, 49(7):2780–2792, 2019.

[29] Arijit Patra and Julia Alison Noble. Hierarchical class incremental learning of anatomical structures in fetal echocardiography videos. *IEEE journal of biomedical and health informatics*, 24(4):1046–1058, 2020.

[30] Rodolfo M Pereira, Diego Bertolini, Lucas O Teixeira, Carlos N Silla Jr, and Yandre MG Costa. Covid-19 identification in chest x-ray images on flat and hierarchical classification scenarios. *Computer Methods and Programs in Biomedicine*, 194:105532, 2020.

[31] Weijie Zheng and Hong Zhao. Cost-sensitive hierarchical classification via multi-scale information entropy for data with an imbalanced distribution. *Applied Intelligence*, pages 1–13, 2021.

[32] Xinxin Liu and Hong Zhao. Robust hierarchical feature selection with a capped $\ell 2$-norm. *Neurocomputing*, 443:131–146, 2021.

[33] Shunxin Guo, Hong Zhao, and Wenyuan Yang. Hierarchical feature selection with multi-granularity clustering structure. *Information Sciences*, 568:448–462, 2021.

[34] Ju-Youn Park and Jong-Hwan Kim. Incremental class learning for hierarchical classification. *IEEE transactions on cybernetics*, 50(1):178–189, 2018.

[35] Ankita Raj, Anima Majumder, and Swagat Kumar. Hifi: A hierarchical framework for incremental learning using deep feature representation. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–6. IEEE, 2019.

[36] Zhenzhong Kuang, Zongmin Li, Tianyi Zhao, and Jianping Fan. Deep multi-task learning for large-scale image classification. In *2017 IEEE Third International Conference on Multimedia Big Data (BigMM)*, pages 310–317. IEEE, 2017.

[37] Salim Malakouti and Milos Hauskrecht. Hierarchical adaptive multi-task learning framework for patient diagnoses and diagnostic category classification. In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 701–706. IEEE, 2019.

[38] Xiaoni Li, Yucan Zhou, Yu Zhou, and Weiping Wang. Mmf: Multi-task multi-structure fusion for hierarchical image classification. In *International Conference on Artificial Neural Networks*, pages 61–73. Springer, 2021.

[39] Julio Noe Hernandez, Luis Enrique Sucar, and Eduardo F Morales. A hybrid global-local approach for hierarchical classification. In *The Twenty-Sixth International FLAIRS Conference*, 2013.

[40] Chunlin Zhang, Ju Jiang, and Mohamed Kamel. Intrusion detection using hierarchical neural networks. *Pattern Recognition Letters*, 26(6): 779–791, 2005.

[41] Azeem Khan and Shehroz Khan. Two level anomaly detection classifier. In *IEEE ICCEE'08*, pages 65–69, 2008.

[42] Farrukh Aslam Khan, Abdu Gumaei, Abdelouahid Derhab, and Amir Hussain. TSDL: A two-stage deep learning model for efficient network intrusion detection. *IEEE Access*, 2019.

[43] Hui Lu and Jinhua Xu. Three-level hybrid intrusion detection system. In *IEEE ICIECS*, pages 1–4, 2009.

[44] Jae-Hak Yu, Han-Sung Lee, Young-Hee Im, Myung-Sup Kim, and Dai-Hee Park. Real-time classification of internet application traffic using a hierarchical multi-class svm. *KSII Transactions on Internet and Information Systems (TIIS)*, 4(5):859–876, 2010.

[45] Luigi Grimaudo, Marco Mellia, and Elena Baralis. Hierarchical learning for fine grained internet traffic classification. In *International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 463–468. IEEE, 2012.

[46] Ji-hye Kim, Sung-Ho Yoon, and Myung-Sup Kim. Study on traffic classification taxonomy for multilateral and hierarchical traffic classification. In *2012 14th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–4. IEEE, 2012.

[47] Sung-Ho Yoon, Kyu-Seok Shim, Su-Kang Lee, and Myung-Sup Kim. Framework for multi-level application traffic identification. In *Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 424–427. IEEE, 2015.

[48] Wazen M Shbair, Thibault Cholez, Jérôme François, and Isabelle Chrisment. A multi-level framework to identify HTTPS services. In *IEEE/IFIP NOMS*, pages 240–248, 2016.

[49] Chun Guo, Yuan Ping, Nian Liu, and Shou-Shan Luo. A two-level hybrid approach for intrusion detection. *Neurocomputing*, 214:391–400, 2016.

[50] Soo-Yeon Ji, Bong-Keun Jeong, Seonho Choi, and Dong Hyun Jeong. A multi-level intrusion detection method for abnormal network behaviors. *Elsevier JNCA*, 62:9–17, 2016.

[51] Yu-ning Dong, Jia-jie Zhao, and Jiong Jin. Novel feature selection and classification of internet video traffic based on a hierarchical scheme. *Computer Networks*, 119:102–111, 2017.

[52] Quentin Schueller, Kashinath Basu, Muhammad Younas, Mohit Patel, and Frank Ball. A hierarchical intrusion detection system using support vector machine for SDN network in cloud data center. In *IEEE ITNAC*, pages 1–6, 2018.

[53] Zheng Wu, Yuning Dong, Lingyun Yang, and Pingping Tang. A new structure for internet video traffic classification using machine learning. In *2018 Sixth International Conference on Advanced Cloud and Big Data (CBD)*, pages 322–327. IEEE, 2018.

[54] Liangmin Wang, Hantao Mei, and Victor S Sheng. Multilevel identification and classification analysis of tor on mobile and pc platforms. *IEEE Transactions on Industrial Informatics*, 17(2):1079–1088, 2020.

[55] Onur Barut, Yan Luo, Tong Zhang, Weigang Li, and Peilong Li. Multitask hierarchical learning based network traffic analytics. *arXiv preprint arXiv:2106.03850 (Acc. ICC21)*, 2021.

[56] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, Valerio Persico, and Antonio Pescapé. Know your big data trade-offs when classifying encrypted mobile traffic with deep learning. In *IEEE/ACM Network Traffic Measurement and Analysis Conference (TMA)*, 2019.

[57] Xunzhang Li, Yong Wang, Wenlong Ke, and Hao Feng. Real-time network traffic classification based on CDH pattern matching. In *IEEE 14th International Conference on Computational Intelligence and Security (CIS)*, pages 130–134, 2018.

[58] Valerio D'Alessandro, Byungchul Park, Luigi Romano, Christof Fetzer, et al. Scalable network traffic classification using distributed support vector machines. In *IEEE 8th International Conference on Cloud Computing (ICCC)*, pages 1008–1012, 2015.

[59] Martino Trevisan, Idilio Drago, Marco Mellia, Han Hee Song, and Mario Baldi. WHAT: A big data approach for accounting of modern web services. In *IEEE International Conference on Big Data (Big Data)*, pages 2740–2745, 2016.

[60] Jia Lingyu, Liu Yang, Wang Bailing, Liu Hongri, and Xin Guodong. A hierarchical classification approach for tor anonymous traffic. In *IEEE International conference on communication software and networks (ICCSN)*, pages 239–243, 2017.

[61] Masataka Mizukoshi and Masaharu Munetomo. Distributed denial of services attack protection system with genetic algorithms on Hadoop cluster computing framework. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1575–1580, 2015.

[62] Zhengwu Yuan and Chaozheng Wang. An improved network traffic classification algorithm based on Hadoop decision tree. In *IEEE International Conference of Online Analysis and Computing Science (ICOACS)*, pages 53–56, 2016.

[63] Sufian Hameed and Usman Ali. Efficacy of live DDoS detection with hadoop. In *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pages 488–494. IEEE, 2016.

[64] Luong-Vy Le, Bao-Shuh Lin, and Sinh Do. Applying big data, machine learning, and SDN/NFV for 5G early-stage traffic classification and network QoS control. *Transactions on Networks and Communications*, 6 (2):36, 2018.

[65] Amjad Alsirhani, Srinivas Sampalli, and Peter Bodorik. DDoS detection system: Using a set of classification algorithms controlled by fuzzy

logic system in Apache Spark. *IEEE Transactions on Network and Service Management*, 16(3):936–949, 2019.

[66] Jae-Hak Yu, Han-Sung Lee, Young-Hee Im, Myung-Sup Kim, and Dai-Hee Park. Real-time classification of internet application traffic using a hierarchical multi-class svm. *KSII Transactions on Internet and Information Systems (TIIS)*, 4(5):859–876, 2010.

[67] Siddharth Gopal and Yiming Yang. Recursive regularization for large-scale classification with hierarchical and graphical dependencies. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 257–265, 2013.

[68] Eric P Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data*, 1(2):49–67, 2015.

[69] Kai Yang, Hui Ma, and Shaoyu Dou. Fog intelligence for network anomaly detection. *IEEE Network*, 34(2):78–82, 2020.

[70] Niccolo Cascarano, Alice Este, Francesco Gringoli, Fulvio Risso, and Luca Salgarelli. An experimental evaluation of the computational cost of a dpi traffic classifier. In *GLOBECOM IEEE Global Telecommunications Conference*, pages 1–8. IEEE, 2009.

[71] Haipeng Yao, Danyang Fu, Peiying Zhang, Maozhen Li, and Yunjie Liu. MSML: A novel multi-level semi-supervised machine learning framework for intrusion detection system. *IEEE Internet Things J.*, 2018.

[72] Jiaqi Li, Zhifeng Zhao, Rongpeng Li, Honggang Zhang, and Tianhao Zhang. AI-based two-stage intrusion detection for software defined IoT networks. *IEEE Internet Things J.*, 2018.

[73] Imtiaz Ullah and Qusay H. Mahmoud. A two-level hybrid model for anomalous activity detection in IoT networks. In *IEEE CCNC*, pages 1–6, 2019.

[74] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici. N-baIoT —network-based detection of IoT botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 17(3):12–22, 2018.

[75] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint*, 2018.

[76] Fredrik Jejdling et al. Ericsson mobility report. *Ericsson AB, Business Area Networks, Stockholm, Sweden, Tech. Rep. EAB-20*, 9174, November 2020.

[77] Khalid Shahbar and A Nur Zincir-Heywood. Packet momentum for identification of anonymity networks. *Journal of Cyber Security and Mobility*, pages 27–56, 2017.

[78] Stephen R Lawrence and Edward C Sewell. Heuristic, optimal, static, and dynamic schedules when processing times are uncertain. *Journal of Operations Management*, 15(1):71–82, 1997.

[79] Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *arXiv preprint arXiv:1810.12488*, 2018.

[80] Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[81] Cinna Wu, Mark Tygert, and Yann LeCun. A hierarchical loss and its problems when classifying non-hierarchically. *Plos one*, 14(12): e0226222, 2019.

[82] Michael J Trammell, Priyanka Oberoi, James Egenrieder, and John Kaufhold. Contextual label smoothing with a phylogenetic tree on the inaturalist 2018 challenge dataset. *Washington Academy of Sciences. Journal of the Washington Academy of Sciences*, (1):23–45, 2019.

[83] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[84] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[85] 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.

[86] Bernhard Schölkopf, Robert C Williamson, Alexander J Smola, John Shawe-Taylor, John C Platt, et al. Support vector method for novelty detection. In *NIPS*, volume 12, pages 582–588. Citeseer, 1999.

[87] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *IEEE ICDM'08*, pages 413–422.

[88] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF: identifying density-based local outliers. In *ACM Sigmod Record'00*, volume 29, pages 93–104.

[89] Giuseppina Andresini, Annalisa Appice, Nicola Di Mauro, Corrado Loglisci, and Donato Malerba. Exploiting the auto-encoder residual error for intrusion detection. In *IEEE EuroS&PW'19*, 2019.

[90] Aris Kosmopoulos, Ioannis Partalas, Eric Gaussier, Georgios Paliouras, and Ion Androutsopoulos. Evaluation measures for hierarchical classification: a unified view and novel approaches. *Data Mining and Knowledge Discovery*, 29(3):820–865, 2015.

[91] 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.

[92] John Barker, Peter Hannay, and Patryk Szewczyk. Using traffic analysis to identify the second generation onion router. In *IFIP International Conference on Embedded and Ubiquitous Computing*, pages 72–78. IEEE, 2011.

[93] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the ACM workshop on Cloud computing security*, pages 31–42, 2009.

[94] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pages 103–114, 2011.

[95] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website fingerprinting at internet scale. In *NDSS*, 2016.

[96] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated feature extraction for website fingerprinting through deep learning. In *NDSS Symposium*, 2018.

[97] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, pages 1928–1943, 2018.

[98] Gaofeng He, Ming Yang, Junzhou Luo, and Xiaodan Gu. Inferring application type information from tor encrypted traffic. In *Second International Conference on Advanced Cloud and Big Data*, pages 220–227. IEEE, 2014.

[99] Mashael AlSabah, Kevin Bauer, and Ian Goldberg. Enhancing Tor's performance using real-time traffic classification. In *Proceedings of ACM conference on Computer and communications security*, pages 73–84, 2012.

[100] Khalid Shahbar and A Nur Zincir-Heywood. Benchmarking two techniques for tor classification: Flow level and circuit level classification. In *Symposium on Computational Intelligence in Cyber Security (CICS)*, pages 1–8. IEEE, 2014.

[101] Khalid Shahbar and A Nur Zincir-Heywood. An analysis of Tor pluggable transports under adversarial conditions. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7, 2017.

[102] Khalid Shahbar and A Nur Zincir-Heywood. Effects of shared bandwidth on anonymity of the I2P network users. In *IEEE Security and Privacy Workshops (SPW)*, pages 235–240, 2017.

[103] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. Characterization of Tor traffic using time based features. In *ICISSp*, pages 253–262, 2017.

[104] Stefan Burschka and Benoît Dupasquier. Tranalyzer: Versatile high performance network traffic analyser. In *IEEE Symposium Series on Computational Intelligence*, pages 1–8, 2016.

[105] Khalid Shahbar and A Nur Zincir-Heywood. Traffic flow analysis of tor pluggable transports. In *11th International Conference on Network and Service Management (CNSM)*, pages 178–181. IEEE, 2015.

[106] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. DDoS in the IoT: Mirai and other botnets. *IEEE Computer*, 50(7):80–84, 2017.

[107] Ayush Kumar and Teng Joon Lim. Edima: early detection of IoT malware network activity using machine learning techniques. In *IEEE WF-IoT*, pages 289–294, 2019.

[108] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O'Flynn. IoT goes nuclear: Creating a ZigBee chain reaction. In *IEEE SP*, pages 195–212, 2017.

[109] Alberto Dainotti, Antonio Pescapé, and Giorgio Ventre. A cascade architecture for DoS attacks detection based on the wavelet transform. *IOS Press Journal of Computer Security*, 17(6):945–968, 2009.

[110] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges. *IEEE Transactions on Network and Service Management*, 16(2):445–458, 2019.

[111] Sudarshan S. Chawathe. Monitoring iot networks for botnet activity. In *IEEE International Symposium on Network Computing and Applications (NCA)*, pages 1–8, 2018.

[112] Zubair A Baig, Surasak Sanguanpong, Syed Naeem Firdous, Tri Gia Nguyen, Chakchai So-In, et al. Averaged dependence estimators for DoS attack detection in IoT networks. *FGCS*, 2019.

[113] Nickolaos Koroniotis, Nour Moustafa, Elena Sitnikova, and Benjamin Turnbull. Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset. *FGCS*, 100:779–796, 2019.

[114] Osama AlKadi, Nour Moustafa, Benjamin Turnbull, and Kim-Kwang Raymond Choo. Mixture localization-based outliers models for securing data migration in cloud centers. *IEEE Access*, 7:114607–114618, 2019.

[115] Hieu Mac, Dung Truong, Lam Nguyen, Hoa Nguyen, Hai Anh Tran, and Duc Tran. Detecting attacks on web applications using autoencoder. In *ACM SoICT'18*, pages 416–421.

[116] Abebe Abeshu Diro and Naveen Chilamkurti. Distributed attack detection scheme using deep learning approach for internet of things. *Future Generation Computer Systems*, 82:761–768, 2018.

[117] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2), January 2019. ISSN 2157-6904.

[118] Tim Stöber, Mario Frank, Jens Schmitt, and Ivan Martinovic. Who do you sync you are? smartphone fingerprinting via application behaviour. In *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, pages 7–12, 2013.

[119] Qinglong Wang, Amir Yahyavi, Bettina Kemme, and Wenbo He. I know what you did on your smartphone: Inferring app usage over encrypted data traffic. In *2015 IEEE conference on communications and network security (CNS)*, pages 433–441. IEEE, 2015.

[120] Hasan Faik Alan and Jasleen Kaur. Can android applications be identified using only tcp/ip headers of their launch time traffic? In *Proceedings of the 9th ACM conference on security & privacy in wireless and mobile networks*, pages 61–66, 2016.

[121] Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 439–454. IEEE, 2016.

[122] Yanjie Fu, Hui Xiong, Xinjiang Lu, Jin Yang, and Can Chen. Service usage classification with encrypted internet traffic in mobile messaging apps. *IEEE Transactions on Mobile Computing*, 15(11):2851–2864, 2016.

[123] Iyad Lahsen Cherif and Abdesselem Kortebi. On using extreme gradient boosting (xgboost) machine learning algorithm for home network traffic classification. In *2019 Wireless Days (WD)*, pages 1–6. IEEE, 2019.

[124] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence*, 27(8):1226–1238, 2005.