

Università degli Studi di Napoli Federico II
Ph.D. Program in
Information Technology and Electrical Engineering
XXXVI Cycle

Thesis for the Degree of Doctor of Philosophy

Classification and Prediction of Communication and Collaboration Mobile Apps' Traffic via Deep Learning Approaches:

From Data Collection to Models' Explainability

by

Idio Guarino

Advisor: Prof. Antonio Pescapè

Scuola Politecnica e delle Scienze di Base
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

To me,

to Ilaria,

and to our future together

...Always

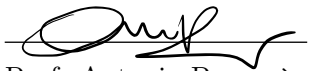
Classification and Prediction of Communication and Collaboration Mobile Apps' Traffic via Deep Learning Approaches:
From Data Collection to Models' Explainability

Ph.D. Thesis presented
for the fulfillment of the Degree of Doctor of Philosophy
in Information Technology and Electrical Engineering
by

Idio Guarino

November 2023

Approved as to style and content by



Prof. Antonio Pescapè, Advisor

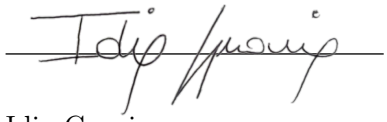
Università degli Studi di Napoli Federico II
Ph.D. Program in Information Technology and Electrical Engineering
XXXVI cycle - Chairman: Prof. Stefano Russo

http://itee.dieti.unina.it

Candidate's declaration

I hereby declare that this thesis submitted to obtain the academic degree of Philosophiæ Doctor (Ph.D.) in Information Technology and Electrical Engineering is my own unaided work, that I have not used other than the sources indicated, and that all direct and indirect sources are acknowledged as references.
Parts of this dissertation have been published in international journals and/or conference articles (see list of the author's publications at the end of the thesis).

Napoli, January 10, 2024



Idio Guarino

Abstract

Internet traffic is constantly changing as networks and applications evolve. The COVID pandemic has greatly increased the use of **communication and collaboration mobile apps (CC apps)**, significantly affecting the volume of traffic they generate. This requires appropriate management of network resources, also based on the multiple activities that can be performed by such apps. This Thesis focuses on the traffic generated by CC apps to understand their peculiarities and design advanced tools for their **classification** and **prediction**. Integrating these tools can result in a more flexible and adaptable network management strategy. To address modern traffic challenges, they use **Deep Learning (DL)** techniques that can adapt to frequently changing input data and improve performance on complex tasks. Given the centrality of data, experimental analyses exploit **real traffic from popular CC apps**, purposely collected and publicly released. First, a **characterization** of the traffic is provided to highlight its distinctive features. Next, a novel approach to **traffic classification** is designed to overcome experimentally verified state-of-the-art gaps, particularly by exploiting contextual representations of observed traffic. Innovative approaches for **traffic prediction** at both packet- and aggregate-level are then designed by evaluating the trade-off between performance and complexity. Finally, to address the limitations in transparency and reliability due to the black-box nature of DL models, the **interpretability** of their results is analyzed via eXplainable Artificial Intelligence techniques. In summary, this Thesis investigates and tackles the peculiar challenges of CC apps' traffic from multiple perspectives. It provides new solutions to support classification and prediction and focuses on interpretability and reliability issues arising from DL adoption.

Keywords: communication-and-collaboration-apps, encrypted traffic, deep learning, traffic classification, traffic prediction, XAI.

Sintesi in lingua italiana

Il traffico Internet cambia costantemente con l'evolversi delle reti e delle applicazioni. La pandemia da COVID ha incentivato l'uso delle **app mobili di comunicazione e collaborazione (CC app)**, incidendo in modo significativo sul volume di traffico che esse generano. Ciò richiede un'adeguata gestione delle risorse di rete, anche in base alle molteplici attività che possono essere svolte da tali app. Questa Tesi si focalizza sull'analisi del traffico delle CC app per comprenderne le peculiarità e progettare strumenti innovativi per la sua **classificazione** e **predizione**. L'integrazione di questi strumenti può portare a una strategia di gestione della rete più flessibile e adattabile. Per affrontare le sfide del traffico moderno, essi utilizzano tecniche di **Deep Learning (DL)** in grado di adattarsi a dati di input che cambiano frequentemente e di migliorare le prestazioni su problemi complessi. Data la centralità dei dati, le analisi sperimentali utilizzano **traffico reale di CC app popolari** appositamente collezionato e rilasciato pubblicamente. Dapprima si effettua una **caratterizzazione** del traffico mirata a evidenziarne le caratteristiche distintive. Quindi si progetta un approccio innovativo per la **classificazione**, volto a superare le limitazioni dello stato dell'arte verificate sperimentalmente, sfruttando rappresentazioni contestuali del traffico. Si propongono inoltre soluzioni avanzate per la **predizione** del traffico a livello pacchetto ed aggregato, valutando il compromesso tra prestazioni e complessità. Infine, alla luce delle limitazioni di trasparenza ed affidabilità scaturite dalla natura black-box dei modelli DL, si analizza l'**interpretabilità** dei loro risultati attraverso tecniche di eXplainable Artificial Intelligence. In sintesi, questa Tesi investiga le problematiche inerenti agli strumenti per l'analisi del traffico delle CC app da varie prospettive, fornendo nuove soluzioni a supporto della classificazione e predizione e si concentra sui problemi di interpretabilità e affidabilità derivanti dall'adozione del DL.

Parole chiave: app di comunicazione e collaborazione, traffico cifrato, deep learning, classificazione del traffico, predizione del traffico, XAI.

Acknowledgements

I am extremely grateful to my supervisor, Prof. Antonio Pescapè, for his invaluable guidance and unwavering support throughout my doctoral journey. His expertise, encouragement, and mentorship have shaped my research and academic progress. I am truly thankful for the opportunity to learn under his leadership and I am inspired by his dedication to fostering a nurturing and intellectually stimulating research environment.

I want to extend my heartfelt thanks to my labmates (in rigorous alphabetical order)—Alfredo, Antonio, Ciro, Davide, Domenico, Francesco, Giampaolo, Giuseppe, and Valerio—who have been a constant source of inspiration with their diverse perspectives and approaches. Every discussion and suggestion has aided me in tackling challenges and scientific glitches rigorously and systematically. Moreover, they are not just colleagues but also friends. Thank you to all the friends of Arclab, a place where I leave a piece of my heart.

I want to thank Consortium GARR, the Italian national network of University and Research, for supporting part of the work presented in this thesis through the "O. Carlini" research grant. Its financial support was instrumental in advancing my academic activities and allowed me to advance my field of study. I want to thank Dr. Eng. Alessandro Finamore, my internship advisor at the Huawei R&D Center in Paris. He played an important role in my professional growth by providing me with different perspectives on research. I express my sincere gratitude to Prof. Stefano Russo for his kindness, availability, and expertise in guiding our ITEE Ph.D. program. I am grateful to Prof. Marilia Curado (University of Coimbra, Portugal) and Dr. Claudio Fiandrino (IMDEA Networks Institute, Spain) for their valuable comments on the draft version of my Ph.D. thesis, which helped me improve it significantly.

I express my deep gratitude to my parents and brother Nicola. The

achievement of this milestone is entirely due to you. Throughout my years at the university, you have consistently supported every decision I have made, always showing confidence in my abilities, and never asking for anything in return. I hope to repay your sacrifices, which have made my growth and inclusion in this society possible. I also want to thank my grandparents and my aunt Maria.

Finally, I want to thank Ilaria. You have changed my life and your love, presence, and patience have been instrumental in reaching this milestone, which is only the starting point for our much-dreamed life together.

Contents

List of Acronyms

The following acronyms are used throughout the thesis.

`#CF`	Number of Contextual Biflows
`#TP`	Number of Trainable Parameters
`ACall`	Audio-Call activity
`ACT`	Activity-related ground truth
`Act-TC`	Activity Traffic Classification task
`ADB`	Android Debug Bridge
`ALL`	Overall model
`APP`	App-related ground truth/Per-app model
`App-TC`	App Traffic Classification task
`APPxACT`	Joint App-and-Activity ground truth
`ARIMA`	AutoRegressive Integrated Moving Average
`AVP`	Average Value Predictor
`bCONTEXT`	Branch of MIMETIC-ALL taking as input `CONTEXT`
`BiGRU`	Bidirectional Gated Recurrent Unit
`bPAY`	Branch of MIMETIC-ALL taking as input `PAY`
$\texttt{BR}_\texttt{u}$/$\texttt{BR}_\texttt{d}$	Upstream/Downstream bit-rate
`bSEQ`	Branch of MIMETIC-ALL taking as input `SEQ`
`Chat`	Chat activity
`CNN`	Convolutional Neural Network

CONTEXT	Context inputs computed at the arrival of the N_p-th packet
CovLSTM	Convolutional LSTM
CV	Current Value
CW-ECE	Class-Wise Expected Calibration Error
DCC	Dilated Causal Convolutional Layer
DIR	Packet Direction
DL	Deep Learning
DNN	Deep Neural Network
DQN	Deep Q-Network
DSANet	Dual Self-Attention Network
DT	Decision Tree
ECE	Expected Calibration Error
FARIMA	Fractional Auto-Regressive Integrated Moving Average
G-mean	Geometric-Mean
GNN	Graph Neural Network
GRU	Gated Recurrent Unit
HMM	Hidden Markov Model
IAT	Inter-Arrival Time
JNN	Jordan Neural Network
Joint-TC	Joint App-and-Activity Traffic Classification task
k-NN	K-Nearest Neighbors Regressor
LIME	Lightweight Interpretable Model-Agnostic Explanations
LR	Logistic Regressor
LSTM	Long Short-Term Memory
MA	Moving Average
MC	Markov Chain
MCE	Maximum Calibration Error
ML	Machine Learning
MLP	Multi-Layer Perceptron
MM	Multi-Modal architecture

MMG	Markov Modulated Gamma
MSE	Mean Squared Error
MT	Multitask architecture
NAT	Network Address Translation
$\mathtt{P_u}/\mathtt{P_d}$	Amount of transmitted/received packets
PAY	First N_b bytes of transport-layer payload
$\mathtt{PKTs_{up}}/\mathtt{PKTs_{dw}}$	Amount of transmitted/received packets in the time interval Δ
PL	Transport-layer Payload Length
PMF	Persistence Model Forecast
$\mathtt{PR_u}/\mathtt{PR_d}$	Upstream/Downstream packet-rate
PROTO	Per-proto model
PS	Packet-Size
RBF	Radial-Basis Function
RCLSTM	Random Connectivity LSTM
RF	Random Forest
RFR	Random Forest Regressor
RNN	Recurrent Neural Network
SAE	Stacked AutoEncoder
SEQ	Informative fields extracted from the sequence of the first N_p packets
SERIESNET	SeriesNet
SHAP	Shapley Additive Explanations
STL	Seasonal-Trend decomposition procedure based on Loess
SVM	Support Vector Classifier
SVR	Support Vector Regressor
TC	Traffic Classification
TCPWIN	TCP Window Size
t-SNE	t-Distributed Stochastic Neighbor Embedding
$\mathtt{V_u}/\mathtt{V_d}$	Amount of transmitted/received byte
VCall	Video-Call activity
$\mathtt{VOL_{up}}/\mathtt{VOL_{dw}}$	Amount of transmitted/received byte in the time interval Δ

`W`	Memory Window Size
`XAI`	eXplainable Artificial Intelligence
`ZP`	Zero Predictor

List of Figures

List of Tables

Chapter 1

Introduction and Background

The Web does not just connect machines, it connects people.

Sir Tim Berners-Lee

NETWORK traffic and the underlying infrastructure share a mutually influential relationship, evolving in harmony. Traffic adjusts to take advantage of advanced network capabilities, like increased bandwidth, reduced latency, and enhanced resilience and flexibility, while the infrastructure adapts to fulfill the requirements arising from emerging applications. Recent reports from global operators show that fixed and mobile broadband usage has increased significantly in the past four years, with growth rates ranging from 20% to 50% and 15% to 35%, respectively [1]. The COVID pandemic has accelerated the pace of digitization by several years and has caused millions of people to stay at home, work, and study remotely.

This has led to a significant increase in internet traffic from residential users, with a +20% increase in the volume of EU internet traffic [2]. This shift has also affected user mobility patterns in cellular networks, with an increase in voice and uplink traffic in suburbs (+10%) [3]. Similar changes have been observed in campus networks [4, 5, 6, 7]. Changes in online presence have resulted in a *measurable impact on network performance*, such as increased variability in delay, loss rate, and latency [8], and degradation of over-the-top services [9]. These changes have also been

used for global-scale inference of work-from-home and lockdown events and maps [10]. The widespread adoption of remote work, remote education, online commerce, and entertainment activities during lockdowns and periods of social distancing has caused these shifts. However, with COVID becoming more endemic in many parts of the world, the continuous increase in Internet traffic is still mainly attributed to enduring habits and the widespread adoption of *communication and collaboration apps* (*CC apps* in the following) that people have become accustomed to. This phenomenon is not driven solely by lingering COVID-related behaviors, but signifies the establishment of permanent habits and the global familiarity with these ubiquitous apps.

In this complex and dynamic scenario, network resource management faces a challenge due to unexpected and massive changes and variations in traffic patterns. To ensure a better customer experience, efficient network monitoring and management capabilities are required. Consequently, on the one hand, the process of *Traffic Classification* [11] (TC in the following), which involves inferring the application or service type responsible for the observed traffic, becomes crucial for optimizing network management based on the transmitted content. TC is fundamental for various management, planning, and policy enforcement initiatives.

On the other hand, ensuring optimal network performance requires implementing automated and adaptive management of network resources [12]. This helps achieve reactive systems that promptly respond to observed network traffic. However, a *proactive* system that can predict the future traffic pattern, provide ample time to plan, and execute the appropriate actions would be even more effective. Therefore, the process of *Traffic Prediction* [13] (TP in the following), which involves obtaining information about how traffic characteristics will evolve, is crucial for predicting network needs. TP helps optimize resource provisioning and enables the application of new techniques, architectures, and services to meet the predicted demands. Moreover, incorporating TC and TC can result in a more adaptable and flexible network management strategy [14]. In particular, it can help in developing specific prediction models for each app or service. These models can be trained using historical data to identify patterns, trends, and variations in the traffic generated by each app and predict how it will evolve in the future. Additionally, integrating classified infor-

mation with predictions allows resources to be dynamically allocated. For instance, if a critical app is expected to experience a surge in traffic, its traffic can be prioritized by allocating more bandwidth or server resources to handle the expected load and ensure a higher quality of service.

Nevertheless, despite TC and TP being long-standing and actively researched topics, the specific characteristics of mobile apps make their application even more challenging. In detail, the predominant use of *encryption* in the traffic generated by CC apps makes classification techniques based on content inspection (i.e., payload) impractical, as it limits the exploitable information (i.e., cleartext) exchanged between parties.

Moreover, established communications often refer to port numbers that are dynamically assigned or subject to NAT protocol interference. As a result, payload inspection (i.e., *payload-based*) and port analysis (i.e., *port-based*) techniques are not applicable. In addition, the spread of apps with servers hosted on (a few) cloud infrastructures further confounds IP address-based methods. Finally, the use of application protocols as transport sublayers (TLS and HTTP), common to even very different apps, further complicates the problem.

The difficulties related to proper traffic classification and prediction are further exacerbated in the case of mobile apps due to their numerousness and heterogeneity, with consequences on the ability to collect and analyze sufficient quantities of traffic. On one hand, according to data recorded during Q3 2022, the number of mobile apps available in the Google Play Store and the Apple App Store stands at around 2.7 million and 1.6 million [15, 16], respectively. On the other hand, the heterogeneity of these apps is due to their execution on mobile devices, which are platforms characterized by automatic and frequent updates and the presence of different functioning modes (*activities*) for a single application. Indeed, what makes CC apps unique from others is their *multi-activity* nature, which means they no longer deal with a single type of traffic: it is increasingly common to have video, voice, chat, and game content within the same app [1].

As we show in our experimental evaluation, this specific characteristic poses several challenges: *(i)* different activities in the same app present different traffic patterns while being similar to the same activity within other apps—this can likely confuse the classifiers and lead to poor performance (as we prove experimentally); *(ii)* different activities have different require-

ments in terms of Quality of Service, policy enforcement, and monitoring—managing traffic at the app level overly extends the network management actions to be performed; *(iii)* analyses of users' behaviors, to identify needs, and plan infrastructure and service deployments, are impacted by the coarseness of surveyed information. These challenges are now pushing toward the adoption of advanced *Deep Learning* (DL) approaches, able to cope with frequently changing input nature, and offering promising performance when dealing with complex and hard-to-model problems [17, 18, 19]. Compared to traditional rule- or ML-based approaches, which rely on obtaining handcrafted (domain-expert driven) rules or features, DL has a significant advantage in automatically extracting useful features from input data through end-to-end learning. Indeed, for traditional approaches, such a process is time-consuming and unsuitable for automation. Additionally, they quickly become obsolete compared to the constantly evolving mobile traffic, making it difficult to design accurate and up-to-date solutions. Moreover, DL models achieve state-of-the-art performance by jointly training the feature extraction and classification (resp. prediction) and allowing for expert-free updates.

Thus, on the one hand, there is a need for a deeper understanding of the traffic patterns of applications that have seen a surge in utilization after the COVID pandemic. On the other hand, an assessment and improvement of modern TC and TP approaches applied to this specific scenario is needed.

Following these considerations, the first objective of this thesis is to tackle the *app- and activity-level early classification of network traffic generated by the most popular communication and collaboration mobile apps*, whose utilization has increased with the COVID-19 pandemic—and keeps shaping the nature of Internet traffic. Specifically, we target nine of such apps that have seen a dramatic increase in usage during and after the lockdowns, namely `Discord`, `GotoMeeting`, `Meet`, `Messenger`, `Skype`, `Slack`, `Teams`, `Webex`, and `Zoom`. We analyze their traffic and assess the performance of the state-of-art DL approaches for classifying the specific app or the kind of activity the user performs, involving `ACall`, `Chat`, and `VCall`. As we find results much wanting, informed by traffic characterization of mobile apps [20] we propose and evaluate a *novel set of inputs observed from contextual traffic from the same app*, and design a novel architecture exploiting these data as well, showing *much-improved activity-classification*

performance. Exploiting such novel inputs, we also design a novel DL architecture *that does not rely on application-layer payload*, with minimal loss of accuracy but better training time and increased robustness to more opaque encrypted protocols. Although the present work focuses on internet traffic data generated by CC apps, whose use has grown significantly in conjunction with and after the COVID pandemic, the proposed methodology is general. Indeed, it can be applied in all contexts in which it is crucial to classify the app and the activity performed by the user (e.g., QoS/QoE management and user profiling).

Moreover, to respond to the need for effective solutions to predict network traffic, we focus on *fine-grained prediction* by designing a *novel DL* solution that predicts at *packet-level* granularity. Further, we assess if there is room for improvements by having *prediction models tailored on specific apps and protocols*. This approach diverges from previous methods, which typically focused on *aggregated metrics* covering extensive time intervals, such as total volume and average rate. Although existing scientific literature has demonstrated the suitability of deep learning-based solutions for predicting aggregate traffic [21, 22], leveraging deep learning for fine-grained forecasting presents ongoing challenges that require overcoming performance gaps [23].

Finally, in line with the concerns related to the adoption of Artificial Intelligence (AI) for driving critical systems, we also deepen aspects related to the *trustworthiness* of the designed solutions, focusing on *technical robustness* and *transparency*. In fact, DNN-based solutions have shown significant improvements in various application domains. They provide a general approach to solving a wide range of problems that were traditionally solved using specific solutions. However, these solutions often lack interpretability, which can hinder their practical adoption. The inability to understand these models can lead to several problems, including a lack of trust in their performance in new environments, difficulty in debugging wrong decisions, and vulnerability to adversarial manipulation [24].

In response to these issues, The European Commission's High-Level Expert Group on Artificial Intelligence (AI HLEG) has set guidelines for AI systems [25], which require AI systems to be transparent and understandable to humans. It means that people should be able to understand how these systems work and make informed decisions regarding their us-

age. Additionally, the security and reliability of AI systems need to be ensured to minimize potential risks. Lastly, AI systems must respect the fundamental rights of users, such as privacy and non-discrimination. These guidelines are crucial for compliance with the European AI Act [26]. The act was approved by the European Commission on December 9, 2023, to ensure the safe and dependable use of AI for the benefit of society.

1.1 Contribution and organization of the thesis

As depicted in Fig. 1.1, the *main contributions* of this doctoral thesis are summarized in the following:

Traffic Collection

We collect and publicly release[1] a novel dataset, named MIRAGE-COVID-CCMA-2022 [2], encompassing the traffic of nine *Communication and Collaboration mobile apps (CC apps)* run by users executing three different activities, namely Audio-call, Video-call, and Chat. The collected dataset is human-generated, recent, and *reliably labeled* at the biflow level, with both the *mobile app* that generated the traffic and the *activity* that the user performed. A biflow (or bidirectional flow) is an aggregation of packets sharing the common 5-tuple (transport-layer protocol and destination/source IP address and transport-layer port) regardless of their direction. The considered apps have experienced a sudden change in volume and spatial/temporal patterns during the pandemic. They are of specific interest to network operators, network managers, academia, and society at large.

Traffic Characterization and Modeling

We analyze the collected traffic in terms of *adopted protocols*, and then we characterize it in terms of bitrate, packet rate, and upstream/downstream ratio (in bit volume and packet volume) at different levels of granularity, namely at *trace* and *flow* level, for each *app* and related *activities*.

[1] http://traffic.comics.unina.it/mirage/mirage-covid-2022

[2] CCMA stands for Communication and Collaboration Mobile Apps

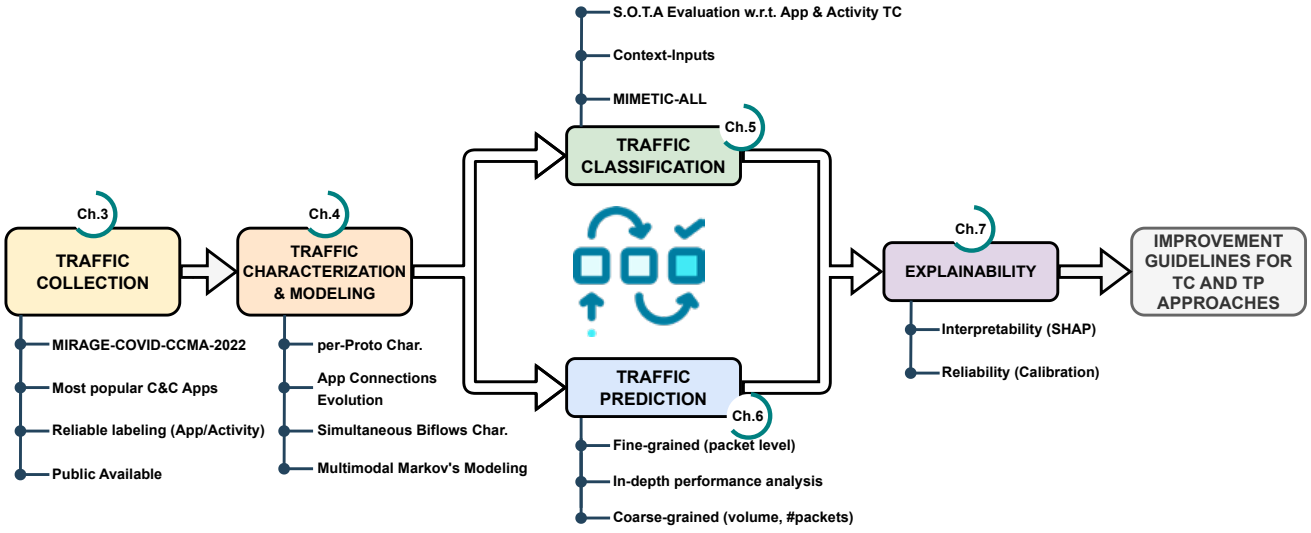


Figure 1.1. Workflow of the study presented as part of this thesis. For each contribution (milestone), the corresponding chapter is indicated.

Furthermore, we further characterize the sequence of packet sizes and directions with data-driven modeling solutions based on *Multimodal Markov Chains*, at *per-activity* granularity.

Traffic Classification

We experimentally evaluate the capability of *state-of-the-art DL approaches* to address the early classification of biflows constituting the traffic generated by such CC apps at different granularity levels—i.e., *app* and *activity*. Specifically, we use state-of-art DL architectures with optimized input size—such as a 1D-CNN [27], a hybrid 2D-CNN+LSTM [28], and the multimodal Mimetic-Enhanced architecture [29]—to model the collected dataset, with the objective of classifying the traffic at *app*, *activity*, and *app*×*activity* granularity. Our findings highlight the shortcomings deriving from the traffic inputs leveraged by these solutions in activity recognition.

Prompted by the limitations in classifying user activities deriving by the inputs commonly adopted in related literature [18], we investigate the traffic patterns generated by a traffic source and design a novel set of inputs, named *Context inputs*, able to provide hints about a biflow by observing its context (i.e., the set of co-existing biflows which run in parallel). Hence, we perform a characterization analysis suggesting that Context inputs are able to support user-activity classification.

Capitalizing on the appeal of Context inputs, we design a *novel classification solution* starting from Mimetic-Enhanced, named Mimetic-All, which leverages Context inputs as an additional modality. Our Mimetic-All effectively exploits the Context inputs obtaining *always the best performance* when compared with both ML- and DL-based TC solutions fed with the same input while also being suitable for *early traffic classification*.

Exploiting the multimodal nature of the architecture, we also provide a *second novel classification solution*—we name Mimetic-ConSeq—that

trades the inputs based on the payload for the new Context inputs, paying a negligible performance cost (less than 1% F-measure drop for app classification w.r.t. Mimetic-Enhanced) to gain both a shorter training time and greater robustness to future more opaque encryption sublayers (e.g., TLS with *Encrypted Server Name Indication* or *Encrypted Client Hello* extensions [30]).

Traffic Prediction

We tackle the challenging task of predicting network traffic at the finest granularity level, namely the *packet level*, by leveraging Deep Neural Networks (DNNs). Referring to a biflow as the analyzed object, our proposed models utilize a limited memory of previously observed packets to accurately predict various characteristics of the next packet, such as its direction, inter-arrival time, and transport-layer payload length. This fine-grained prediction capability empowers efficient traffic management at the biflow level and on short timescales, making valuable contributions to advancing next-generation networks.

Focusing on the traffic of CC apps, we investigate the advantage of *multitask DL models* designed for specific apps or protocols (TCP/UDP) with respect to a *single-model* trained on all the considered CC apps' traffic. In a complementary fashion, we assess how traffic prediction performance varies from multiple perspectives: (*i*) different *apps*, (*ii*) different *activities*, or (*iii*) different *protocols*.

By capitalizing on the outcomes of the fine-grained packet-level multitask predictor, we define an approach that can deal with *coarser-grained traffic prediction tasks* (e.g., prediction of traffic volume and number of packets) with *arbitrary aggregation intervals*.

Explainability

To overcome the limitations of black-box AI models, particularly Deep Neural Networks (DNNs), we use *eXplainable Artificial Intelligence (XAI)* techniques for both traffic classification and prediction tasks. XAI allows us to evaluate model performance by assessing the *reliability* of each prediction. It also helps us identify connections between performance and traffic characteristics by assigning importance to different input parameters. By utilizing XAI, we strengthen our confidence in the results and identify areas for improvement. To the best of our knowledge, this study is the first to propose a DL approach evaluated using XAI to predict network traffic at the packet level.

The rest of the thesis is organized as follows. Chapter 2 surveys previous research related to changes in Internet traffic during the COVID pandemic, as well as ML/DL techniques for app and app-user activity identification and traffic prediction. Additionally, Chap. 2 covers the trustworthiness analysis of DL models that are focused on network traffic analysis, and also provides the positioning of this work against related literature. Then, Chap. 3 details the MIRAGE-COVID-CCMA-2022 dataset covering real and human-generated traffic of popular CC apps and representing the basis for the entire study proposed in this doctoral thesis. Following, Chap. 4 provides the characterization of the collected traffic at different levels of granularity and the modeling of the sequence of packet sizes and directions with *Multimodal Markov Chains*. In Chap. 5, we highlight the shortcomings of current state-of-art (multimodal) DL-based approaches used for TC. Then, we introduce the Context inputs and describe our novel proposals based on such inputs. The chapter ends with the experimental analysis and related take-home messages. Hence, Chap. 6 details our methodology regarding packet-level traffic prediction via multitask DL and the prediction of traffic aggregates. It also presents the experimental setup and evaluation with related take-home messages. In Chap. 7 we detail our methodology regarding the XAI techniques adopted to investigate the interpretability and reliability of both TC and TP DL models. The chapter also presents the results of this analysis and the corresponding take-home messages. Finally, Chap. 8 provides conclusions of this thesis.

Chapter 2

Related works and Positioning

In this chapter, we first discuss the recent studies focusing on the impact of COVID pandemic on Internet traffic (Sec. 2.1). Then, we provide an overview of the works that have addressed app or activity traffic classification (Sec. 2.2) or traffic prediction (Sec. 2.3). We also detail works that exploited XAI techniques in the context of network traffic analysis (Sec. 2.4). For each reviewed topic, at the end of the respective section, we provide the positioning of the present thesis against the related literature.

2.1 Impact of COVID on the Nature of Internet Traffic

Several studies have investigated the changes that the spread of the COVID pandemic on a global scale has caused to Internet traffic as a result of lockdown periods and the consequent shift in daily habits. Regarding the increasing use of tools useful for smart working and distance learning, Affinito et al. [31] provide insights on the usage of different categories of Internet applications for collaboration and entertainment by analyzing websites and domains visited during the enforcement of the lockdown to contain the spread of the virus. As shown, during the reference period, the most used applications were `Youtube`, `Netflix`, `Facebook`, `Whatsapp`, `Skype`, and `Zoom`. Other works investigate the variation in traffic volumes and network performance [2, 3, 4, 8, 9]. Specifically, Feldmann et al. [2] an-

alyze the effect of the lockdowns on traffic shifts during the period Mar-Jun 2020, by using network flow data from multiple vantage points (i.e., an ISP, three IXPs, and a large academic network). They point out that during the period considered, the volume of European Internet traffic increased by up to +20%, mainly due to residential traffic generated by applications for work and distance education, whose volume increased by up to 200%. Lutu et al. [3] analyze the changes in mobility and their impact on cellular network traffic, showing an overall increase in voice traffic, with a decrease in download traffic (−20%) and an increase in uplink traffic (+10%). Additional degradations of network services are highlighted by Böttger et al. [9], who analyze the traffic growth in different regions of the world and how the network responded to an increased demand from the the perspective of edge network of `Facebook` during the beginning of the COVID pandemic. The authors observed a worldwide increase in traffic throughput between March and July 2020 and a correlation between the phase of traffic growth and the spread of COVID for each region. Similar network traffic shifts are also investigated by Candela et al. [8], highlighting a higher variability in latency and loss rates in Italy during the first weeks of the 2020 lockdown w.r.t. the pre-pandemic period. Finally, Favale et al. [4] analyze the impact of lockdown measures and the switch to online collaboration and e-learning solutions on a university campus during March/April 2020, showing a peak of 1.5Gbit/s in the university network traffic due to the increased use of digital tools for collaboration and remote working.

Positioning w.r.t. Related Literature. Starting from the findings of the studies above, this thesis focuses on the traffic of CC apps. The wide adoption of these apps, both during and after the COVID pandemic, has resulted in significant changes in network traffic.

2.2 Internet Traffic Classification

In this section, we position our study by detailing the latest advancements in TC via DL (Sec. 2.2.1) and user activity recognition (Sec. 2.2.2). Table 2.1 summarizes the main aspects of each work surveyed herein. We categorize each paper based on (a) the object of classification, i.e., the traffic unit will receive the classification label. We then specify (b) whether the proposal uses DL techniques, (c) the specific classification techniques

Table 2.1. Related work tackling app or activity traffic classification tasks using different approaches. Reported papers are listed in chronological order. The last row summarizes the current proposal.

Reference	TO	DL	Techniques	MM	Input Data	App	Activity	Mobile	Data
Wang et al. [32]	BF	✓	SAE		PAY (TCP)	○	○		
Lopez-Martin et al. [28]	BF	✓	2D-CNN+LSTM		SEQ(6 fields)	○	◐		
Wang et al. [27]	F/BF	✓	1D-CNN		PAY(L4), PCAP	●	◐		✓
Huang et al. [33]	BF	✓	2D-CNN		PCAP	●	◐		✓
Aceto et al. [34]	BF	✓	1D-CNN&BiGRU	✓	PAY(L4), SEQ(4 fields)	●	○	✓	✓
Liu et al. [35]	BF	✓	BiGRU		SEQ(1 field)	●	○	✓	
Rezaei et al. [36]	BF	✓	1D-CNN		PAY(L4)	●	○	✓	
Zeng et al. [37]	BF	✓	1D-CNN, LSTM SAE		PCAP	○	○		✓
Zhao et al. [38]	BF	✓	DNN		Flow statistics	●	◐		✓
Lotfollahi et al. [39]	P	✓	1D-CNN, SAE	✓	PAY(L2)	●	◐		✓
Wang et al. [40]	BF	✓	1D-CNN&LSTM	✓	PAY(TLS), SEQ(1 field)	●	○	✓	
Aceto et al. [41]	BF	✓	1D-CNN&BiGRU	✓	PAY(L4), SEQ(4 fields)	●	◐		✓
Akbari et al. [42]†	BF	✓	1D-CNN&BiGRU&MLP	✓	PAY(TLS), SEQ(3 fields), Flow statistics	●	◐	✓	
Conti et al. [43]	SB		RF		Clustering-based statistics	○	●	✓	
Saltaformaggio et al. [44]	SB		SVC		Burst Statistics	●	●	✓	
Grolman et al. [45]	SB		RF, AB		Burst statistics	○	●	✓	
Aiolli et al. [46]	SB		SVC, RF		Burst statistics	●	●	✓	
Li et al. [47]‡	TS	✓	1D-CNN&2D-CNN	✓	Traffic Segment	●	●	✓	
Li et al. [48]	BF	✓	LR, RNN, CNN		SEQ(7 fields)⋆	○	◐	✓	
This Thesis	BF	✓	1D-CNN&BiGRU&MLP	✓	PAY(L4), SEQ(4 fields), Context Inputs	●	●	✓	✓

TO: Traffic Object. **TS**: Traffic Stream. **SB**: service-burst. **F**: flow. **BF**: biflow. **P**: packet. **DL**: Deep-learning. **MM**: Multimodal architecture. **PAY**: Payload bytes. **PCAP**: Raw data of PCAP trace. **SEQ**: (Bi)Flow packet sequence. **App**: App-TC task. **Activity**: Activity-TC task. **Mobile**: Mobile data. **Data**: Publicly Available Dataset. ● present. ◐ partially present (Traffic Service). ○ lacking.
†: disjoint evaluation on non-mobile app traffic and mobile service traffic. ⋆: biased inputs.
&: indicates multi-modal techniques using heterogeneous DL layers.

employed, and (d) whether it uses a multimodal model. We also specify (e) the type of input traffic used to feed the model. Then, we indicate whether these techniques are designed for (f) app or (g) activity traffic classification. Finally, the last two columns flag the works (h) focusing on mobile traffic and (i) publicly releasing the dataset used in their experimentations, respectively. The present section ends with the positioning of our work, whose main points are recapped in the last row of Tab. 2.1.

2.2.1 Deep Learning-based (Mobile) Traffic Classification

In recent years, several works have faced TC via DL approaches. Wang et al. [32] has first used a Stacked AutoEncoder (SAE) for unencrypted traffic identification, achieving superior performance w.r.t. standard neural networks ($\geq 90\%$ precision and recall). **Encrypted TC** is targeted by Wang et al. [27], who propose a method based on 1D Convolutional Neural Network (1D-CNN)—outperforming the 2D variant—to tackle four different TC tasks: (i) VPN/nonVPN, (ii) 6 encrypted traffic classes, (iii) 6 VPN-tunneled traffic classes, and (iv) 12 encrypted applications. Similar tasks are tackled by Lotfollahi et al. [39] proposing *Deep Packet* (based on 1D-CNN and SAE), able to outperform ML-based classifiers for encrypted TC at packet granularity. This proposal outperforms ML-based classifiers in both application identification and traffic characterization. Recurrent Neural Networks have been considered by Lopez-Martin et al. [28], proposing different hybrid DL architectures that combine Long Short-Term Memory (LSTM) and 2D-convolutional layers. Zeng et al. [37] propose a framework for **encrypted TC** and **intrusion detection**—named *Deep-Full-Range*—based on three different DL architectures (i.e., CNN, LSTM, and SAE) using raw traffic as input data. The framework was evaluated on both classification tasks by comparing performance with state-of-the-art methods on two public datasets, obtaining better performance on both tasks.

Focusing on the classification of **mobile-app traffic**, Rezaei et al. [36] leverage a CNN fed with the header and the payload of the first six packets of a biflow. Similarly, Liu et al. [35] devise *FS-Net*, an encoder-decoder architecture based on Bidirectional Gated Recurrent Units (BiGRU) taking as input IP-packet sizes of flow sequences. In this context, Aceto et al. [18] define a systematic framework to dissect the encrypted mobile TC

using DL, and compare a number of the aforementioned techniques for a comprehensive evaluation. Common usage of biased inputs (e.g., local-network metadata [27], or source and destination ports [28]) inflating TC performance is also discussed (and discouraged).

Multimodal DL solutions have been recently proposed to face the challenges of mobile-app TC. Aceto et al. [34] propose Mimetic, a general framework for capitalizing the heterogeneous views associated with a traffic object, along with a novel training procedure based on pre-training and fine-tuning. Experimental results show that the Mimetic classifier outperforms single-modal, ML-based, as well as late-combination of traffic classifiers both in terms of TC performance and training complexity. Following along the same research direction, Wang et al. [40] propose *App-Net*, consisting of two modalities: a (bidirectional) LSTM and a 1D-CNN. Experimental results show that App-Net outperforms ML-based and single-modal DL-based traffic classifiers, while performing almost on par w.r.t. Mimetic. In the same research direction, Aceto et al. [41] propose Distiller, a multimodal multitask DL approach for traffic classification to capitalize on the heterogeneity of traffic data and solving multiple traffic categorization problems simultaneously. A specific instance of the proposed framework was experimentally compared with state-of-the-art multitask DL traffic classifiers [33, 38] on the public dataset ISCX VPN-nonVPN, showing Distiller achieves gains over all the TC tasks considered, also exhibiting very manageable training complexity and lower computational burden than the overall-best-performing multitask baseline.

Recently, Akbari et al. [42] have proposed a tripartite multimodal DL architecture based on convolutional and LSTM layers for encrypted-traffic classification at service (HTTPS traffic) and application (QUIC traffic) level. Each modality is fed with different input data, namely *(i)* raw TLS handshake bytes, *(ii)* flow time-series of IAT, size, and direction, and *(iii)* handcrafted (post-mortem) flow statistics. Unfortunately, the authors (*i*) do not compare their proposal with other multimodal solutions, (*ii*) include the adoption of both handcrafted and post-mortem flow statistics, and (*iii*) do not consider Context inputs in the design of the proposal.

2.2.2 User Activity Recognition

While the app-level classification of mobile app traffic is just a—especially harder—case of network-traffic classification, modern apps characterized by *multiple usage modes (activities)* offer, on the one hand, an even harder problem, on the other hand, the possibility of obtaining more actionable information w.r.t. just the sole indication of the app. Therefore, recent academic studies have investigated and demonstrated *the ability to infer user actions* performed in mobile apps by analyzing encrypted network traffic.

Conti et al. [43] propose a framework to **infer which specific actions** the users perform while running a certain **mobile app**, based on packet direction/size information. This is achieved using both supervised (Random Forest, RF) and unsupervised learning (agglomerative clustering with a distance based on dynamic time warping) approaches for service burst classification. It is shown that *by knowing the app generating the traffic*, it is possible to identify a user action with $\geq 95\%$ accuracy for most of the actions considered within a set of 7 Android apps. Saltaformaggio et al. [44] tackle a similar task via their Netscope proposal: the evaluation is carried out considering a set of 35 popular activities (for both Android and iOS devices), based on statistics originated from IP headers. For elementary-behavior discovery, K-means clustering is used, and then a Support Vector Classifier (SVC) is trained/tested on activity-behaviors binary mapping, resulting in performance that varies depending on the device being tested but averages 78.04% precision and 76.04% recall. Grolman et al. [45] extend the feature-extraction method proposed in [43] and employ transfer learning (based on a modified co-training method requiring only a few samples in the source domain) to transfer patterns that allow identifying specific in-app activities (i.e., tweets and posts) across different configurations (i.e., device- or version-wise) to improve the process of recognizing user actions utilizing existing unlabeled encrypted data. The classification of user actions in target configuration is made by using two co-training learners based on RF and AdaBoost algorithms.

Aiolli et al. [46] focus on **identifying user activities** on smartphone-based **Bitcoin wallet apps**, leveraging the earlier methodology proposed in [49]. The fingerprints are collected by *running apps automatically* on Android and iOS devices and *simulating user actions* using scripted com-

mands. Network traces are pre-processed (to remove background traffic and extract features) to train an SVC and an RF. Statistical features are collected on sets of packets defined through timing criteria and IP address/port pairs. The authors deal with the classification problem in a multi-stage hierarchical fashion to infer the app category (Bitcoin vs. other), the OS (Android vs. iOS), the app, and finally the specific action performed by the user. The results, evaluated on 29 apps (9 Bitcoin) and 7 user actions, report 95% accuracy.

Li et al. [47] address the problem of **inferring activities** from a targeted set of mobile apps via analyzing the **continuous encrypted user traffic stream**[1]. The authors propose a DL-based framework in which, focusing on activities having duration > 15 s, they proceed by dividing each traffic stream into segments using a sliding-window approach, where each segment corresponds to an activity of a targeted app. Subsequently, the segments are normalized and represented by a time-space matrix and a traffic spectrum vector that are used to feed 2D-CNN and 1D-CNN branches of a multimodal DL architecture, respectively. The proposed solution is compared with state-of-art classifiers on a *semi-synthetic* traffic dataset and attains $> 95\%$ accuracy regarding app, activity, and both classification tasks.

Finally, Li et al. [48] propose a two-step strategy method for **mobile-service TC**. In detail, in the first step, a joint DL model is exploited as a basic classifier to observe the mobile service traffic from multiple timescales (i.e., micro-time, short-time, and long-time intervals). Specifically, based on the time scale, the basic classifier leverages a different architecture—including Logistic Regression (LR), Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN)—to extract information from the features used to feed the model. In the second step, an attention mechanism is used to aggregate the basic predictions made by the first step to observe the mobile service traffic on an extra-long-time scale. Experimental results show that the two-step strategy outperforms pure DL strategies when used in the classification of 7 different services (e.g., video on demand, video call, live stream, chat), both in terms of accuracy and time delay.

[1] A traffic stream encompasses all outgoing packets from the mobile device without filtering on protocol, destination IP, and ports.

Positioning w.r.t. Related Literature. Although many Deep Learning-based approaches have been proposed and have proven to be very effective in addressing traffic classification, they have two main limitations concerning *(i)* the type of input used to feed models (e.g., raw payload or flow packets sequence), *(ii)* the type of task which they solve (e.g., app or activity classification).

Regarding the type of inputs, all existing methods only consider the traffic of the biflow to be classified, utilizing its payload or feature sequences extracted from the initial packets. However, as we demonstrate experimentally, state-of-the-art methods relying on this input type do not perform well in classifying user activity by observing just the first few packets of the biflow to be classified (aka *early* TC). Classifying network traffic at an early stage, instead of waiting until the end of the communication (aka *post-mortem* TC), provides timely information about the type of traffic being transmitted. This is crucial for faster responses and enables immediate decisions based on traffic characteristics, leading to better optimization of network resources and a more timely response to any security issues. However, we show that defining new input types that can capture the unique characteristics of CC app activities is essential to solving this challenging task.

About the task addressed, these approaches were designed to classify either only mobile apps [18, 35, 36, 40] or to identify non-mobile traffic services [27, 33, 38, 39, 41, 42].

It is worth noting that traffic service is a more general concept than the activity performed by the user, as it does not account for the type of traffic being generated. For instance, some approaches identify VoIP services without considering whether they are associated with audio or video traffic. However, as we demonstrate, these two types of traffic have different characteristics (e.g., bit- and packet-rate), which require different network requirements (e.g., bandwidth). Our proposal takes into account such kind of differences.

On the other hand, it is evident that almost all approaches aimed at a more precise identification of user activity use the service burst as the subject to be classified (viz. *Traffic Object*). As explained in Sec. 4.3, using service burst involves some limitations since it groups all packets with

an inter-packet time shorter than a given threshold and share the same *transport protocol* and *destination IP:port pair*. These include the need to set an appropriate time threshold to separate bursts. Additionally, the concept of burst groups all packets related to the same server application without distinguishing between the biflows composing it, which may correspond to different clients (viz. devices). This implies that a burst does not consider the scenario where an application on a device contacts services hosted on different servers. In contrast, considering only the initial part of a biflow (focusing on a specific source and destination) and its context, our solution does not suffer from the above limitations.

Based on the above findings, we propose a new solution to address the above problems based on an innovative set of inputs (i.e., Context inputs), which also considers the context of the target biflow (i.e., traffic related to coexisting biflows). Then, to capitalize on these inputs, we design and evaluate a multimodal architecture (i.e., Mimetic-All) that combines them with the inputs already used in the literature.

2.3 Internet Traffic Prediction

The scientific community has shown great interest in predicting network traffic evolution. Researchers have approached different prediction tasks related to a variety of distinct practical network problems. Table 2.2 summarizes the main aspects of each work surveyed herein. We categorize each paper based on whether (a) it tackles the prediction of traffic generated by multiple activities associated with the considered application and (b) it uses a multitask model, detailing also (c) the specific prediction techniques employed. Then, we specify (d) if such techniques are designed for fine-grained or coarse-grained traffic prediction, (e) how they are evaluated (i.e., the granularity of the prediction task), and (f) the traffic parameters/quantity predicted. Finally, the last column flags (g) the works publicly releasing the dataset used in their experimentations. The present section ends with the positioning of our work, whose main points are recapped in the last row of Tab. 2.2.

Firstly, we can notice that none of the previous works performs a **per-activity** breakdown of the predictions associated with the considered application.

Table 2.2. Related work tackling various network traffic prediction tasks using different approaches. Reported papers are listed in chronological order. The last row summarizes the current proposal.

Reference	Per-activity	MT	Techniques	Design	CG-eval	FG-eval	Prediction	Data
Dainotti et al. [13]	○	●	HMM	FG	○	●	PS, IAT	●
Katris and Daskalaki [62]	○	○	FARIMA, RBF, MLP	FG *	●	◐	Frame Size, Traffic Volume	●
Oliveira et al. [57]	○	○	MLP, JNN, SAE	CG	●	○	Traffic Volume	●
Tanwir et al. [60]	○	○	HMM, MMG	FG	○	◐	Frame Size	●
Huang et al. [51]	○	○	3D-CNN, 3D-CNN+LSTM, ARIMA, LSTM, MLP	CG	●	○	Data CDR Count	●
Kalampogia and Koutsakis [61]	○	○	LR, MC	FG	○	◐	B-Frame Size	●
Ramakrishnan and Soni [52]	○	○	ARIMA, CV, GRU, LSTM, MA, RNN	CG	●	○	Packet Distribution Traffic Volume	◐
Hua et al. [53]	○	○	ARIMA, LSTM, MLP RCLSTM, SVR	CG	●	○	Traffic Volume	●
Huo et al. [54]	○	○	STL+Seq2Seq-LSTM	CG	●	○	Traffic Volume	●
Zhang et al. [59]	○	○	Seq2Seq+ConvLSTM	CG	●	○	Traffic Volume	●
He et al. [55]	○	○	AVP, DQN, LSTM PMF, ZP	CG	●	○	Dw Bytes	○
Aceto et al. [50]	○	●	HMM, MC, LR, k-NNR, RFR	FG	○	●	PL, IAT, DIR	●
Montieri et al. [23]	○	●	CNN, LSTM, GRU, SeriesNet, DSANet, MC, LR, k-NNR, RFR	FG	○	●	PL, IAT, DIR	●
Saha et al. [56]	○	○	RNN, LSTM, GRU	CG	●	○	Bit-rate	○
Fang et al. [58]	○	○	GNN	CG	●	○	Traffic Volume	○
This Thesis	●	●	CNN, LSTM, GRU, SeriesNet	FG	●	●	PL, IAT, DIR, Number of Packets, Traffic Volume	●

*The solution is not designed for a specific prediction task and is evaluated on video-traffic datasets with different aggregations (from single frame to seconds).
MT: Multitask. **FG**: Fine-Grained. **CG**: Coarse-Grained. **Data**: Publicly Available Dataset. ● present, ◐ partially present, ○ lacking.

Regarding the particular **techniques** exploited for traffic prediction, Tab. 2.2 highlights a rising utilization of DL models, also in a **multitask** configuration [13, 23, 50]. Particularly, related works mostly employ CNN of different dimensions [23, 51], LSTM [23, 51, 52, 53, 54, 55, 56], GRU [23, 52, 56], SAE [57], GNN [58], and hybrid architectures obtained via their combinations [23, 51, 59]. Fewer works leverage Markov models (e.g., MC, HMM, and MMG) [13, 50, 60, 61], traditional ML models (e.g., LR, SVR, k-NNR, or RFR) [23, 50, 53, 61], and statistical techniques (e.g., ARIMA or FARIMA) [51, 52, 53, 62], usually as performance baselines to evaluate DL models, with the latter commonly showing better prediction performance.

Concerning the **design** of traffic predictors, several works propose solutions tailored for forecasting the evolution of traffic aggregates (**CG**), such as traffic volume in a given time interval. On the other hand, fewer proposals are designed to forecast traffic characteristics at a finer level (**FG**), such as the size and the direction of the next packet.

Regardless of the specific design choice, we also highlight the granularity of the prediction task faced to evaluate the proposed solutions.

The works performing coarse-grained evaluation (**CG-eval**) forecast various traffic aggregates, such as bit rates [56], packet distributions [52], and traffic volumes [52, 53, 54, 57, 58, 59, 62] at different time resolutions, ranging from less than one second to a few seconds, minutes, hours, and even days. Among the latter, some works take into account also the geographical or topological distribution of sources and destinations, rather than leveraging the (sole) temporal information via traffic matrices [53] or considering the geographic distribution of data volumes (e.g., aggregated data calls) as observed at base stations [51, 58, 59].

Differently, the proposals performing fine-grained evaluation (**FG-eval**) can capitalize on different sources of information. Some works tackle packet-level traffic prediction relying entirely on network-layer features available also in case of encryption (●), such as packet sizes, directions, and inter-arrival times [13, 23, 50]. Other prediction approaches consider video traffic and forecast video frame properties exploiting application-layer information, which is not always available when encryption strategies are enforced [60, 61, 62]. We underline this shortcoming by partially flagging (◐) the FG-eval column in Tab. 2.2.

Positioning w.r.t. Related Literature. This work aims to predict the traffic generated by some of the most popular and used CC apps at the packet level (i.e., our proposal is designed to fulfill fine-grained traffic prediction). Nevertheless, we exploit these predictions to also forecast traffic aggregates in terms of the number of packets and volume. Thus, *we perform both fine-grained and coarse-grained traffic prediction* exploiting only network-layer features differently than all the works reported in Tab. 2.2. Indeed, all of them exclusively focus on one of these tasks, namely either CG-eval or FG-eval. The sole exception is the work of [62], which, however, is specifically tailored for the prediction of video-frame sizes and leverages application-layer information unavailable in case of encryption.

Regarding the works exploiting multitask models, in [13], the focus is on the preliminary aspects of traffic modeling of unidirectional flows—thus neglecting the advantage of considering request-response interaction—of non-mobile-app traffic by means of HMM. Similarly, in [50], HMM and MC models are investigated to characterize and predict network traffic generated by mobile apps without exploiting the advantages of multitask DL models.

Conversely, [23] is the most related work as it addresses the same prediction problem (i.e., network-traffic prediction at packet level via DL approaches) investigated herein. However, differently than [23], we focus on traffic generated by *CC apps*, which have become extremely popular with the COVID pandemic. Specifically, we investigate the suitability of DL solutions to predict their fine-grained characteristics. Moreover, by leveraging a more detailed ground truth that includes the specific activity performed by the users (beyond the app generating the traffic), we investigate the impact of *multiple activities* on the capability of the DL model to predict traffic characteristics. Finally, we assess the suitability of considering a single model or a specific model for each transport-level protocol to predict CC-app traffic against per-app models.

2.4 AI Trustworthiness in Traffic Analysis

Recently, researchers have applied XAI techniques to improve the performance, robustness, reliability, and feasibility of AI models that tackle networking-related tasks (also including wired networks) [24, 63]. Table 2.3

Table 2.3. Related work employing XAI when facing various networking problems. Reported papers are listed in chronological order. The last row summarizes the current proposal.

Reference	Networking Problem	Interpretability	Reliability	XAI Method
Amarasinghe et al. [64]	Anomaly Detection	●	○	LRP
Dethise et al. [65]	Video bit-rate Prediction	●	○	LIME
Morichetta et al. [66]	Video Quality Prediction	●	○	LIME
Rezaei et al. [67]	Traffic Classification	●	○	Occlusion Analysis
Beliard et al. [68]	Traffic Classification	●	○	t-SNE Feature Map Visualization
Wang et al. [69]	Traffic Classification	●	○	Deep SHAP
Aceto et al. [41]	Traffic Classification	○	●	Calibration Analysis
Aceto et al. [50]	Packet-level Traffic Prediction	◐	○	Markovian-Distillation
Akbari et al. [70]	Traffic Classification	●	○	Occlusion Analysis
Montieri et al. [23]	Packet-level Traffic Prediction	◐	○	Markovian-Distillation
Nascita et al. [71]	Traffic Classification	●	●	Deep SHAP, Calibration Analysis
Sadeghzadeh et al. [72]	Traffic Classification	●	○	Perturbation Analysis
Fauvel et al. [73]	Traffic Classification	●	○	Explainable-by-Design DL Architecture
This Thesis	Traffic Classification, Packet-level Traffic Prediction	●	●	Deep SHAP, Calibration Analysis

● present, ◐ partially present, ○ lacking.

reports the works tackling various networking problems by means of different XAI methods, focusing on the aim of the trustworthiness analysis conducted. Specifically, we flag the works that aim to (partially) interpret their forecasting results and/or measure to which extent the confidence associated with the latter can be deemed reliable (i.e., high/low confidence leads to high/low accuracy in prediction). The last row of Tab. 2.3 summarizes the present work, whose positioning w.r.t. related work is discussed at the end of this section.

Referring to the **networking problems** addressed in the light of AI trustworthiness, several papers face anomaly detection [64] or traffic classification [41, 67, 68, 69, 70, 71, 72, 73]. Networking-related prediction tasks with different facets are also tackled: video bit-rate adaptation based on reinforcement learning [65], video quality prediction via clustering [66], and packet-level traffic prediction (i.e. the same problem tackled in the present work) via Markovian and DL approaches [23, 50].

We can notice that most of the works apply **interpretability** techniques to provide explanations of the decisions taken.

More specifically, commonly used **XAI methods** provide various forms of *post-hoc explanations* [71]: *(a) layer-wise relevance propagation*(viz. *LRP*) [64], supplies explanations using an iterative approach that leverages the layered structure of a neural network to assign relevance scores to input features. This is achieved by systematically propagating relevance backward from the output layer to the input layer; *(b) interpretable local surrogates* via *LIME* [65, 66], involves approximating the behavior of a black-box model around a specific instance. This is achieved by generating perturbed samples and observing their corresponding predictions. LIME fits a straightforward, interpretable model (e.g., linear regression) to these perturbed samples. This creates a clear representation of the local decision boundary, which helps users to understand the impact of changes in input features on the output provided by the model for a specific prediction, thereby improving the overall interpretability of the model; *(c)* different types of *perturbation analyses*, such as occlusion analysis [67, 70] or universal perturbation attacks [72]. These methods involve training a model with the complete set of inputs and then iteratively assessing its performance by hiding or removing parts of the inputs. The significance of these occluded inputs is determined by evaluating the corresponding variations

in the performance of the model. *(d)* importance attribution based on *Shapley values*, either local [40] or global [71]. This method uses cooperative game theory principles to fairly attribute the contribution of each feature to a specific prediction. It accounts for all possible combinations of features and evaluates how their inclusion or exclusion affects the output of the model. The Shap values, which represent the average contribution of each feature, are based on Shapley values.

Other XAI methods based on *visual representations* (e.g., t-SNE and Feature Maps) [68] inspect the activation of intermediate neurons to highlight the most important features that led to the decision. Moreover, *Markovian Distillation* is applied to interpret traffic-prediction results by comparing Markov Chains' transition probabilities and DL predictions [23, 50]. Going further, solutions aiming at *explainability-by-design* compare parts of input data with per-class prototypes [73].

Finally, the **reliability** of DL models is investigated via a *calibration analysis* [41, 71] of their probabilistic outputs that aims to determine whether the confidence associated with the final decision reflects its reliability and possibly improve it.

Positioning w.r.t. Related Literature. In this study, *we address the inherent lack of trustworthiness of DL models for app and activity traffic classification and packet-level traffic prediction.*

Firstly, we offer *interpretability* through XAI techniques. Our approach involves conducting a comprehensive analysis of DL models using Deep SHAP[2] to provide interpretable results starting from the methodology proposed by Nascita et al. [71]. Regarding the TC task, we adapted such methodology to interpret a newly designed model (viz. Mimetic-All) that classifies both app and activity and exploits an additional view (viz. modality) on traffic based on Context inputs, proposed within this thesis. Moreover, regarding the TP task, we utilized these XAI techniques to investigate the interpretability of DL models for network traffic prediction at the packet-level. To the best of our knowledge, we were the first to face such a problem irrespective of the granularity of prediction or the specific DL model employed. Indeed, our methodology overcomes the limitations of preliminary interpretability solutions exploited in previous works fac-

[2]See Sec. 7.1 for specific details on Deep SHAP.

ing packet-level traffic prediction [23, 50] based on Markovian distillation, which is constrained by memory limitations.

Moreover, we investigate the *reliability* of the resulting DL models for both classification and prediction tasks via a calibration analysis. Again, to the best of our knowledge, no other work dealing with network traffic prediction has investigated such an aspect.

Chapter 3

MIRAGE-COVID-CCMA DATASET

In the field of Internet traffic analysis, researchers often face the challenge of limited availability of high-quality data, which is considered to be one of the major factors that currently hinder the full utilization of data-driven methodologies in network research [74].

Data quality plays a key role in forming effective models, influencing their ability to generalize, accuracy, and interpretability. Such models learn from the data presented to them during the training process. If the data is of poor quality, with noise or incorrect information, the model will learn from these errors, compromising its ability to generalize to new data. In addition, poor-quality data can lead to models that provide inaccurate or imprecise predictions that make them unreliable and, therefore, unsuitable in real-world scenarios, especially in critical or sensitive applications.

Based on these considerations, it becomes crucial to have quality data that are both reliable and representative of reality. Indeed, to obtain effective models in a real-world context, one needs real—and, if possible, human-generated (i.e., which considers the possible patterns of user usage) traffic data. Simulated data may not fully capture the complexity and variability of real-world network traffic, especially in scenarios where traffic varies over time. Because of this inability, models based on such data may not guarantee the expected behavior when applied in real-world contexts. However, overcoming this challenge is becoming increasingly difficult, espe-

cially when dealing with mobile network traffic, due to the widespread use of mobile devices and available applications. Mobile apps pose a particular challenge as their traffic is encrypted, dynamic, and heterogeneous, with different users, devices, and operating systems. Additionally, preserving user privacy further limits the collection and release of this type of data. Moreover, the complexity of mobile apps and their variety of services require more granular information to be effective. For instance, in the case of CC apps, it is crucial to consider the particular type of service or activity performed by the user, as they have different network characteristics and requirements.

Last, the public availability of data is crucial to support scientific research and make results easy to reproduce, comparable, and more generalizable [75].

As a result of the considerations above, in this chapter, we introduce the MIRAGE-COVID-CCMA-2022 dataset we have carefully collected and made available to the public. This dataset comprises real, human-generated traffic data accurately labeled and focuses on the most widely used CC apps. This dataset serves as the foundation for the research conducted in this thesis. In detail, we describe the MIRAGE architecture used for traffic collection (Sec.3.1). We then detail the *crowdsourced* collection campaign, which resulted in the MIRAGE-COVID-CCMA-2022 dataset (Sec.3.2).

3.1 MIRAGE framework

This section starts by providing information about the optimized MIRAGE architecture (Sec. 3.1.1), which is designed to capture the traffic generated by CC apps. Then, we describe the procedure that led to the MIRAGE-COVID-CCMA-2022 dataset (Sec. 3.1.2), which includes capturing the traffic and creating ground-truth data. Finally, we describe the release format of the dataset (Sec. 3.1.3).

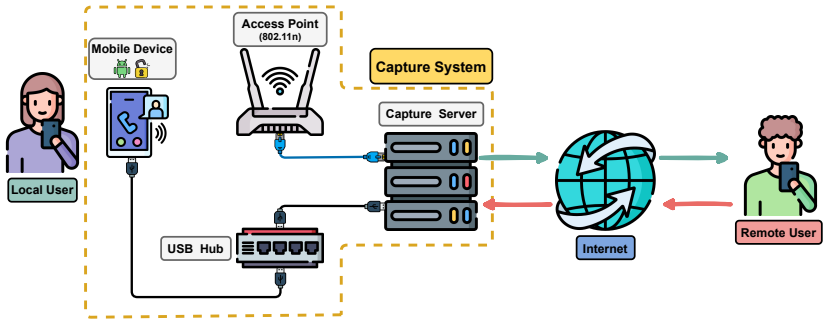


Figure 3.1. Diagram of the *Capture System* of the MIRAGE architecture[1].

3.1.1 Mirage Architecture

Starting from the MIRAGE architecture originally proposed in [76] Fig. 3.1 depicts its *Capture System* that has been appropriately re-designed to collect traffic generated by CC apps. The *capture server* (🖴) plays a crucial role in our setup. It is a workstation equipped with an IEEE 802.11 *access point* (📶), providing connectivity to the *Android mobile device* (🤖), generating traffic when the experimenter uses the apps. The capture server uses a wired connection to connect to the public internet that performs *Network Address Translation* (NAT). Each mobile device is physically attached to the capture server via a *USB hub*. This connection allows us to leverage the *Android Debug Bridge* (viz. ADB) and send commands from the server to the attached device while receiving responses on an off-band channel. It is worth noting that the device must be rooted (🔓) to run traffic capture successfully and that the system can capture traffic from multiple devices simultaneously. Additionally, to fulfill the networking requirements of CC apps, particularly for video or audio calls, we replaced the original IEEE 802.11g [77] access point with one that supports the 802.11n [78] standard. This upgrade was necessary to achieve better network performance and stability. Although it performs less well than newer standards such as 802.11ac [79] and 802.11ax [80], 802.11n significantly improves over 802.11g, providing higher bandwidth, greater connection stability, wider coverage, and more frequency options.

[1]This diagram has been designed using images from *Flaticon.com*

3.1.2 Dataset Building Procedure

The architecture described in Sec. 3.1.1 is leveraged to build the dataset by performing a *collection campaign* in which several capture sessions are executed. During each session, the capture system is used to capture the traffic generated by a specific app (viz. *target*) that is being used by a (*local*) user on a specific device (identified by its *MAC address*). To limit possible background traffic, network access is disabled for all apps except for the target one. Each session results in a PCAP traffic trace and log files used for ground-truth building. To begin a capture session, an experimenter (viz. local user in Fig. 3.1) connects a mobile device to the USB hub. This action automatically starts capturing the network traffic and logs-files with ground-truth information. The network traffic is captured through the wired interface of the capture server using the `tcpdump` tool [81]. In particular, traffic generated by multiple devices can be collected without ambiguity by utilizing traffic filters based on the MAC address of connected devices. In addition, the log files associate each socket descriptor, which includes *<IP:port>* pairs, with the *Android package name* that initiates the call is monitored through the `netstat` tool [82]. This mapping provides accurate metadata and enables reliable labeling of each biflow with the corresponding Android package name that matches the *quintuple* in the log file, considering established network connections. When the log file does not contain an *exact* matching for some biflows, the procedure resorts to a heuristic approach to assign labels. This involves assigning the *most-common* package name in the PCAP trace to these biflows. However, there is a possibility of mislabeling due to this approach. Therefore, the dataset explicitly marks instances where labeling was performed using this heuristic.

Furthermore, it is worth noting that in the specific case of CC apps, the experimenter uses the target app of a capture session to perform a specific activity. Accordingly, activity labels are manually assigned based on the knowledge of the individual activity performed by the user during the specific capture.

Table 3.1. The dataset is released by providing a JSON file for each captured trace (i.e., the resulting PCAP of a self-contained capture session). For each biflow (identified by its quintuple), the JSON file provides information at different granularities: (*i*) *Per-packet data*, extracted from all packets of each biflow; (*ii*) *Per-flow features*, extracted on the sets of upstream, downstream, and complete IP packet lengths and inter-arrival times; (*iii*) *Per-flow metadata*, related to the complete biflow (BF) and related upstream (UF) and downstream (DF) flows.

Data	Name	Description
Per-packet data	timestamp	Timestamp expressed as Unix Epoch time
	src_port	Source transport-layer port
	dst_port	Destination transport-layer port
	packet_dir	Packet direction (0 upstream, 1 downstream)
	IP_packet_bytes	Number of bytes in IP payload
	IP_header_bytes	Number of bytes in IP header
	L4_header_bytes	Number of bytes in L4 header
	L4_payload_bytes	Number of bytes in L4 payload
	iat	Inter-arrival time
	TCP_win_size	TCP window size (0 for UDP packets)
	TCP_flags	TCP flags (empty for UDP packets)
	L4_raw_payload	Byte-wise raw L4 payload (integer $\in [0, 255]$)
Per-flow features	min	Minimum
	max	Maximum
	mean	Arithmetic mean
	std	Standard deviation
	var	Variance
	mad	Mean absolute deviation
	skew	Unbiased sample skewness
	kurtosis	Unbiased Fisher kurtosis
	q_percentile	q^{th} percentile ($q \in [10 : 10 : 90]$)
Per-flow Metadata	BF_device	MAC address of the mobile device
	BF_label	Android-package name
	BF_activity	Activity type
	BF_app_category	Android-package category name ‡
	BF_label_version_code	Android-package version code
	BF_label_version_name	Android-package version name
	BF_labeling_type	Exact or most-common labeling
	{BF,UF,DF}_num_packets	Number of packets
	{BF,UF,DF}_IP_packet_bytes	Total bytes in IP packets
	{BF,UF,DF}_L4_payload_bytes	Total bytes in L4 payloads
	{BF,UF,DF}_duration	(Bi)flow duration in seconds
	{UF,DF}_MSS	TCP Maximum Segment Size (0 for UDP flows)
	{UF,DF}_WS	TCP Window Scale factor (0 for UDP flows)

‡ Category provided by the Google Play Store.

3.1.3 Dataset Release Format

At the end of the collection campaign, the dataset is obtained by processing all the PCAPs and enriching the extracted information with the ground truth.

The data obtained from these operations results in a JSON file for each PCAP, containing information at the biflow level (identified by its quintuple). Tab. 3.1 provides an overview of the information extracted for each biflow, which is categorized into three groups:

- *per-packet data*: refers to 12 informative header fields and the L4 payload from all packets of each biflow, where each identifies a list with a length equal to the number of packets in the biflow.
- *per-flow features*: pertains to data gathered from the entire biflow along with its upstream and downstream flows. This results in 17 statistical features being computed on the sets of upstream, downstream, and complete (i.e., without distinguishing by direction) IP packet lengths and inter-arrival times, accounting for 102 features.
- *per-flow metadata*: is also linked to complete biflow and upstream/-downstream flows and provides ground-truth information at different granularities (e.g., at the app and activity level) obtained through the building procedure described earlier.

3.2 Dataset Collection

In this section, we first explain the rationale behind the selection of the CC apps, and then provide details about the collection campaign that led to the final MIRAGE-COVID-CCMA-2022 dataset.

3.2.1 Apps' and Activities' Selection Rationale

CC apps—used for business meetings, classes, and social interaction—have experienced a huge utilization increment when "stay-at-home" orders were issued worldwide and are still widely used due to the change in life- and work-style of people around the globe. The wide adoption of these apps exposing complex network behaviors has prompted recent research

from academics and practitioners focusing on different facets, ranging from reaction to varying network conditions [83] to the different usages of RTP/RTCP protocols [84] for the multimedia transfers.

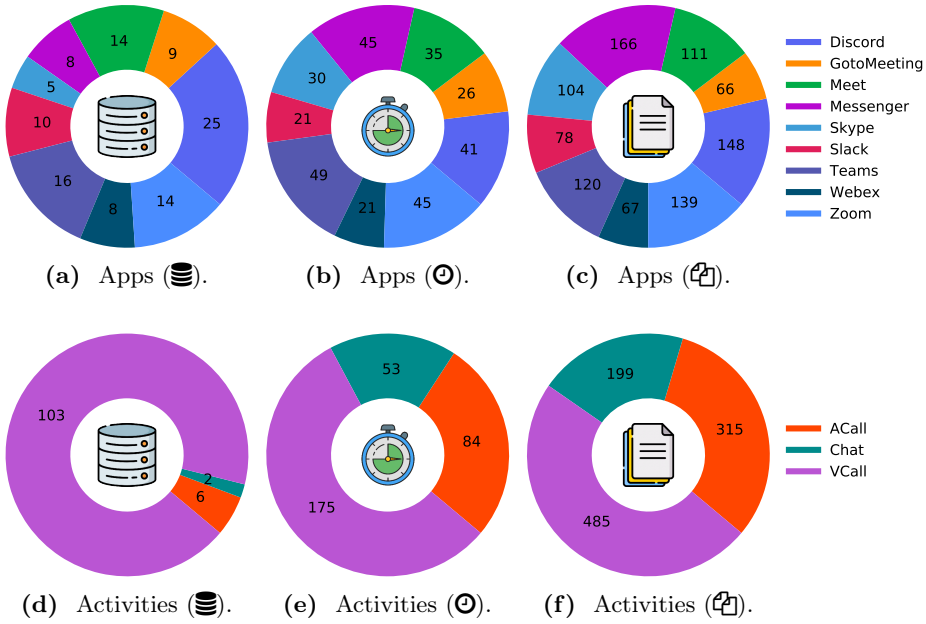
Following popularity and utilization boost, we focus on *nine* CC apps: `Discord`, `GotoMeeting`, `Meet`, `Messenger`, `Skype`, `Slack`, `Teams`, `Webex`, and `Zoom`. Indeed, `Zoom` has obtained the steepest increment with its traffic scaling by orders of magnitude, followed by `Webex`, `GotoMeeting`, `Teams`, `BlueJeans`, and `Skype` [85]. Also, during 15th–21st March 2020, `Zoom` was downloaded 14×, 20×, and 55× more than the weekly average during Q4 2019 in the US, UK, and Italy, respectively [86]. Similarly, `Teams` also experienced significant growth in Italy (resp. France) with 30× (resp. 16×) more downloads. The considered apps have been extensively exploited for remote (and blended) teaching in Italian [87] and European [88] institutions and universities.

As also emphasized by Sandvine [1, 89], in the following years, video traffic demonstrated increasing significance compared to the preceding year (e.g., a +12% volume growth in 2022 compared to 2021). This applies to video traffic both independently and as an integrated element within app mashups, corresponding to ≈ 66% of the overall global traffic volume in 2022. Indeed, social and communication applications have gained further popularity, with `WhatsApp`, `Zoom`, `Teams`, and `Messenger` being the most used for *messaging*, and `Zoom`, `Webex`, and `Teams` for *enterprise conferencing*.

3.2.2 Crowdsourced Collection Campaign

The dataset MIRAGE-COVID-CCMA-2022 was collected through a crowdsourced campaign that involved over 180 volunteers, including students and researchers, who acted as experimenters[2]. The campaign took place between April and December 2021 and was conducted leveraging the Mirage architecture in the ARCLAB laboratory of the University of Naples "Federico II". The traffic was captured using network infrastructure provided by *Consortium GARR*, the Italian national network of University and Research, accessed through the above laboratory, with a maximum

[2]The privacy of the volunteers was preserved by using fictional accounts created specifically to make the captures.



(a) Apps (🗄). **(b)** Apps (🕐). **(c)** Apps (🗍). **(d)** Activities (🗄). **(e)** Activities (🕐). **(f)** Activities (🗍).

Figure 3.2. Summary of the *CrowdSourced Collection* that led to the MIRAGE-COVID-CCMA-2022 dataset. 🗄 refers to the amount of traffic data (GBytes, GB). 🕐 refers to the capture time (hours, h). 🗍 refers to the number of capture sessions.

available bandwidth of 100Mbps. We highlight that the captures were carried out by adhering to the distancing/mask-wearing rules prescribed by regional/national decrees in force at the moment of the collection.

Based on Subsection 3.2.1, we conducted a campaign to gather traffic data from nine popular CC apps. These CC apps were selected based on their popularity and increased usage. Moreover, taking advantage of their multi-activity nature, each of these apps was used to perform three different types of activities all related to live events: *Chat ("Chat")*—involves just two participants exchanging textual messages and/or multimedia content (e.g., images or GIFs); *Audio-call ("ACall")*—involves just two participants transmitting only audio; *Video-call ("VCall")*—involves many attendees who can transmit both video and audio (e.g., live events such as video

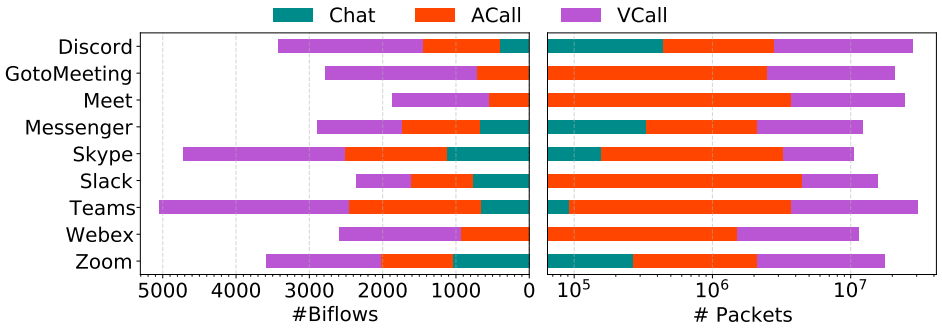


Figure 3.3. Communication and collaboration apps in MIRAGE-COVID-CCMA-2022 (alphabetic order). Performed activities, the number of biflows (*left*-bar) and number of packets (*right*-bar) are reported for each app. Note that the log scale is used to report the packets number.

calls between two or more attendees or webinars). This allowed us to diversify the traffic collected according to the app used and the specific activity performed by the user.

During the campaign, experimenters used three *different* mobile devices: a Google Nexus 6 and two Samsung Galaxy A5. All devices were equipped with the custom firmware *LineageOS* 17.1 [90], corresponding to Android 10. In each capture session, the experimenters performed a *specific activity* on a *given CC apps* to obtain a traffic dataset that reflects the common usage of the considered apps. Each traffic capture session has been performed with the up-to-date version of the app, and its duration spanned from 15 to 80 minutes based on the specific activity being carried out. On one hand, Fig. 3.2 presents an overview of the resulting MIRAGE-COVID-CCMA-2022 dataset, including the amount of (*i*) traffic data, (*ii*) capture time, and (*iii*) capture sessions, which are broken down by CC apps and activities. On the other hand, Fig. 3.3 depicts the mobile apps collected in MIRAGE-COVID-CCMA-2022 and used in this study, also highlighting the activities carried out with each and the amount of traffic collected in terms of the number of biflows and packets.

Finally, to foster replicability and reproducibility we *publicly release* the MIRAGE-COVID-CCMA-2022 dataset[3]. In detail, we release the

[3] http://traffic.comics.unina.it/mirage/mirage-covid-2022

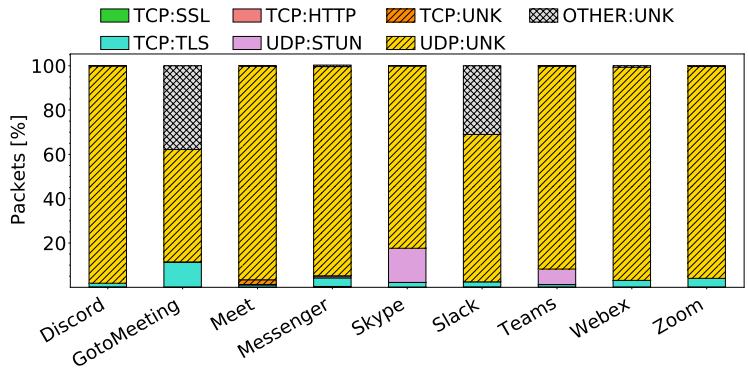
dataset in two formats, making available both the raw traffic data captured (in *JSON* format) and a pre-processed version providing the set of inputs leveraged in this specific thesis to address the TC task (in *pickle* format).

Chapter 4

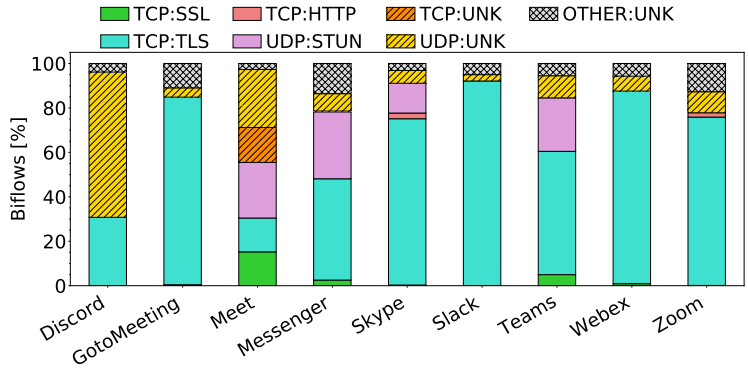
Characterization and modeling of communication and collaboration app traffic

In this chapter, we provide a detailed analysis of the peculiar characteristics of CC apps in MIRAGE-COVID-CCMA-2022 dataset based on the allowed user activities. In Sec. 4.1, we analyze app traffic based on their adopted protocols. In Sec. 4.2, we analyze the discriminability of activities related to specific apps by observing their typical behavior in the initial part of communication based on the inputs commonly used to feed DL-based traffic classifiers. Then, we highlight the peculiarities of the activity by analyzing the temporal evolution of the app connections (viz. biflows) that are established simultaneously (Sec. 4.3) and characterizing the traffic carried by simultaneous biflows (Sec. 4.4), in terms of aggregate metrics, namely bit- and packet-rate. Finally, in Sec. 4.5, we bring out the peculiar characteristics of each app and the related activities by modeling the sequence of packet sizes and directions with Multimodal Markov Chains.

The provided analysis can be useful for different network-related tasks, supporting a detailed understanding of the network traffic. This is crucial for properly managing the network, as well as for identifying unique characteristics (e.g., network fingerprints) of both apps and specific activities. These peculiarities can be leveraged for modeling purposes to support traf-



(a) Percentage of Packets.



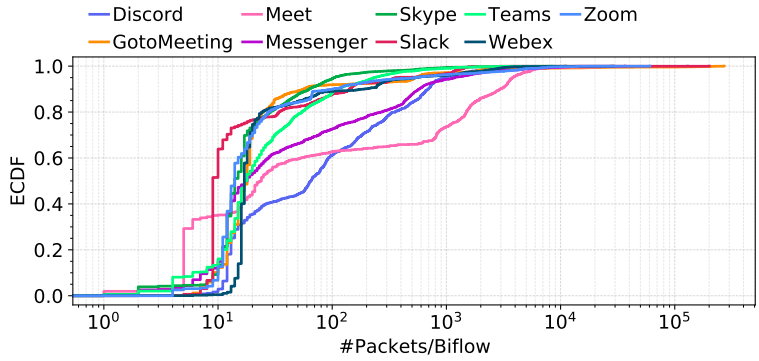
(b) Percentage of Biflows.

Figure 4.1. Protocol distribution in terms of packets (a) and biflows (b). *UNK* stands for *unknown*, SSL stands for *undetected* version of *SSL/TLS*.

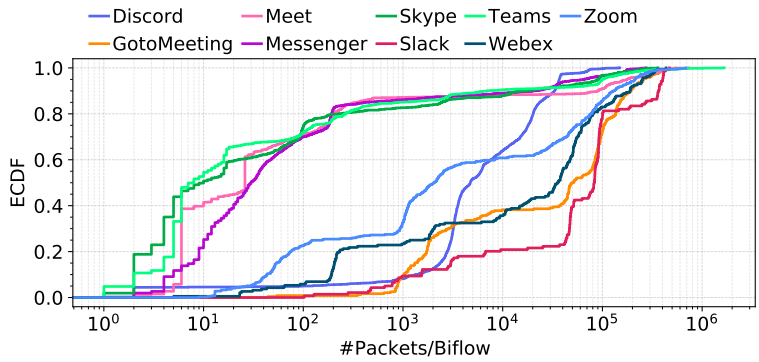
fic classification and prediction tasks.

4.1 Characterization of Adopted Protocols

Herein, we analyze the characteristics of MIRAGE-COVID-CCMA-2022 traffic in terms of the adopted protocols for each app. In practice, CC apps combine multiple protocols to balance reliable data delivery and low-latency transmission. This is necessary to implement control and data



(a) TCP.



(b) UDP.

Figure 4.2. Biflow-length distribution (in terms of the number of packets) for each app for TCP (a) and UDP (b) protocols. Values on the x-axis are reported in log scale.

plane functionalities essential for communication and collaboration scenarios. As a result, Fig. 4.1 depicts the percentage of packets and biflows related to the adopted protocols for the traffic used in this study. In particular, we can observe that all the apps tend to generate significantly more TCP biflows (Fig. 4.1b) ranging from ≈31% (`Discord`) to ≈92% (`Slack`) of the total number of biflows. More in detail, for all the apps, a significant percentage of SSL/TLS biflows is observed (ranging from 30% for `Discord` to 90% for `Slack`). Moreover, for `Meet`, `Messenger`, `Skype`, and

`Teams` there is also a significant presence of STUN[1] biflows (between 10% and 30%), commonly used for multimedia communications in the prevalent case of the presence of a Network Address Translator (NAT). These findings are consistent with the outcomes of traffic analysis performed in other studies [83, 84].

On the other hand, although the share of UDP biflows is significantly lower than the TCP ones, UDP packets correspond to at least 82% of the packets for almost all apps (Fig. 4.1a). Interestingly, unlike `Meet` and `Messenger`, `Skype` and `Teams` have a significant share of STUN packets, corresponding to $\approx 15\%$ and $\approx 7\%$, respectively.

Nevertheless, recent reports [1] indicate a rising trend in the utilization of the QUIC protocol among applications offering communication and video conferencing services. The shift towards QUIC is mainly due to its advantages, such as reduced latency, built-in multiplexing, and better packet loss management. Moreover, QUIC provides end-to-end encryption by default, improving security, and it is designed to manage mobile connections effectively. These features are prompting many platforms to adopt the QUIC protocol to enhance overall performance and provide a better user experience. As a result, the QUIC protocol is becoming the preferred choice to replace TLS over TCP, which is the standard for reliable and secure internet communication. Therefore, an interesting follow-up to the analysis shown in Fig. 4.1 could be to assess the degree to which the QUIC protocol has been adopted by the CC apps.

To deepen the above characterization, Fig. 4.2 reports for each app the distribution of the number of packets per biflow, according to the transport-layer protocol (i.e., TCP and UDP). On the one hand, regarding the TCP protocol (Fig. 4.2a), we observe that $\approx 90\%$ of the biflows have at most 100 packets for almost all apps. Conversely, this number decreases to $\approx 60\%$ and $\approx 70\%$ in the case of `Discord` (`Meet`) and `Messenger`, respectively. Also, we observe that 20% of biflows have less than ≈ 10 packets for almost all apps. Interestingly, the above share for `Meet` and `Slack` reaches $\approx 35\%$ and $\approx 65\%$, respectively. On the other hand, when examining the UDP protocol in detail (ref. Fig. 4.2b), the observed behavior is more closely linked to the specific application being used. In

[1]Session Traversal Utilities for NAT (STUN) protocol allows an end-point to determine the IP address and port assigned to it by a NAT [91]

the case of `Meet`, `Messenger`, `Skype`, and `Teams`, ≈30% of biflows consists of more than 100 packets. Interestingly, for `Skype` and `Teams`, ≈ 50% of the remaining biflows have no more than 6 packets, while for `Meet` it is ≈ 40%. Finally, among these apps, more than 90% of biflows in the case of `Discord`, `GotoMeeting`, and `Slack` have more than 1000 packets.

Takeaways

RQ: *What is the share of protocols used by CC apps in terms of biflows and packets?*

CC apps employ a mix of protocols, primarily relying on TCP biflows, ranging from approximately 31% to 92%. SSL/TLS biflows are prevalent while STUN biflows are notable in some apps like `Skype` and `Teams`. Although UDP biflows constitute a smaller share, they represent at least 82% of packets for most apps;

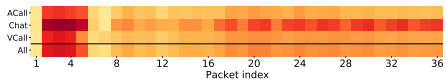
Based on transport-layer protocols, the distribution of packets per biflow further highlights specific app behaviors, with TCP biflows typically having fewer than 100 packets. In contrast, UDP biflows vary based on the app: for some apps (e.g., `Skype` and `Teams`), 70% of the biflows have less than 100 packets, while for others (e.g., `GotoMeeting` and `Slack`) 90% have more than 1000 packets.

4.2 Biflow-level Characterization of Early Behavior

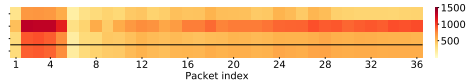
In this section, we analyze the typical behavior of biflows in the initial part of the communication to understand if, by observing the inputs commonly used as input to DL traffic classifiers, it is possible to discriminate the activities related to a specific app.

Accordingly, we analyze the sequence of *packet direction* (`DIR`), *transport-level payload length* (`PL`), *inter-arrival time* (`IAT`), and *TCP-window size* (`TCPWIN`) across the first 36 packets of each biflow[2] . Similarly, we analyze

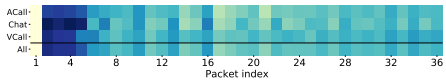
[2]Packets with no payload are discarded since they reflect transport-layer signaling



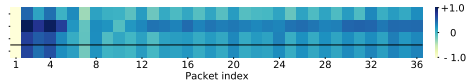
(a) `Skype`- Avg. `PL` [B] of the first 36 packets.



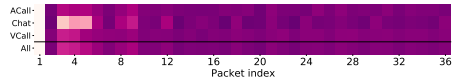
(b) `Teams`- Avg. `PL` [B] of the first 36 packets.



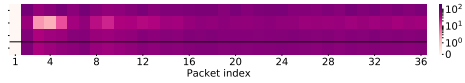
(c) `Skype`- Avg. `DIR` of the first 36 packets.



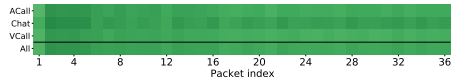
(d) `Teams`- Avg. `DIR` of the first 36 packets.



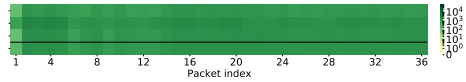
(e) `Skype`- Avg. `IAT` [ms] of the first 36 packets.



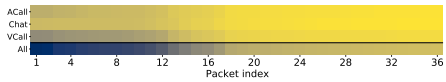
(f) `Teams`- Avg. `IAT` [ms] of the first 36 packets.



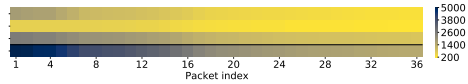
(g) `Skype`- Avg. `TCPWIN` of the first 36 packets.



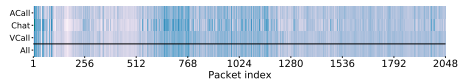
(h) `Teams`- Avg. `TCPWIN` of the first 36 packets.



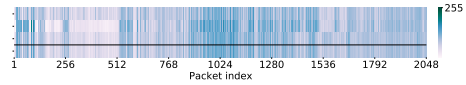
(i) `Skype`- Biflow count on the first 36 packets.



(j) `Teams`- Biflow count on the first 36 packets.

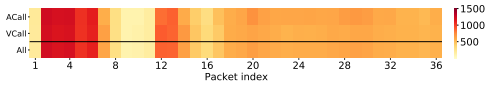


(k) `Skype`- Avg. `PAY` of the first 2048 Bytes.

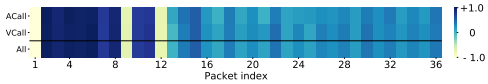


(l) `Teams`- Avg. `PAY` of the first 2048 Bytes.

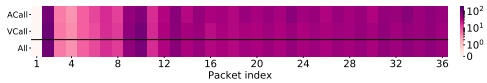
Figure 4.3. Properties of biflows' time-series with respect to `PL`, `DIR`, `IAT`, `TCPWIN`, and `PAY` for `Skype` (a, c, e, g, k), `Teams` (b, d, f, h, l), and `Webex` (m, n, o, p, r) based on the *activity*-type and in summary (*All*) form. For each packet index, is also showed the number of biflows used to calculate the corresponding mean value for `Skype` (i), `Teams` (j), and `Zoom` (q). The downstream and upstream `DIR` is mapped on $+1$ and -1, respectively.



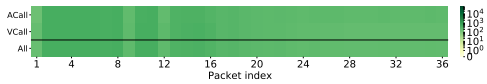
(m) `Webex`- Avg. `PL` [B] of the first 36 packets.



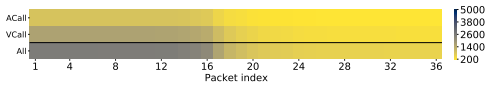
(n) `Webex`- Avg. `DIR` of the first 36 packets.



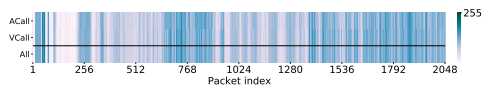
(o) `Webex`- Avg. `IAT` [ms] of the first 36 packets.



(p) `Webex`- Avg. `TCPWIN` of the first 36 packets.



(q) `Webex`- Biflow count on the first 36 packets.



(r) `Webex`- Avg. `PAY` of the first 2048 Bytes.

Figure 4.3. Properties of biflows' time-series with respect to `PL`, `DIR`, `IAT`, `TCPWIN`, and `PAY` for `Skype` (a, c, e, g, k), `Teams` (b, d, f, h, l), and `Webex` (m, n, o, p, r) based on the *activity*-type and in summary (*All*) form. For each packet index, is also showed the number of biflows used to calculate the corresponding mean value for `Skype` (i), `Teams` (j), and `Zoom` (q). The downstream and upstream `DIR` is mapped on $+1$ and -1, respectively.

the first 2048 B of the transport-level payload . For each packet/byte index, we report the average value across all biflows for a given app. The analysis presented in Fig. 4.3 focuses on `Skype`[3], `Teams`[4], and `Webex`[5]. For these apps, the above information is broken down into the considered activities (`ACall`, `Chat`, and `VCall`) and also reported in summary form (*All*). For completeness, we also show the number of biflows used to calculate the corresponding average value for each packet index.

Considering the generic behavior of the specific app (*All*), in all cases, there are significant differences between the initial part—which extends at most up to the 15^{th} packet—and the remaining part of the sequence, when considering `PL`/`DIR`/`IAT`. This does not occur in the case of `TCPWIN`, for which the behavior is slightly less evident. Specifically, in the cases of `Skype` and `Teams`, the main patterns are located on the first 5 packets, while for `Webex` the trend extends up to the 15^{th} packet.

Moreover, when focusing on the specific app, similarities between the patterns associated with the different activities can be seen. For `Skype` and `Teams` we observe characteristic patterns associated with `ACall` and `VCall` on the first 5 packets, while for `Chat` we observe a trend that extends over all 36 packets. Specifically, focusing on `VCall` and `ACall`, if we consider `PL` and `DIR`, in the case of `Skype` we observe identical behavior between the two activities (excluding the 5^{th} packet). In contrast, in the case of `Teams`, the pattern associated with `VCall` is slightly more pronounced (i.e., packets with higher payload in the downstream direction). At the same time, this does not hold when looking at the `IAT` for which we observe longer times in the case of `ACall`. It is also interesting to note that in the case of `Webex`, `ACall` and `VCall` present an identical behavior, and this is probably because `Webex`[6], unlike the other two shown, is an app designed to allow users to make video calls, and this function has been adapted to make simple audio calls since it does not have a native function to perform this type of activity. Finally, referring to `PAY`, similar observations can be made about the indistinguishability between the activities associated with `Skype`, `Teams`, and `Webex`, especially regarding the `ACall` and `VCall`.

neither depending on the nature of the app nor the performed activity.

[3] A similar behavior as `Skype` has been observed for `Messenger`, `Slack`, and `Zoom`.

[4] A similar behavior as `Teams` has been observed for `Discord`.

[5] A similar behavior as `Webex` has been observed for `GotoMeeting` and `Meet`.

[6] The same reasoning applies to `Meet` and `GotoMeeting`.

Takeaways

RQ: *Is it possible to differentiate between the activities of a CC app by only looking at the initial part of its biflows?*

The activities performed by a user within a particular app tend to show similar behavior patterns when looking at individual biflows. This similarity becomes evident when examining specific features that are typically used to feed DL traffic classifiers, such as transport-layer payload (viz. `PAY`) or the per-packet information of the biflows (i.e., `PL`, `DIR`, `IAT`, and `TCPWIN`).

What emerged in this analysis about the inability of classical inputs to capture the peculiarities of the activities performed is confirmed in Sec. 5.1.1, where we show experimentally that they are insufficient to solve their traffic classification.

4.3 Analysis on Time Evolution of App Connections

The analysis shown in Sec. 4.2 suggests that the inputs commonly used as input to DL traffic classifiers fail to capture the peculiarities of the activities performed by the user within the same app. Hence, here we investigate whether alternate paths are viable based on the observation of "simultaneous" (viz. concurrent) biflows. Indeed, the behavior of an app on the network is the result of the mixture of multiple concurrent traffic biflows between the client—running a certain (communication and collaboration) app—and a variety of servers [20], whose number and characteristics may depend both on the app and the activity. To validate this hypothesis, we analyze the *overall traffic* generated or received by the app during a traffic capture to quantify the number of concurrent biflows over time by considering (non-overlapping) windows.

Formally, for a capture starting at time t_0 and having duration D, the i^{th} aggregation interval gathers all biflows that transmit at least one packet within $[t_0 + (i-1)\Delta, t_0 + i\Delta)$, with $i \in \{1, 2, \cdots \lceil \frac{D}{\Delta} \rceil\}$, where Δ is

the duration of the sampling window.

Fig. 4.4 shows the amount of concurrent biflows for three exemplifying apps—i.e., `Skype`, `Teams`, and `Webex`)[7]—using a window size $\Delta = 5\,\text{s}$.

In detail, we observe a limited time frame at the very beginning of the communication during which the client generates several biflows whose number depends on the type of activity conducted by the user (this is particularly evident, for instance, in Fig. 4.4d). These concurrent interactions might be connected to authorization/accounting, telemetry, and advertising services, each expected to be located on dedicated servers[8]. The number of generated biflows then settles to a lower value (which varies with both the app and the activity and ranges from 3 to 13 biflows). Interestingly, for some apps such as `Teams`, this plateau value appears more stable for both `ACall` and `VCall`, whereas `Chat` exhibits more pronounced variations across multiple captures. Generally, if we compare the number of biflows associated with `ACall` and `VCall` before reaching the plateau, we can observe that in the case of `Discord`, `GotoMeeting`, and `Teams`, `VCall` is characterized by a higher number of biflows than `ACall`, while the opposite scenario occurs in the case of `Webex` and `Zoom`.

Observing the steady-state behavior, `VCall` tends to maintain a higher number of active biflows than `ACall` for `Discord`, `GotoMeeting`, `Teams`, and `Zoom`, while the two activities tend to maintain the same number of active biflows for `Webex`. Interestingly, the behavior of these activities is very different for `Teams`—with `VCall` always being associated with a higher number of active biflows. This does not hold for `Skype` where the behavior is similar in both phases. Finally, for both `Skype` and `Teams`, the traffic related to `Chat` is always carried by fewer biflows than the other two activities. On the other hand, `Discord` maintains a comparable number of active biflows in the case of `Chat` and `ACall` activities. At the same time, for `Zoom`, this holds only in the early stages between `Chat` and `VCall`, while at steady state `Chat` maintains fewer active biflows than `VCall`.

We also evaluated the number of different server sockets contacted by the client, which characterize the packets belonging to the same *service*

[7]Similar behaviors to `Skype` and `Webex` were also observed for `Meet` and `Slack`, respectively.

[8]An experimental analysis of 500 popular mobile apps conducted in [20] has found an average of 4.5 different cloud services accessed per mobile app.

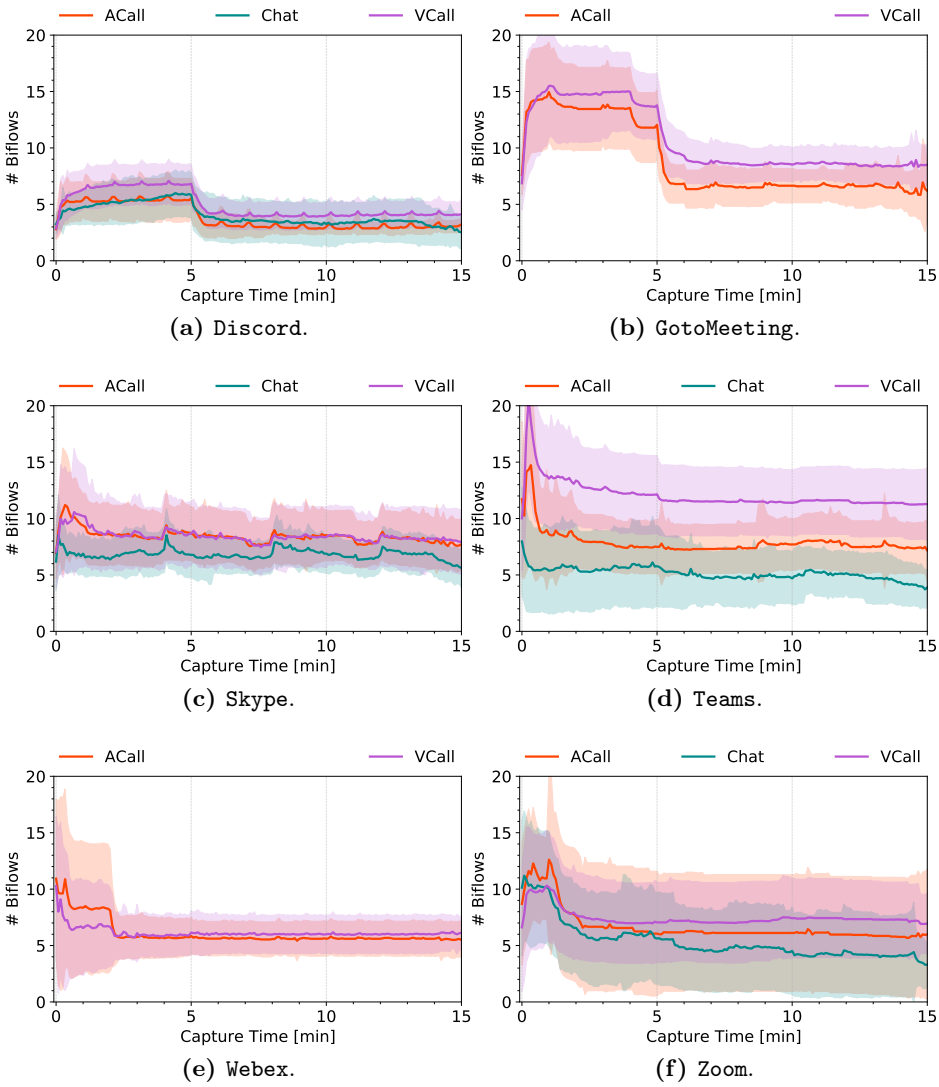


Figure 4.4. Amount of concurrent biflows ($mean \pm std$ across different captures) in each 5-second slot for `Skype` (a) and `Teams` (b), and `Webex` (c) across the different activities performed. Values are calculated considering the first 15 min of each capture.

burst [92]. The definition of service burst starts from that of *burst* [93], defined as a sequence of packets (regardless of the biflow they belong to) having an inter-packet time shorter than a given threshold. Hence, a service burst is defined as the subset of packets of a burst belonging to biflows that share the same *transport protocol, and destination IP:port pair*. The service burst ideally groups packets related to the same server application and strictly-related information (due to the timing) and has been used previously for the purpose of TC [43, 92, 94], implying also that the client device is set (as in our analysis). At any time, a service burst can refer to a single server socket, not including traffic related to other simultaneous communications from the same client. Performing an analysis of concurrent server sockets (strictly analogous to what we have done with biflows), we can observe trends similar to those shown in Fig. 4.4 for biflows but with a notable offset. Specifically, we found that the number of server sockets is at most 50% of the overall number of active biflows. This clearly implies that multiple biflows concurrently reach the same service. *Hence, a significant number of concurrent biflows and multiple concurrent service bursts imply that a single service burst (not to say a single biflow) cannot account for most traffic exchanged by a given app at a given time interval. Therefore, contextual information is missed when using biflow-based or service-burst-based inputs.*

Takeaways

RQ: *How do CC apps' connections change over time based on user activity?*

Concurrent biflows are important to consider when analyzing the network behavior of user activities. The number varies depending on the user activity and the app. A single biflow or service burst cannot account for all the traffic related to a specific activity, highlighting the need for contextual information when using biflow or service-burst-based inputs for this analysis.

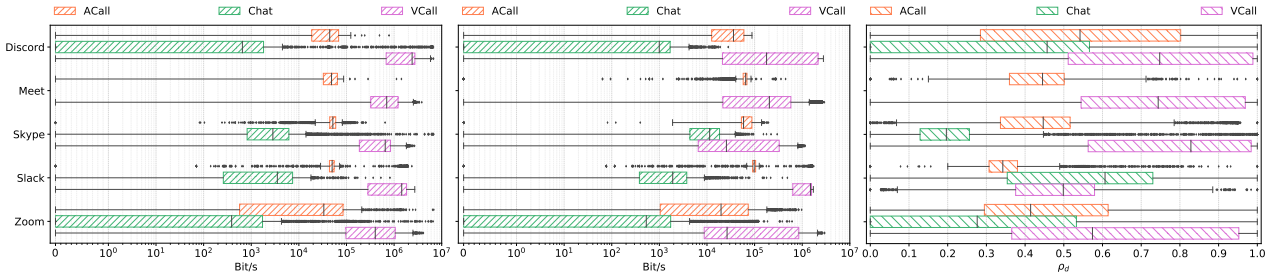
As we will see in Sec. 5.2, these experimental findings will be crucial for defining novel inputs specially designed to capture the peculiarities of activities when addressing their traffic classification.

4.4 Characterization of Simultaneous Biflows

Informed by the analysis of the time evolution of app communications, in this section, we characterize the collected traffic in terms of (a) bitrate and (b) packet-rate. In detail, we refer to *aggregated rates*, i.e., computed by considering the whole traffic either generated or received by the app (i.e., including concurrent bidirectional flows) during a traffic capture. In the following, aggregate metrics are computed considering a (non-overlapping) window size $\Delta = 5\,\text{s}$.[9] Formally, for a capture starting at time t_0 and having duration D, the i^{th} aggregation interval gathers all packets whose arrival time falls within $[t_0 + (i-1)\Delta, t_0 + i\Delta)$, with $i \in \{1, 2, \cdots \lceil \frac{D}{\Delta} \rceil\}$. Figs. 4.5a–4.5b and 4.5d–4.5e report the boxplots of bitrate and packet-rate (both computed over 5-second intervals), respectively, for five exemplifying apps (i.e., `Discord`, `Meet`, `Skype`, `Slack`, and `Zoom`) considering upstream and downstream directions separately. For each app, the distribution is broken down across the specific activities highlighted with different colors.

Observing the **downstream bit-** and **packet-rates** (see Figs. 4.5a

[9] Further significant values of $\Delta = 10, 30, 60$ s have been tested without showing notable discrepancies.



(a) Downstream bitrate. **(b)** Upstream bitrate. **(c)** % of Downstream Volume, ρ_d.

(d) Downstream packet-rate. **(e)** Upstream packet-rate. **(f)** % of Downstream Packets, γ_d.

Figure 4.5. Downstream bitrate (a), upstream bitrate (b), percentage of downstream volume (c), downstream packet-rate (d) upstream packet-rate (e), and percentage of downstream packets (f) for `Discord`, `Meet`, `Skype`, `Slack`, and `Zoom`. Values are evaluated over $\Delta = 5$ s time intervals. Boxes report the 1^{st} and 3^{rd} quartiles (1Q and 3Q, respectively), while whiskers mark 1Q−1.5 IQR and 3Q+1.5 IQR, where IQR=3Q−1Q. Black diamonds highlight outliers. Values on the x-axis are reported in Symlog-scale.

and 4.5d, respectively), we can notice that the different activities are characterized by significantly-different patterns in terms of median rate, which, however, do not seem strictly related to a specific app. For all apps `VCall` always corresponds to the highest bit-rate (resp. packet-rate), which varies from $\approx$ 402 Kbps (resp. $\approx$ 104 pps) to $\approx$ 2.4 Mbps (resp. $\approx$ 295 pps), corresponding to `Zoom` and `Discord`, respectively. This is expected given the *simultaneous transmission of both audio and video traffic*. In contrast, for `ACall` and `Chat`, the bit-rate (resp. packet-rate) varies in the range $[33, 52]$ Kbps (resp. $[27, 53]$ pps) and $[0.4, 3.5]$ Kbps (resp. $[0.2, 1.2]$ pps), with `Skype` and `Slack` having the highest rates, respectively. Interestingly, most apps exhibit a similar bit-rate (resp. packet-rate) for `ACall`, varying in a range of only 8 Kbps (resp. 2 pps). Finally, we observe that `Chat` and `VCall` have the least and the most variability in terms of bit-rate (resp. packet-rate), with IQRs varying from 1.7 Kbps (resp. 1 pps) to 7 Kbps (resp. 1.8 pps) and 650 Kbps (resp. 98 pps) to 2.05 Mbps (resp. 295 pps), respectively.

Moving to the **upstream bit-** and **packet-rate** (Fig. 4.5b and 4.5e, respectively), there is not always a clear separation as in the downstream case. Indeed, for `Zoom` [10], `ACall` and `VCall` tend to have closer median values, whose difference is only $\approx$ 7 Kbps (resp. $\approx$ 7 pps) in terms of bit-rate (resp. packet-rate). In addition, `Meet` exhibits similar behavior only in terms of packet rates, where the difference between the medians is $\approx$ 9 pps.

In contrast, other apps result in significant discrepancies when comparing upstream bit- and packet-rate of such activities. In fact, results on `Discord`, `Meet`, and `Slack` [11] confirm the previously observed trends in the downstream direction, where the median bit-rate (resp. packet-rate) of `VCall` is consistently higher than that of `ACall`. The largest difference is observed for `Slack`—i.e., 97.8 Kbps vs 1.5 Mbps (resp. 101 pps vs 242 pps). Conversely, the smallest difference is observed for `Discord`—i.e., 36 Kbps vs 178 Kbps (resp. 25 pps vs 67 pps). Notably, the above trend is the opposite for `Skype` [12], where `ACall` corresponds to the higher median bit-rate and packet-rate compared to `VCall`—i.e., 59 Kbps vs 26 Kbps and 52 pps vs 9 pps. For `Chat`, the activity pattern seems quite distinguishable from

[10] Similar results were also observed for `GotoMeeting` and `Webex`.
[11] Similar results were also observed for `GotoMeeting` and `Messenger`.
[12] Similar results were also observed for `Teams`.

`ACall` and `VCall` for *both* rate metrics (cf. Figs. 4.5b and 4.5e) for almost all apps. Conversely, this does not hold for `Skype`, for which `Chat` bit-rate is much more similar to that of `VCall`.

Considering **traffic direction**, we quantify the fraction of downstream volume and packet using ρ_d and γ_d, which are defined as follows: $\rho_d = \frac{B_d}{B_d+B_u}$ and $\gamma_d = \frac{P_d}{P_d+P_u}$, where B_d and B_u refer to the number of downstream and upstream bytes, respectively, and P_d and P_u refer to the number of downstream and upstream packets, respectively.

Figs. 4.5c and 4.5f report the distribution of these two metrics computed over 5-second intervals. In detail, the results show that `ACall` is the most balanced activity in terms of volume with ρ_d consistently in the range $[0.4, 0.6]$ for all apps except `Slack` (whose median $\rho_d \approx 0.34$). The above results are even more pronounced when considering the number of packets where γ_d is close to 0.5 for most apps except `Slack` (whose median $\gamma_d \approx 0.34$). Still considering `ACall`, it is worth noticing that ρ_d and γ_d span over slightly-lower values, witnessing the non-negligible presence of small upstream packets, likely due to signaling operations. Conversely, traffic is remarkably unbalanced towards the downstream direction when considering the volume of `VCall`, where the median value of ρ_d is consistently > 0.75 for most apps. Similar but less general results can also be observed in the number of packets. Only 4 apps exhibit a median $\gamma_d > 0.6$, while for the remaining ones, it is close to 0.5, making them comparable to `ACall` regarding traffic balancing. In addition, results obtained for `Meet` [13] show that it tends to generate larger downstream packets.

Finally, focusing on `Chat` results in more balanced traffic, with a ρ_d close to 0.5 for most apps. In contrast, we observe a significant imbalance toward the upstream direction on `Skype` and `Zoom`, with median ρ_d (resp. γ_d) values of 0.20 (resp. 0.33) and 0.27 (resp. 0.33), respectively. However, it is worth noting that both ρ_d and γ_d tend to extend over larger values, which affects the left side of the figure. This is a witness to the non-negligible presence of upstream packets of varying sizes.

[13] Similar results were also observed for `GotoMeeting` and `Webex`.

Takeaways

RQ: *Can aggregate traffic of simultaneous biflows provide insights about user activities of CC apps?*

Different activities display distinct patterns in median rates in the downstream direction related to the specific activity, with `VCall` consistently exhibiting the highest bit-rate and packet-rate compared to `ACall` and `Chat`.

In the upstream direction, there is less separation between `ACall` and `VCall`, with some apps showing similar median values for both activities and others displaying significant differences. The pattern related to `Chat` activity is distinguishable from `ACall` and `VCall` for most apps.

Focusing on traffic direction, `ACall` is the most balanced in volume traffic, while `VCall` traffic is significantly unbalanced towards the downstream direction. Finally, `Chat` results in more balanced traffic for almost all apps.

4.5 Traffic Modeling via Markov Chains

Herein, we aim to bring out the peculiar characteristics of each app by modeling its traffic through (*Multimodal*) *Markov Chains* (referred to as *MCs*), following [50].

Our objective is to use this modeling approach to better understand the distinctive behaviors of CC apps, by examining the relationships between individual packets rather than their aggregate traffic. In addition, we aim to provide a clear and intuitive visual representation comparable to the methods used in previous analyses but with finer granularity.

A MC is a stochastic model $\{\boldsymbol{x}^n, n \in \mathbb{N}_0\}$ that describes how an M-dimensional random variable varies over time. Specifically, $\boldsymbol{x}^n$ is the discrete random variable that describes the state of the system at time n, whose m^{th} modality, x_m^n, can only assume S_m possible values. Consequently, the total number of different states is defined on the Cartesian

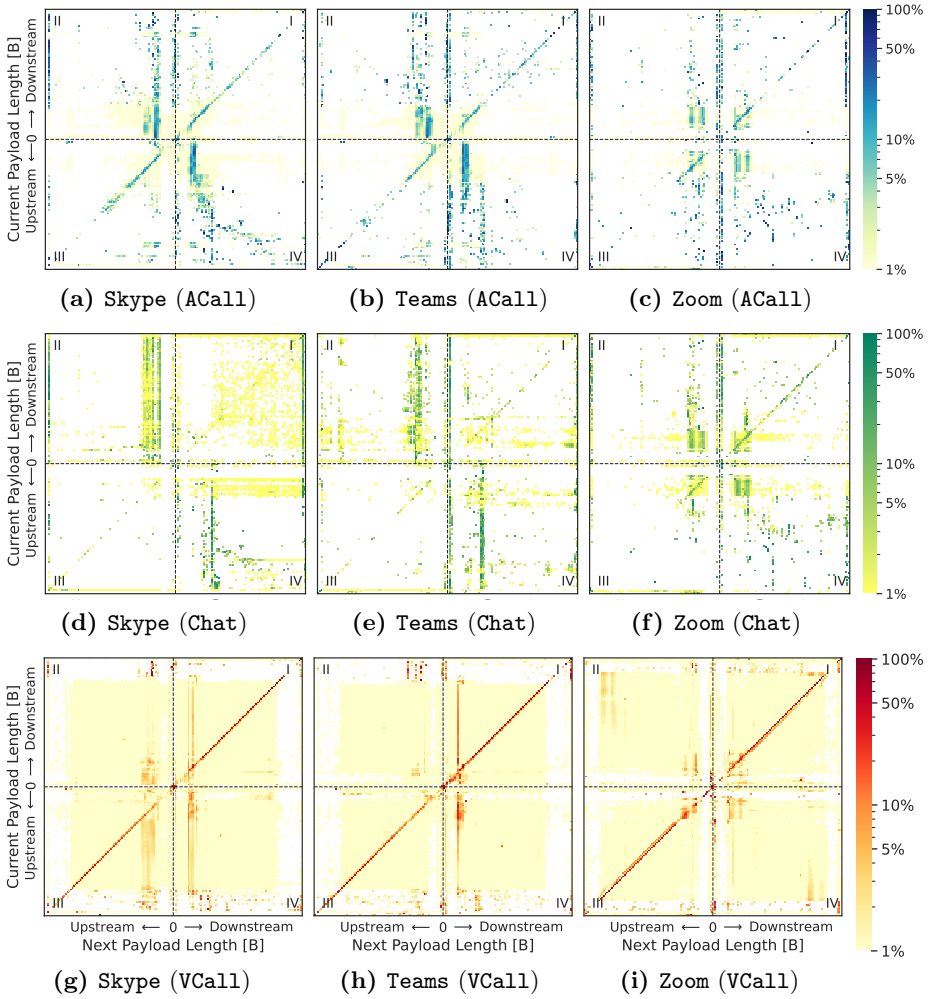


Figure 4.6. Transition matrices of payload length and packet direction for `Skype` (a, d, g), `Teams` (b, c, h), and `Zoom` (c, f, i), and related to `ACall`, `Chat`, and `VCall` activities.

product of the individual modality spaces, having size $S \triangleq \prod_{m=1}^{M} S_m$.

Herein, we refer to stationary first-order MCs, which are a type of discrete-time stochastic process satisfying two assumptions: (i) 1^{st}-order *Markov property* $\Pr(\boldsymbol{x}^{n+1}|\boldsymbol{x}^n, \ldots) = \Pr(\boldsymbol{x}^{n+1}|\boldsymbol{x}^n)$, namely the probability distribution of the state at time $n+1$ depends only on the most recent observation; and (ii) *homogeneity* $\Pr(\boldsymbol{x}^{n+1}|\boldsymbol{x}^n) = \Pr(\boldsymbol{x}^n|\boldsymbol{x}^{n-1})$, $\forall n$, namely the state transition distribution is independent on n. Accordingly, a stationary MC can be equivalently represented via the *initial probability vector* $\boldsymbol{q}^0$—with $q_i^0 = \Pr(\boldsymbol{x}^0 = \boldsymbol{s}_i)$ and $\sum_{i=1}^{S} q_i^0 = 1$—and the *transition probability matrix* $\boldsymbol{P} \in [0,1]^{S \times S}$—with $p_{i,j} = \Pr(\boldsymbol{x}^n = \boldsymbol{s}_j | \boldsymbol{x}^{n-1} = \boldsymbol{s}_i)$ and $\sum_{j=1}^{S} p_{i,j} = 1$.

In this analysis, we focus on $\boldsymbol{P}$ to emphasize the distinctive traffic patterns of CC apps, considering the specific activities performed by users.

Specifically, for each app, we consider *jointly* the (transport-layer) `PL` and the `DIR` of the packets of each biflow (i.e., $M = 2$). We derive the corresponding transition matrix, where $< (p_i, d_i), (p_j, d_j) >$ represents the probability that the next packet comes with a `PL` of p_j bytes and a direction d_j if the last observed packet has a `PL` of p_i bytes and a direction d_i.

Each transition matrix is learned via *Maximum-Likelihood* (ML) estimation from the MIRAGE-COVID-CCMA-2022 dataset, herein referred to as $\mathscr{D}$, comprising multiple time series of the vector variable, namely $\mathscr{D} \triangleq \bigcup_{t=1}^{T} \boldsymbol{X}_t$, where $\boldsymbol{X} \triangleq \{\boldsymbol{x}^0, \ldots, \boldsymbol{x}^{N-1}\}$ represent the time series of the vector observation over the whole sequence $n = 0, 1, \ldots, N-1$.

To do this, we performed two pre-processing operations: *(i)* removal of null-payload packets, which are assumed to be non-informative since they do not reflect the behaviors of the app but only the mechanisms of the transport layer; *(ii)* discretization of the `PL` through an unsupervised adaptive binning procedure based on K-means, leading to 80 bins.[14]

In Fig. 4.6, we show the transitions matrices obtained for (a) `Skype`, (b) `Teams`, and (c) `Zoom` by considering (i) `ACall`, (ii) `Chat`, and (iii) `VCall` activities separately. From visual inspection of results (higher values are associated with a darker color), **different patterns** can be identified in the matrices depending on the app and the activity considered.

Specifically, for all apps, we can observe a **dark pattern on the main**

[14] Such choice was obtained by evaluating the quantization error of `PL` values versus the number of bins and by choosing the value balancing the error and the model complexity.

diagonal (↗) for activities generating audio or video traffic (i.e., `ACall` and `VCall`). This pattern indicates that such activities generate pairs of packets with equal `PL` and `DIR`. We also observe that this pattern appears in both directions, reflecting the nature of the activity involving the active (symmetric) participation of the users. In all cases on `VCall`, the highly probable values are less sparse than on `ACall` and tend to be concentrated mostly along the main diagonal. In particular, we note that these patterns involve values of `PL` $\leq$ 500B in the case of `ACall`, whereas they vary in the range $[60, 1300]$B in the case of `VCall`.

In addition, especially on `ACall`, we observe some **darker areas** (blocks ■), indicating that apps are more likely to generate packets that match values falling within a similar set of values. Interestingly, for `Zoom`, these areas do not depend on the direction of the packets (i.e., they appear in all quadrants). In contrast, for `Skype` and `Teams`, they only appear when the observed pairs of packets have opposite directions (i.e., II and IV quadrants). This indicates a greater correlation between the traffic exchanged by the two parties involved in the communication.

In the case of `Chat`, we can observe many **vertical patterns** (↑) in the II and IV quadrants, highlighting the presence of highly probable-values of downstream (resp. upstream) `PL`s within the observed traffic that do not depend on the current upstream (resp. downstream) value. These patterns also highlight that apps generate packets with fairly small or very large `PL` (i.e., $\leq$ 100B or $\geq$ 1460B, respectively). Finally, it is worth noting that these patterns appear in all quadrants for `Zoom`, suggesting that the direction of the previous packet has less impact on the features of the next packet.

Takeaways

RQ: *Can Markov Chains understand how a CC app's behavior varies with activity?*

Within the same biflow, activities in a single app display varying transition patterns that describe the link between two successive packets in terms of their payload length (viz. `PL`) and direction (viz. `DIR`).

`ACall` and `VCall` tend to generate pairs of packets with identical `PL` and `DIR`. This pattern appears in both communication directions, reflecting the symmetric nature of user participation. Moreover, especially for `ACall`, apps tend to generate packets with similar values, and this correlation is more pronounced when packets have opposite directions.

For `Chat`, the identified patterns show highly probable `PL` values for the next packet. These values are either small or very large (i.e., $\leq$ 100B or $\geq$ 1460B, respectively), and their direction depends only on that of the current packet.

Chapter 5

Traffic Classification using Deep Learning

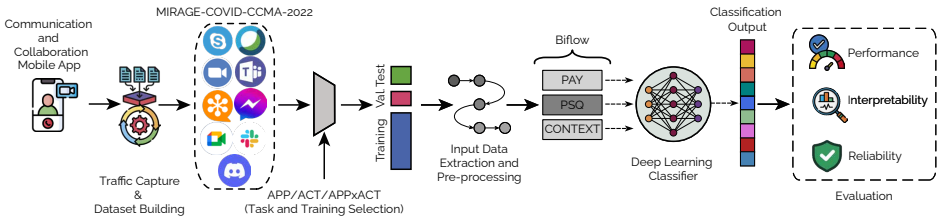


Figure 5.1. Workflow of the methodology defined for classifying the traffic of *communication-and-collaboration mobile apps* via DL approaches and explaining/assessing the classification outcomes via XAI techniques.

THIS chapter delves into classifying traffic generated by *Communication and Collaboration Apps* using the framework depicted in Fig. 5.1. Specifically, in Sec. 5.1, we initially assess the effectiveness of state-of-the-art DL approaches in classifying such traffic at varying levels of detail, specifically focusing on individual apps and user activities. This analysis shows the limitations in classifying user activity using common inputs adopted in the existing literature [18]. In response to these limitations, in Sec. 5.2, we introduce a novel set of inputs, referred to as `Context`, which gather information about a biflow by examining its surrounding context, specifically the coexisting biflows that occur simultaneously. Capitalizing

on the potential of these `Context` to enhance user activity classification, we present a novel multimodal classification approach named Mimetic-All, which leverages them as an additional modality.

In the following, we provide a detailed description of each approach employed, including (a) the specific traffic input used, (b) the employed single- or multi-modal DL architecture; and (c) the learning and evaluation setup used to assess its performance.

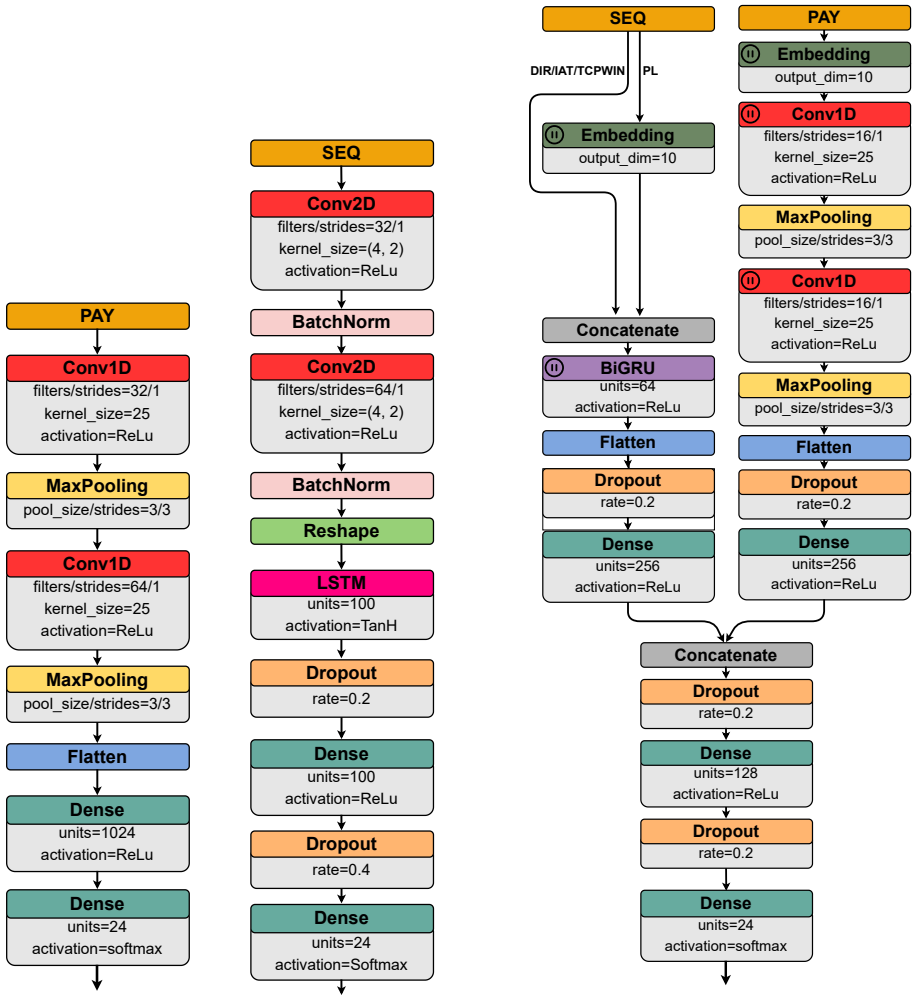
5.1 Assessing the current state-of-the-art

In this section, we provide a comprehensive overview of the state-of-the-art DL classifiers that were taken into account during the experimental analysis, along with the employed experimental setup (Sec. 5.1.1). Subsequently, following a sensitivity analysis aimed at optimizing the input sizes (Sec. 5.1.4), we proceed to evaluate the effectiveness of these solutions in addressing classification at both the app and activity levels (Sec. 5.1.5).

5.1.1 Baselines considered

In the following, we leverage the biflow as the relevant traffic object of our TC tasks. A biflow encompasses packets sharing the same quintuple `(IP src, IP dst, port src, port dst, protocol)` in both upstream and downstream directions [18].

We consider state-of-the-art classifiers selected among the best single-modal and multimodal alternatives—based on extensive performance evaluation carried out in previous works [18, 29, 34]—in terms of both DL architecture and *unbiased* input data. Furthermore, this choice provides the benefit of having a set of comparison baselines differentiating in terms of input types and architectural layers (e.g., convolutional, recurrent, and hybrid combinations). More in detail, we evaluate a 1D-CNN architecture that takes the initial N_b bytes of the transport-layer payload (`PAY`) from each biflow as input [27]. The 1D-CNN architecture (rf. Fig. 5.2a) comprises two 1D convolutional layers, with 32 and 64 filters, respectively, using a kernel size of 25 and a unit stride. Each convolutional layer is followed by a 1D max-pooling layer with a spatial extent of 3 and a unit stride. Finally, the network has a dense layer with 1024 neurons.



(a) 1D-CNN **(b)** Hybrid **(c)** Mimetic-Enhanced

Figure 5.2. 1D-CNN (a), Hybrid (b), and Mimetic-Enhanced (c) architecture. For each single block, the color filling indicates the type of layer. For Mimetic-Enhanced, the ⏸ symbol marks layers frozen during the fine-tuning phase.

Additionally, we assess a hybrid composition of *2D-CNN+LSTM* (referred to as HYBRID hereafter), as proposed in [28]. This hybrid model takes the following informative fields (`SEQ`) from the initial N_p packets of each biflow as input: *(i)* packet direction (`DIR`) $\in \{-1, 1\}$, *(ii)* the number of bytes in the transport-layer payload (`PL`), *(iii)* TCP window size (`TCPWIN`, set to zero for UDP packets), and *(iv)* inter-arrival time (`IAT`). This architecture (rf. Fig. 5.2b) comprises two 2D convolutional layers, with 32 and 64 filters, respectively. These layers use a kernel size of $(4, 2)$ and a unit stride, each followed by a batch normalization layer. Subsequently, the reshaped output is passed through a sequence consisting of an LSTM and a dense layer with 100. Following each of these layers, a dropout of 0.3 is applied.

Both architectures end with a softmax classifier and employ Rectified Linear Unit (ReLU) and Hyperbolic Tangent activations in the convolutional and LSTM layers, respectively.

Lastly, we also consider the *multimodal* MIMETIC-ENHANCED classifier [29], which represents an *enhanced* version of the original MIMETIC framework initially introduced in [34]. The MIMETIC-ENHANCED architecture (rf. Fig. 5.2c) involves two modalities, each receiving one of the two input types, namely `PAY` and `SEQ`, employed in the baseline single-modal classifiers described above. To enrich the information contained in the input data, MIMETIC-ENHANCED enhances the original MIMETIC by leveraging a *trainable* embedding layer for both modalities, transforming each input into a vector of size $e = 10$. This results in an overall embedding matrix denoted as $E \in \mathscr{R}^{N \times e}$, where N denotes the input dimensionality.

From a structural perspective, the modality fed with the `PAY` input type comprises two 1D convolutional layers, with 16 and 32 filters, respectively, utilizing a kernel size of 25 and a unit stride. Each layer is followed by a 1D max-pooling layer with a spatial extent of 3 and a unit stride, concluding with a dense layer containing 256 neurons.

Conversely, the modality fed with the `SEQ` input type includes a BiGRU layer with 64 units and return-sequences behavior, followed by a dense layer with 256 neurons.

Subsequently, the intermediate features extracted from each modality are concatenated and input into a shared dense layer with 128 neurons, followed by the final softmax classifier. All the layers mentioned above, ex-

cept for the softmax layer, utilize Rectified Linear Unit (ReLU) activations. Additionally, a dropout of 0.2 is applied at the end of each single-modality branch and after every dense layer to prevent overfitting and promote regularization.

Training Mimetic-Enhanced involves a two-phase process, beginning with an independent *pre-training* of each single-modality branch and concluding with fine-tuning the entire architecture after freezing the lower single-modality layers (⓫ in Fig. 5.2c). In the former stage, the individual pre-training of the p^{th} single-modality branch is accomplished using a *softmax stub* [34]. This process also incorporates an adaptive learning-rate scheduler that halves the learning rate every 5 epochs, starting with a value of $2 \cdot 10^{-3}$ and 10^{-3} for pre-training and fine-tuning, respectively.

For further in-depth details on the Mimetic-Enhanced architecture and related hyperparameters, please refer to [29].

5.1.2 Evaluation Procedure and Metrics

In the following analyses, we employ a *stratified ten-fold cross-validation* method for performance evaluation. This approach is chosen because it provides a robust evaluation setup that maintains a consistent ratio of samples among different classes in each fold.

Moreover, to ensure a fair and equitable comparison, we train all the classifiers, including both the newly proposed models and the baseline classifiers (for a detailed description of our proposals, refer to Sec. 5.2.3), for a total of 100 epochs. For Mimetic-Enhanced and the new multimodal proposals, we divide these epochs into 30 epochs for pre-training each modality and 40 epochs for fine-tuning. The training objective is to minimize the categorical cross-entropy loss. We employ the Adam optimizer with a batch size of 50 and implement a *validation-based early-stopping* technique to prevent overfitting. Specifically, for each fold, we designate 80% of the entire training data as the actual training set, while the remaining 20% is allocated to the validation set.

Furthermore, hereinafter, we leverage the available ground-truth information associated with each biflow—which includes labels for both the app generating the traffic and the specific activity the user performs. This ground-truth data serves as the basis for instructing and evaluating various supervised strategies for *three* Traffic Classification (TC) tasks:

(i) classifying the app (App-TC), *(ii)* classifying the activity (Act-TC), and *(iii)* classifying both the app and the activity (Joint-TC). Consequently, we *train* the aforementioned models using three different approaches: *(a)* app-related ground truth only (APP), *(b)* activity-related ground truth only (ACT), and *(c)* the joint app-and-activity ground truth (APP×ACT).

It is worth noting that models trained using the APP and ACT approaches are designed specifically for the task they are trained on, either classifying apps or activities. On the other hand, when training the deep learning architectures based on the APP×ACT approach, all three TC tasks mentioned above can be effectively addressed.

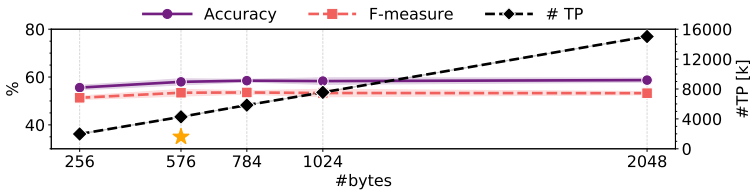
We use two metrics to evaluate the classification performance of DL models: *accuracy* and *F-measure*. Accuracy is the share of correctly-classified samples, while F-measure is the harmonic mean of precision (the proportion of classifier decisions for a given class that are actually correct) and recall (the per-class accuracy).

5.1.3 Implementation Details

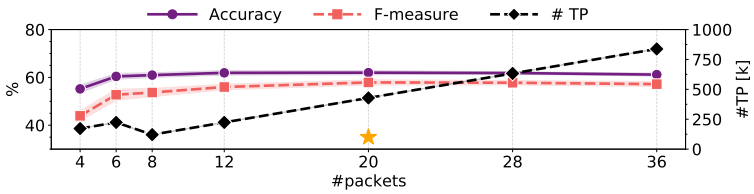
For implementing and testing the DL architectures described in this chapter, we exploit the model provided by `Keras` (https://keras.io) Python API running on top of `TensorFlow 2` (https://www.tensorflow.org/). Input data are formatted in Parquet and optimally managed via `Apache PyArrow` (https://arrow.apache.org/). Data pre- and post-processing have been performed mainly by means of `numpy` (https://numpy.org/) and `pandas` (https://pandas.pydata.org/) libraries. For the evaluation metrics reported in Sec. 6.3.3, we use the implementation of `scikit-learn` (https://scikit-learn.org/). Finally, the graphical data representation has been obtained using `matplotlib` (https://matplotlib.org/) and `seaborn` (https://seaborn.pydata.org/) libraries.

5.1.4 How to set traffic parameters?

First, we perform a sensitivity analysis to determine the optimal number of bytes and packets used as input, denoted by N_b and N_p, respectively. We refer to the Joint-TC task for brevity, i.e., the hardest in its nature.



(a) 1D-CNN.



(b) Hybrid.

Figure 5.3. Accuracy [%], F-measure [%], and number of trainable parameters of 1D-CNN (a) when varying the input dimensions N_b and Hybrid (b) when varying the input dimensions N_p. Results refer to the Joint-TC task. The best trade-off value is highlighted via a $\star$ marker.

Fig. 5.3 shows the *accuracy* and *F-measure* attained by the 1D-CNN and Hybrid architectures, when varying $N_b \in [256, 2048]$ B and $N_p \in [4, 36]$ packets, respectively. We chose not to analyze the multimodal Mimetic-Enhanced due to the combinatorial complexity of considering all possible combinations of N_b and N_p and the time required to train and test each configuration. We also report how the number of trainable parameters (`#TP`) varies with the size of the considered input data to highlight the (necessary) input size-complexity trade-off.

As reported in Fig. 5.3a, although the best performance in terms of accuracy and F-measure is obtained with $N_b = 784$ B, when passing from $N_b = 576$ B to $N_b = 784$ B, both metrics remain stable despite the larger input size. Additionally, the same variation of N_b causes a non-negligible increase in the number of trainable parameters (+1.5M), resulting in a much more complex architecture with a limited improvement (i.e., $\leq 0.5\%$ in terms of F-measure).

On the other hand, regarding Fig. 5.3b, we can notice that Hybrid reaches the best performance regarding both accuracy and F-measure by

using $N_p = 20$ packets. In this specific case, considering 20 packets instead of 12 still provides a small improvement of performance in terms of F-measure, at the cost of a slight increase (and thus manageable) of complexity (+200k in terms of number of trainable parameters).

Regarding the latter, it is useful to underline that the number of trainable parameters corresponding to $N_p \in [4, 6]$ is comparable to that obtained by using $N_p = 12$ packets. The reason for this can be traced to the fact that such a small input implies the use of different padding, which causes additional complexity in implementing the HYBRID architecture. Moreover, we underline that the number of trainable parameters of HYBRID is dominated by that of 1D-CNN, being the former more than one order of magnitude smaller than the latter. This consideration also applies to all architectures and single-modality branches fed with `SEQ` compared to those fed with `PAY`.

Takeaways

RQ: *How many packets (viz. N_p) or payload bytes (viz. N_b) do I need to jointly classify the app and the activity?*

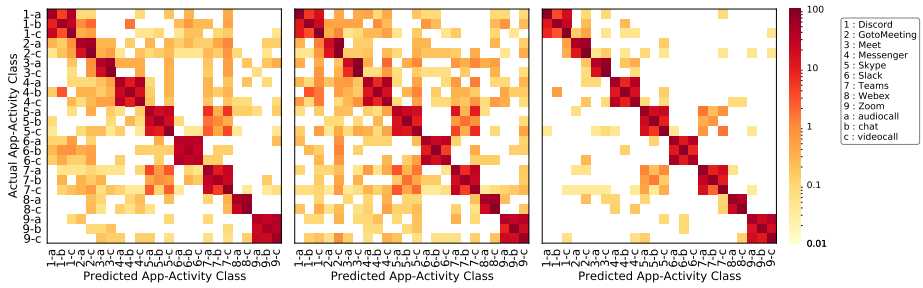
Even when optimizing the input size, there is a limit to the achievable TC performance. Therefore, to balance the classification performance (measured by F-measure) with the complexity of the classifiers obtained, in the following, we employ $N_b = 576$ B and $N_p = 20$ packets.

5.1.5 Is current state-of-the-art suitable to classify both apps and user activities?

Hereinafter, we evaluate the performance of state-of-the-art classifiers (i.e., 1D-CNN, HYBRID, and MIMETIC-ENHANCED) when adopting different training strategies (i.e., APP, ACT, and APP×ACT). We recall that APP (resp. ACT) can solve only the App-TC (resp. Act-TC) task, whereas APP×ACT allows to solve both each separate task and the Joint-TC problem. As a result, Tab. 5.1 reports the performance of the classifiers in terms of both accuracy and F-measure attained for each TC task (column-wise). The table is also complemented by a computational complexity assessment

Table 5.1. Comparison of accuracy, F-measure, number of Trainable Parameters (#TP), and Training Time (Time) for the three state-of-art DL architectures (1D-CNN, Hybrid, and Mimetic-Enhanced) when trained on different class-labels (i.e. related to App×Act, App, and Act) for different classification tasks. #TP slightly varies with the classification task (with variations being smaller than reported precision). Results are in the format *avg.* (±*std.*) obtained over 10-folds. Training time was computed by pre-training the individual modalities in parallel. For each metric (column), the best and the worst architectures are highlighted in green and red, respectively. **MM** denotes a multi-modal architecture.

Classifier	MM	Training Strategy	Joint-TC		App-TC		Act-TC		#TP [k]	Time [*min*]
			Accuracy [%]	F-measure [%]	Accuracy [%]	F-measure [%]	Accuracy [%]	F-measure [%]		
1D-CNN	○	App×Act	57.99(±1.30)	53.44(±1.42)	95.93(±0.49)	96.49(±0.40)	59.94(±1.46)	56.37(±1.57)	4272	57(±5)
		App	-	-	97.33(±0.38)	97.93(±0.32)	-	-	4256	52(±4)
		Act	-	-	-	-	59.56(±1.23)	55.76(±1.38)	4250	65(±5)
Hybrid	○	App×Act	61.97(±0.86)	57.89(±1.06)	94.25(±0.61)	95.01(±0.60)	65.11(±0.81)	63.47(±1.02)	428	26(±5)
		App	-	-	94.95(±0.46)	95.69(±0.40)	-	-	426	29(±7)
		Act	-	-	-	-	64.19(±0.79)	62.45(±0.96)	426	19(±5)
Mimetic-Enhanced	●	App×Act	70.17(±0.64)	66.10(±0.60)	99.02(±0.08)	99.19(±0.08)	70.73(±0.63)	69.16(±0.60)	1235	80(±11)
		App	-	-	99.19(±0.16)	99.35(±0.13)	-	-	1225	58(±4)
		Act	-	-	-	-	69.72(±1.19)	68.25(±1.36)	1221	74(±10)



(a) 1D-CNN. **(b)** HYBRID. **(c)** MIMETIC.

Figure 5.4. Confusion matrices of 1D-CNN (a), HYBRID (b), and MIMETIC-ENHANCED (c) considering the APP×ACT and related to the Joint-TC. Note that the log scale is used to evidence small errors.

(via the "`#TP`" and the training "Time" columns). It is worth noting that, in the case of MIMETIC-ENHANCED, since the training methodology adopted allows parallelizing the pre-training of the two modalities, we have calculated the resulting training time by adding the time needed to perform the pre-training of the "slower" modality and the time needed to perform the successive fine-tuning of the whole architecture.

By comparing the *general performance* achieved on the three TC problems, Joint-TC confirms to be the *most difficult* task to tackle (with $58\% - 70\%$ accuracy and $53\% - 66\%$ F-measure ranges) due to the greater number of classes (i.e., the 24 combinations of apps and activities). In contrast, the (easier) App-TC task (consisting of 9 classes) can be effectively solved by all the architectures ($94\% - 99\%$ accuracy and $95\% - 99\%$ F-measure). Finally, the performance obtained on the Act-TC task highlights the *actual difficulty* in activity recognition. Indeed, despite the smallest number of classes (i.e., the 3 activities) considered in our study, very low performance ($60\% - 71\%$ accuracy and $56\% - 69\%$ F-measure) is attained with all state-of-art approaches. By looking at the classifier standpoint, MIMETIC-ENHANCED *outperforms* the two competing approaches on all the three considered TC tasks. For instance, MIMETIC-ENHANCED roughly achieves +8% (resp. +13%) F-measure than HYBRID (resp. 1D-CNN) on Joint-TC. Interestingly, Tab. 5.1 also highlights that the training strategy adopted may impact the performance achieved by the models. In-

deed, while for App-TC, all the architectures achieve better performance when relying on App training strategy, for Act-TC, the training App×Act performs better for all architectures. Hence, this highlights that app classification may be conducive to activity recognition, while the opposite does not necessarily hold.

Furthermore, looking at the *complexity of the considered architectures*, the analysis provides other exciting pieces of evidence (last two columns of Tab. 5.1). The complexity highly varies with considered architecture: 1D-CNN and Mimetic-Enhanced are $\approx 10\times$ and $\approx 4\times$ more complex than Hybrid, respectively, in terms of trainable parameters (which are roughly proportional to both the training time and the memory occupation). *Hence,* Mimetic-Enhanced *provides the best trade-off between TC performance and complexity.* To provide details on the performance at a finer grain, Fig. 5.4 reports the *confusion matrix* of each architecture on the Joint-TC task. Delving into these matrices, we can notice that Mimetic-Enhanced can substantially reduce the misclassification patterns w.r.t. both 1D-CNN and Hybrid, confining the errors within the activities of the same app, illustrated by the more evident block diagonal pattern in Fig. 5.4c. Still, the *block-diagonal* pattern of all the confusion matrices confirms *the general difficulty in discriminating adequately among the activities with the given set of inputs.*[1]

[1]This is also confirmed by the confusion matrices on the Act-TC task, not shown for brevity.

Takeaways

RQ: *Are single- and multi-modal state-of-the-art deep learning approaches suitable to classify both apps and user activities?*

While all the considered models excel in effectively solving the App-TC task, their performance fails when dealing with tasks involving user activity classification (i.e., Joint- and Act-TC).

*Among these models, the (*multimodal*)* MIMETIC-ENHANCED *outperforms the (*single-modal*) competing approaches on all TC tasks also providing the best trade-off between TC performance and complexity.*

The confusion matrices also reveal that MIMETIC-ENHANCED reduces misclassifications compared to other models, particularly within activities of the same app. *However, distinguishing between activities remains challenging overall.*

5.2 Context-based Multimodal Deep Learning Traffic Classification

Hereinafter, we introduce the definition of context behavior and related `Context` in Sec. 5.2.1; the traffic characterization of `Context` is then provided in Sec. 5.2.2. Finally, Sec. 5.2.3 describes the novel MIMETIC-ALL architecture proposed herein.

5.2.1 Definition of Context Behavior

Based on the above considerations, in addressing the TC problem at the activity level we intend to exploit *both* the (a) intrinsic characteristics of the biflow to be classified and (b) the information related to its context, namely taking into account the traffic related to the biflows that are contextually created by the same app when the user performs a specific activity. To be able to do this in a practically meaningful and useful manner, we need to clarify the concepts of *classification object* and contextual information.

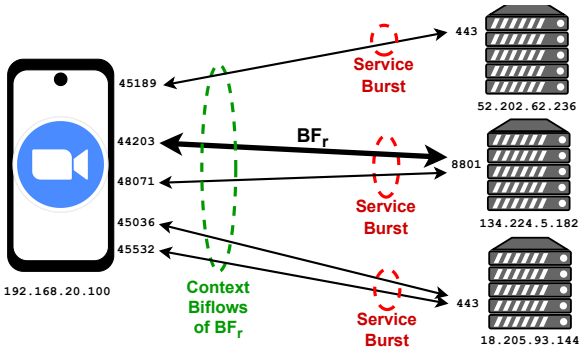


Figure 5.5. Comparison between *context biflows of a reference biflow* BF_r and *service bursts*. All packets are generated or received by the same mobile app.

A classification object represents the (aggregated) unit of traffic that will receive the classification label—and thus will be subject to the management actions that motivated the classification, e.g., it will be throttled, blocked, or given higher priority, or will trigger a special logging rule, etc. Typically, the classification object is also the source of input data for the classifier. This is the case for the biflow, whose information is used to extract classification features and is the typical traffic unit for network management.

Considering the *service burst* (ref. Fig 5.5), defined in Sec. 4.3, and recently used in the context of TC as *classification object* [43, 92, 94], we notice that this aggregate of traffic presents however some significant *limitations*, discussed hereafter. First, the (service) burst definition is highly-sensitive w.r.t. the choice of the inter-packet-time threshold: this value can critically depend on the network conditions and on the presence (or lack thereof) of multiple clients accessing the same service. Secondly, using a service burst as the TC object poses a practical problem with the usage of the classification result: from both network management and network security standpoints, the biflow is a well-known and widely-used granularity to apply relevant actions such as filtering, throttling, priority queuing, and accounting. Conversely, it is not straightforward (or necessarily meaningful) to apply the same actions on varying sets of packets separated by the aforementioned transmission gaps. Finally, service

bursts are unlikely to reflect (and allow to exploit) the heterogeneous nature of modern-application traffic. Indeed, a single application to perform its functions can *at the same time* communicate with different network services [20], and a mix of services is likely to be more indicative of a specific *activity* of the same application. In Secs. 4.3 and 4.4, we experimentally validated the presence of multiple simultaneous biflows and several service bursts that characterize the time evolution and volume of traffic of each app and activity differently. Following these considerations, **we keep the biflow as the TC object**, and we will use the information from simultaneous same-app communications as *contextual information* that will help in discerning the activities of a given application.

Specifically, in what follows, we refer to the TC object as the *reference biflow* or BF_r. We consider the traffic generated by the same device, identified by its IP address[2] hereafter referred to as *device IP*, and the same app generating BF_r: a filter on the *device* IP address and a pre-classification stage detecting the app[3] allows to select all biflows potentially related to BF_r. To restrict the contextual information to only communications simultaneous to BF_r, and to impose *causality* (thus enabling online classification), we further restrict considered packets to biflows that were open during the transmission of the first N_p payload-carrying packets of BF_r. In summary, naming t_r^{start} the arrival time of the `SYN` packet of BF_r, and $t_r^{N_p}$ the arrival time of its $\mathtt{N_p}$-th payload-carrying packet, the set of packets defining *contextual biflows* satisfy *all* the following *four* conditions:

- same *device* IP of BF_r;
- same app label of BF_r;
- biflow did not end before t_r^{start};
- arrival time of the packet precedes $t_r^{N_p}$.

[2]If NAT is performed, the original IP is easily available if the traffic capture happens on the gateway performing NAT or if the log of NAT mapping is provided: both cases are common in enterprise scenarios. If this information is not explicitly available, different approaches to cluster traffic originating from a single device can be applied to passively monitored NATted traffic [95].

[3]For app-level TC, an F-measure greater than 92% can be achieved also just using the single-modality bSEQ classifier fed with informative fields of the first 20 packets (see Tab. 5.3). Fig. 5.3b shows that this is a design choice based on deployment constraints.

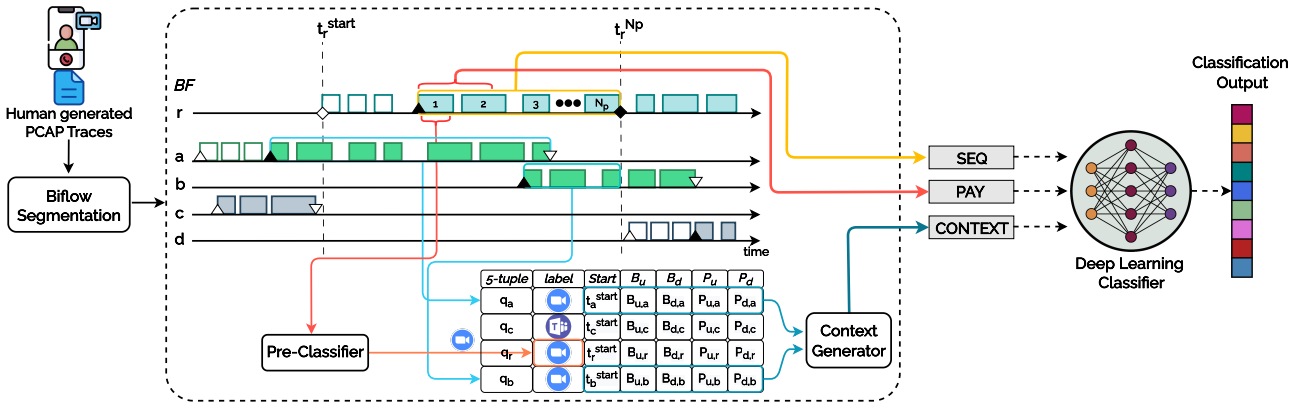


Figure 5.6. For each biflow are highlighted: the arrival time of the first packet (△) (i.e. the `SYN` packet for `TCP` biflows), the arrival time of the last packet (▽), and the arrival time of first packet with *non-zero* payload (▲). Additionally, for the current biflow $\texttt{BF}_\texttt{r}$, in addition to the arrival time of the first packet (◊), the arrival time of the $\texttt{P}^{\texttt{th}}$ packet with *non-zero* payload is also highlighted (⧫). **PAY** denotes the first N_b byte of transport-level payload. **SEQ** denotes header fields extracted from the sequence of the first N_p packets. **CONTEXT** denotes Context inputs computed at the arrival of the N_p-th packet of the reference biflow $\mathbf{BF}_r$ (i.e., the biflow to be classified). $\mathbf{t}_i^{\mathbf{start}}$ refers to the starting time of the generic biflow $\mathbf{B}_i$ identified by the quintuple $\mathbf{q}_i$. $\mathbf{B_{u},}_i/\mathbf{B_{d},}_i$ and $\mathbf{P_{u},}_i/\mathbf{P_{d},}_i$ denote the total amount of byte and packets, respectively, transmitted/received by the biflow $\mathbf{B}_i$, up to time $\mathbf{t}_r^{\mathbf{N}_p}$.

The overall mechanism is illustrated in Fig. 5.6 and described hereafter. For each biflow, the starting time and the current number of bytes/packets transmitted (in both the upstream and downstream directions) are saved. Then, for each reference biflow BF_r, at time $t_r^{N_p}$ (arrival of its $\mathtt{N_p}$-th payload-carrying packet), the packets belonging to its contextual biflows are used to compute *nine* aggregate metrics to be used as *Context inputs* (hereinafter referred as `Context`). Specifically, these metrics correspond to: (a) the number of contextual biflows (#`CF`), (b) the amount of transmitted byte/packets ($\mathtt{V}_*/\mathtt{P}_*$) and (c) the bit-/packet- rate[4] ($\mathtt{BR}_*/\mathtt{PR}_*$) in both directions (we use $* = \mathtt{u}$ and $* = \mathtt{d}$ to denote the *upstream* and *downstream* directions, respectively).

Regarding the practical feasibility of the definition above, we highlight that the `Context` are time-sliced analogous to flow counters kept by routing devices and traffic monitoring middleboxes (*NetFlow* and *IPFIX* standard [96]) and can be directly derived from their values sampled at two instants determined by each reference biflow (the start and the arrival of N_p-th packet).

5.2.2 Analysis of Context Inputs

In this section, we use `Context` previously defined above to characterize the traffic generated when the user performs one of the considered activities (i.e., `ACall`, `Chat`, and `VCall`), also considering the specific app. To this end, in Fig. 5.7, for each activity, we report the average value of each feature calculated in correspondence of the 20^{th} packet of the *reference biflow* as a radar chart. In detail, Fig. 5.7a focuses on activities without taking into account the generating app, whereas Figs. 5.7b- 5.7d provide a drill-down for `Skype`, `Teams`, and `Webex`, respectively.

As shown in Fig. 5.7a, although the (average) number of contextual biflows is similar (#`CF`), the different activities show very different behaviors w.r.t. the other `Context`. In fact, as expected, `VCall` represents the activity originating *the highest contextual traffic for both directions*, in terms of packets and bytes. This is mainly due to the simultaneous transmission of audio and video traffic streams. Also, comparing `Chat` and `ACall`, we

[4]Rates are computed by considering the time interval between the starting of the oldest contextual flow and $t_r^{N_p}$.

notice that the former presents a predominance of downstream traffic (in terms of both volume and byte rate). In contrast, the latter presents a more balanced traffic that stands out, especially for the quantity and rate of packets transmitted in the upstream direction.

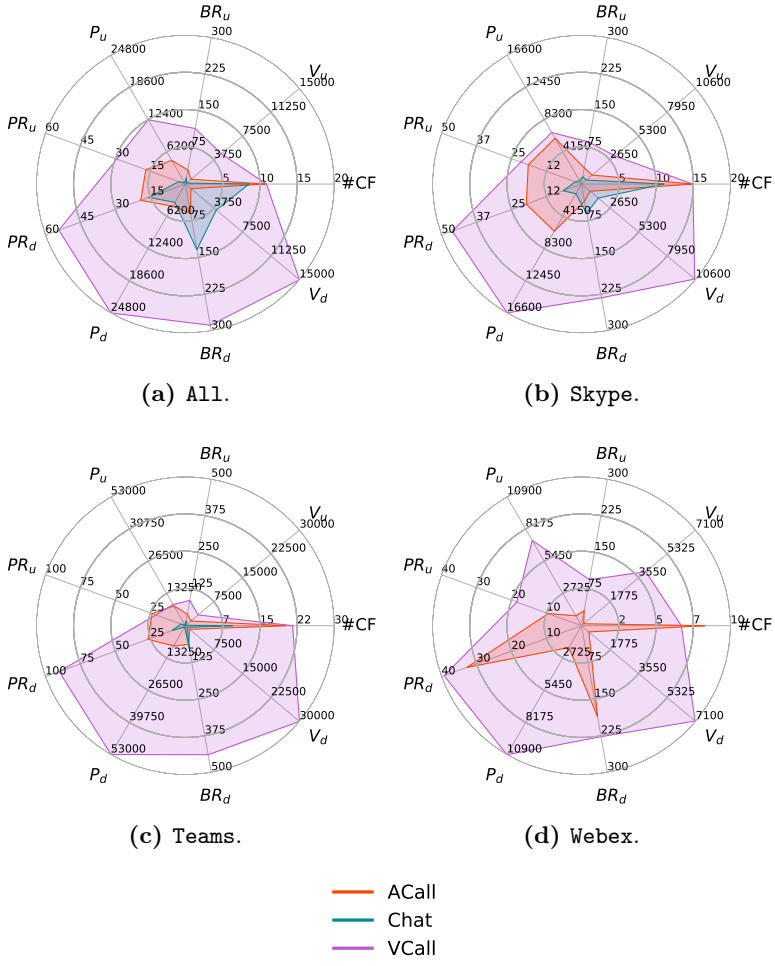
Focusing on the downstream direction, we notice that for both `Skype` and `Teams` (cf. Figs. 5.7b and 5.7c) `VCall` still continues to have the highest amount of traffic both in terms of bytes and packets. Furthermore, for `Webex` (cf. Fig. 5.7d), we notice a similar behavior between `VCall` and `ACall` in terms of packet- and byte-rate, whereas this does not occur when considering the number of packets and byte volume received, which are higher in the case of `VCall`. Moreover, when comparing `ACall` and `Chat` in the case of `Skype` and `Teams`, the two activities differ mostly in terms of packets. In contrast, the difference becomes less evident when considering the byte volume, especially for `Teams`.

On the other hand, by considering the upstream direction, we notice a clear pattern for `Chat` that results in a smaller amount of traffic w.r.t. all the `Context`. Conversely, this does not apply to `ACall` and `VCall`. In fact, for `Skype` and `Teams` (cf. Figs. 5.7b and 5.7c), there is a similarity between the two activities in terms of number of packets transmitted and packet-rate. However, this is not true when looking at the byte volume and the byte rate. Finally, in the case of `Webex` (cf. Fig. 5.7d), a clear distinction between `ACall` and `VCall` w.r.t. all the `Context` is evident.

Takeaways

RQ: *Can `Context` differentiate between different activities within the same CC app?*

`Context` could be crucial in enhancing the performance of activity-level TC. Various activities exhibit very distinct behaviors regarding `Context`.



(a) `All`. **(b)** `Skype`.

(c) `Teams`. **(d)** `Webex`.

Figure 5.7. Properties of biflows' with respect to the *nine* Context inputs (i.e., $\mathtt{V_u}, \mathtt{V_d}, \mathtt{BR_u}, \mathtt{BR_d}, \mathtt{P_u}, \mathtt{P_d}, \mathtt{PR_u}, \mathtt{PR_d}, \#\mathtt{CF}$, arranged symmetrically in the upper half for upstream, lower half for downstream) for `All` apps (a), `Skype` (b), `Teams` (c), and `Webex` (d) based on the activity-type (*ACall*, *Chat*, and *VCall*). Values are calculated at the arrival of the 20^{th} packet for each current biflow and averaged over all biflows. $\mathtt{V_u}$ and $\mathtt{V_d}$ are reported in KB. $\mathtt{BR_u}$ and $\mathtt{BR_d}$ are reported in Kbps. $\mathtt{PR_u}$ and $\mathtt{PR_d}$ are reported in Pps. $\mathtt{P_u}$, $\mathtt{P_d}$, and $\#\mathtt{CF}$ are reported in unit.

5.2.3 Leveraging Context Inputs: the Mimetic-All Architecture

In Sec. 5.1.5, we have shown that Mimetic-Enhanced [29] is able to outperform the considered single-modality classifiers, especially when addressing classification tasks that take into account the activities performed by the users. In this section, taking advantage of the *modularity* offered by the general Mimetic framework [34], we describe the design of Mimetic-All to exploit `Context` discussed in Sec. 5.2.1.

As shown in Fig. 5.8, the proposed Mimetic-All architecture consists in *three per-modality* (viz. input-specific) branches, henceforth named simply bPAY, bSEQ, and bContext (where the initial "b" stands for "branch"). As in the original proposal [29], bPAY and bSEQ branches take as input the first N_b bytes of the transport layer payload (`PAY`) and the informative fields extracted from the sequence of the first N_p packets (`SEQ`), respectively (cf. Sec. 5.1.1 for details on such input types). Additionally, both branches exploit a *trainable embedding layer* to embed each input element into a vector of dimension $e = 10$, resulting in an overall embedding matrix $E \in \mathbb{R}^{N \times e}$—with N denoting the input dimensionality (i.e., N_b and N_p for the `PAY`- and `SEQ`-modality, respectively).

The motivation for applying an embedding to these inputs is due to the categorical nature of `PAY` and `SEQ` and for providing a better representation (i.e., more informative) of the inputs.

Specifically, in the bPAY branch, the corresponding embedding matrix is fed to a sequence of single-modality layers, consisting of two 1D convolutional layers—with a kernel size of 25, unit stride, and 16 and 32 filters, respectively—each followed by a 1D max-pooling layer—with spatial extent of 3 and unit stride—and finally, one dense layer with 256 neurons.

On the other hand, the layers of the bSEQ branch are a bidirectional GRU (BiGRU)—with 64 units and return-sequences behavior—and one dense layer with 256 neurons. All the layers are set with the *Rectified Linear Unit (ReLU)* activation function. Besides the above branches, the newly-added bContext branch takes as input the contextual (aggregated) inputs associated with the N_p-th packet of each reference biflow B_r (`Context`). Indeed, as opposed to `PAY` and `SEQ`, these aggregated inputs do not have a natural ordering or sequentiality. In detail, inspired by the proposal of [42], we have used a Multi-Layer Perceptron (MLP) network

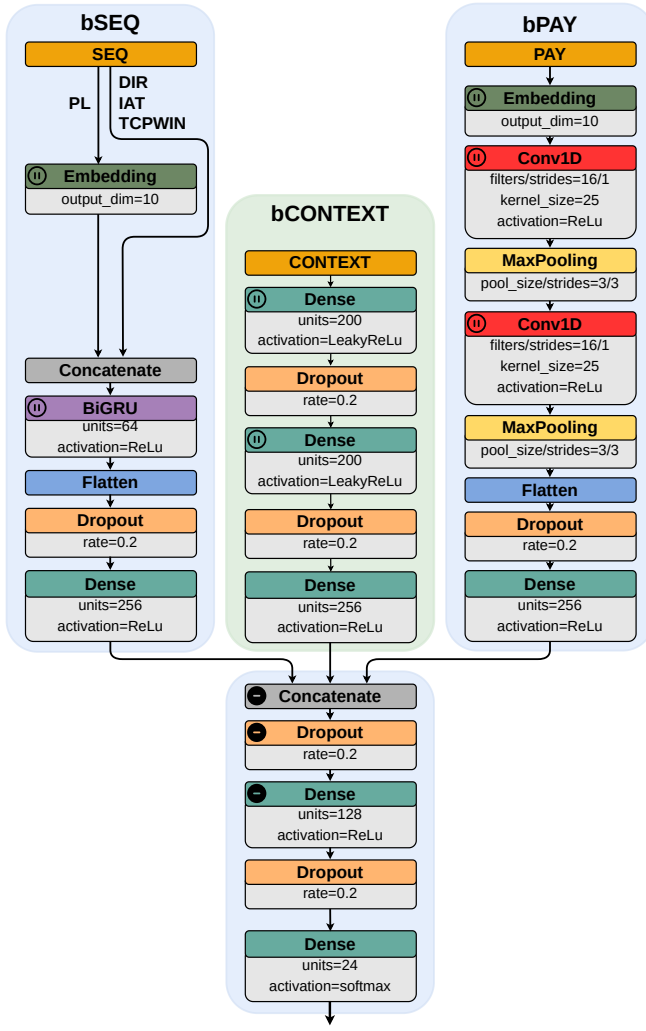


Figure 5.8. Proposed MIMETIC-ALL classifier. The macro-blocks shared with the original MIMETIC-ENHANCED architecture are highlighted in blue. The green macro-block denotes the new branch (i.e., bCONTEXT) dedicated to the novel set of *Context inputs*. For each single block the color filling indicates the type of layer. The ⊖ symbol characterizes the layers that are not part of the architecture when considering the single-modality (bPAY, bSEQ, or bCONTEXT) alone. The ⏸ symbol marks layers frozen during the fine-tuning phase.

to ingest (viz. distill information from) them. This choice is driven by the fact that, unlike `PAY` and `SEQ`, `Context` are calculated in an aggregate form and, having no natural ordering or sequentiality, there is no spatial/time evolution dependence that could be exploited by convolutional/recurrent layers. Specifically, bContext consists of two dense layers—characterized by 200 neurons and a *LeakyReLU* activation function. Like the other two branches, we use a final dense layer with 256 units and a *ReLU* activation function. Finally, the features extracted by the single-modality branches are joined via a concatenation layer and fed to a (dense) *shared representation layer*—with 256 neurons and *ReLU* activation—before performing the classification through a softmax.

As in the original proposal, Mimetic-All is trained via a two-phase procedure consisting of an independent *pre-training* of each single-modality branch followed by a *fine-tuning* of the entire architecture after freezing the lower single-modality layers (i.e., the dense, 1D convolutional, and BiGRU layers, as highlighted in Fig. 5.8 via the ⏸ symbol). In more detail, we performed the pre-training of each branch and the fine-tuning for 30 and 40 epochs, respectively, to minimize respective categorical cross-entropy loss functions via the ADAM optimizer (with a batch size of 50).[5] Additionally, as opposed to the Mimetic-Enhanced proposal [29], preliminary investigation showed that using a fixed learning-rate (i.e., equal to 0.001) instead of an adaptive one results in a performance improvement for Joint-TC.[6] Finally, to improve regularization and mitigate the possible overfitting, we added a dropout of 0.2 at the end of each single-modality branch and after every dense layer. We adopted an early-stopping technique measured on the validation accuracy with the same setup described in Sec. 5.1.2 (i.e., 20% of training data are used for validation).

Remarks on hyperparameter choice: during the design process of the bContext branch, we evaluated several combinations regarding the tuning of the activation functions and the number of neurons of the dense layers.

[5]The single-modality branches, when considered alone (i.e., not in a multimodal configuration), are trained for 100 epochs without dividing the training process into *pre-training* and *fine-tuning*.

[6]In addition, we have also tried a hybrid configuration—i.e., consisting of an adaptive learning rate during the pre-training of `PAY` and `SEQ` and the fine-tuning, and a fixed on bContext—obtaining a lower performance compared to the fixed setting.

We obtained slightly better performance regarding the former by using a *LeakyReLU* on the first two layers and a *ReLU* on the last one. Specifically, the *LeakyReLU* allows for a small, non-zero gradient when the unit is saturated and not active compared with the *ReLU*, alleviating potential problems caused by the hard activation of the latter [97]. However, we discovered that the *LeakyReLU* function is not fully supported by the *SHAP Python library*, which we rely on in Chap. 7 to assess the interpretability of our model. As a result, we decided to use the *ReLU* function on all layers of the bCONTEXT branch.

Conversely, trying different configurations[7] we obtained the best trade-off—between performance and complexity—by setting 200 neurons on the first two dense layers. Moreover, we noticed better performance when considering the same number of neurons (i.e., 256) in the final dense layers of the single modalities before concatenation.

5.3 Experimental Evaluation

Table 5.2. Variants of classifiers used in this work with respective input data used to feed them.

Variant	PAY	SEQ	Context
bPAY	✓	—	—
bSEQ	—	✓	—
bCONTEXT	—	—	✓
MIMETIC-ENHANCED	✓	✓	—
MIMETIC-CONPAY	✓	—	✓
MIMETIC-CONSEQ	—	✓	✓
MIMETIC-ALL	✓	✓	✓

`PAY`: First N_b bytes of transport-layer payload;
`SEQ`: Informative header fields extracted from the sequence of the first N_p packets;
`Context`: Context inputs computed at the arrival of the N_p-th packet.

As shown in Sec. 5.1.1, the sole combination of biflow transport-layer

[7] We tested 32, 64, 128, 200, and 256 neurons.

payload (`PAY`) and packet-level (`SEQ`) inputs within Mimetic-Enhanced classifier *is not sufficient to guarantee adequate performance* when dealing with Joint-TC and Act-TC tasks. Thus, in Sec. 5.2, we identified a new set of inputs—which we named `Context`—suitable to distinguish traffic associated with different activities performed with the same app and proposed the Mimetic-All classifier leveraging them (Sec. 5.2.3). In this section, we evaluate the impact of this context-based modality on TC performance when adopting the App×Act training strategy. The latter strategy allows tackling *all the considered classification tasks* (i.e., App-TC, Act-TC, and Joint-TC).

For the sake of a complete evaluation and aiming at performing a *reasoned ablation study*, other than Mimetic-Enhanced and Mimetic-All, in the following analysis, we investigate the performance of classifiers obtained by selecting a subset of the three modalities available. For readers' convenience, in Tab. 5.2, we report the different instances drawn from the Mimetic framework according to the considered type of input (i.e., `PAY`, `SEQ`, and `Context`).

5.3.1 Are `Context` suitable for joint App and Activity classification?

In Tab. 5.3, we report the performance of DL-based traffic classifiers combining different mixes of modalities, namely when considering both single-modality branches alone—each fed with one of the three different input types `PAY`, `SEQ`, and `Context`—and their combinations. The following results are obtained considering the best-performing $N_b = 576$ B and $N_p = 20$ packets.

As shown, performance varies depending on the considered type of input and TC task.

In detail, the suitability of `Context` inputs discussed in Sec. 5.2.2 is demonstrated by the fact that considering only the bContext branch results in a significant performance improvement on the Act-TC task compared to Mimetic-Enhanced (+9% F-measure). With the ability of Mimetic-Enhanced to discriminate apps and the modularity of the multimodal architecture being given, these results suggest that the addition of `Context` inputs can lead to a classifier that is able to achieve good performance for all the considered tasks simultaneously.

Table 5.3. Accuracy, F-measure, number of Trainable Parameters (#TP), and Training Time (Time) comparison of DL-based traffic classifiers when combining different mixes of modalities. Models are trained using APP×ACT class labels. Results are in the format *avg.* (±*std.*) obtained over 10-folds. For each metric (column), the best and the worst architectures are highlighted in green and red, respectively. Training time is calculated by pre-training the individual modalities in parallel. The input types fed to each classifier are shown in Tab. 5.2.

Classifier	Joint-TC		App-TC		Activity-TC		#TP [k]	Time [*min*]
	Accuracy [%]	F-measure [%]	Accuracy [%]	F-measure [%]	Accuracy [%]	F-measure [%]		
bPAY	42.55(±1.15)	36.16(±1.49)	76.54(±0.51)	77.46(±0.53)	53.80(±0.98)	46.71(±1.94)	460	38 (±8)
bSEQ	66.36(±1.01)	61.52(±0.89)	97.39(±0.21)	97.98(±0.18)	67.84(±1.05)	66.24(±0.98)	706	32 (±7)
bCONTEXT	67.61(±1.18)	66.10(±1.40)	82.52(±1.30)	82.38(±1.19)	79.42(±0.83)	78.21(±0.96)	99	2 (±1)
MIMETIC-ENHANCED	70.17(±0.64)	66.10(±0.60)	99.02(±0.08)	99.19(±0.08)	70.73(±0.63)	69.16(±0.60)	1235	80 (±15)
MIMETIC-CONPAY	77.72(±1.18)	76.68(±1.28)	94.84(±0.50)	94.87(±0.56)	81.01(±1.05)	80.22(±1.09)	628	44 (±5)
MIMETIC-CONSEQ	80.45(±0.73)	79.52(±0.98)	98.49(±0.26)	98.80(±0.19)	81.42(±0.62)	80.79(±0.78)	875	30 (±3)
MIMETIC-ALL	81.60(±0.89)	79.96(±0.93)	99.22(±0.13)	99.34(±0.12)	82.07(±0.85)	81.25(±1.03)	1368	78 (±11)

Therefore, the impact on performance when combining `Context` with `PAY` and `SEQ` is evaluated. Considering the classification performance of Mimetic-Enhanced as a reference, combining `Context` and `PAY` (Mimetic-ConPay) results in an increase of $\approx$+11% in terms of F-measure for both Joint- and Act-TC tasks ($\approx$+8% and $\approx$+10% of accuracy, respectively). Unfortunately, the same combination results in a non-negligible decrease of $\approx -4\%$ of both F-measure and of accuracy for the App-TC.

In contrast, combining `SEQ` and `Context` (Mimetic-ConSeq), we obtain a performance increase for both Joint- and Act-TC—i.e., $\approx +13\%$ (resp. $\approx +12\%$) and $\approx +10\%$ (resp. $\approx +11\%$) in terms of F-measure (resp. accuracy)—w.r.t. Mimetic-Enhanced. Also, in this case, the above combination results in a small decrease in performance for App-TC (i.e., $\leq -0.5\%$ for both F-measure and accuracy). Notably, it is possible to ignore the payload content (i.e., `PAY`) by using the latter input combination. This peculiarity makes this solution suitable also for future scenarios where more opaque encryption sublayers are deployed (e.g., extensions for *Encrypted Server Name Indication* and *Encrypted Client Hello* in TLS 1.3 [30]), likely hindering classification solutions leveraging payload (see [29] for an analysis of DL usage of payload-based inputs). Finally, combining the *three input types* (Mimetic-All), this model can outperform the other configurations for all the considered TC tasks.

Overall, in agreement with the results already discussed in Sec. 5.1.1, leveraging the bPAY and bSEQ branches individually leads to a performance degradation for all TC tasks, compared to Mimetic-Enhanced. In fact, when considering the `PAY` input only (i.e., the bPAY branch), we incur the worst performance, which corresponds to a loss between -22% (resp. -17%) and $\approx -30\%$ (resp. $\approx -28\%$) in terms of F-measure (resp. accuracy). On the contrary, when we consider only the `SEQ` input (i.e., the bSEQ branch), we obtain a more substantial loss regarding the Joint-TC task ($\approx -5\%$ F-measure) while for App- and Act-TC tasks the loss is more contained (i.e., $\approx -1\%$ and $\approx -3\%$ F-measure, respectively).

More in detail, Fig. 5.9a depicts the Act-TC performance of Mimetic-Enhanced (i.e., the state-of-art baseline), conditioned on each application. To further investigate the results obtained by combining `Context` with the other inputs in Figs. 5.9b–5.9d we report the *gain/loss w.r.t.* Mimetic-Enhanced at the activity granularity (conditioned on each app).

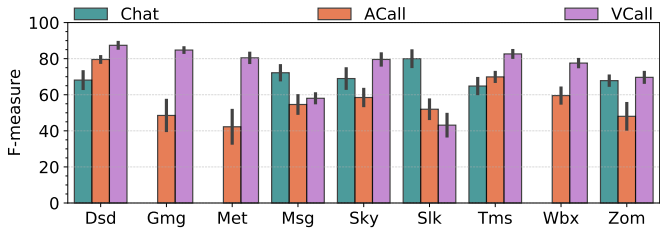
As depicted in Fig. 5.9a, for most applications (6 out of 9), `VCall` shows to the best classification performance (i.e., ≈ 80% of F-measure). This does not hold for `Messenger`, `Slack`, and `Zoom`, which result in the range 40%–70% of F-measure. On the contrary, in the case of `Chat` ≈65% F-measure is achieved, regardless of the considered app (except for `Slack`). Finally, for `ACall` the performance varies strongly depending on the app between ≈40% (for `Meet`) and 80% (for `Slack`).

The addition of `Context` results in a gain in terms of F-measure that depends on the specific combination of inputs used and varies with the app and the activity (Figs. 5.9b–5.9d). Moreover, while MIMETIC-CONSEQ and MIMETIC-ALL exhibit approximately the same average performance, Figs. 5.9c and 5.9d witness that the gains obtained strongly depend on the specific app. Indeed, MIMETIC-ALL results in higher gains than MIMETIC-CONSEQ on the activities of `Skype`, `Teams`, and `Webex`, but lower performance improvements are observed for some of the remaining apps (i.e., `Discord`, `Messenger`, `Slack`, and `Zoom`). Finally, the worst performance of MIMETIC-CONPAY compared to MIMETIC-CONSEQ and MIMETIC-ALL is mainly due to a general lower gain for the activities of all the apps, especially regarding `VCall` for `Teams` and `ACall` for `Meet`, `Teams`, and `Webex`. Finally, it is worth noting that for `Teams`, as opposed to MIMETIC-ALL, both MIMETIC-CONPAY and MIMETIC-CONSEQ bring no improvement w.r.t. MIMETIC-ENHANCED for the `VCall` activity.

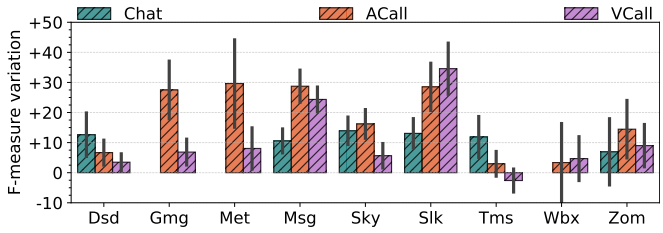
Takeaways

RQ: *How does `Context` impact the performance of the joint app and activity classification?*

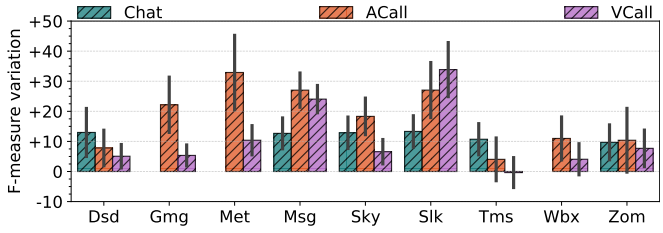
The novel set of `Context` is thus useful when dealing with the classification of both the app and the activity performed by the user. Also, results prove that the combination with the traffic inputs commonly used in the literature at the biflow level (e.g., the first N_b byte of the transport level payload and/or features extracted from the first N_p packets) is fruitful.



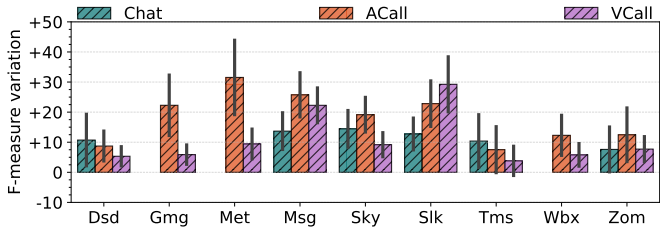
(a) Mimetic-Enhanced.



(b) Mimetic-ConPay.



(c) Mimetic-ConSeq.



(d) Mimetic-All.

Figure 5.9. Per-activity performance for each app achieved by Mimetic-Enhanced (i.e., the basic configuration) (a). Gain/loss w.r.t. Mimetic-Enhanced obtained by considering the *Context inputs* and related to Mimetic-ConPay (b), Mimetic-ConSeq (c), and Mimetic-All (d).

5.3.2 How does MIMETIC-ALL perform compared with ML Traffic Classifiers?

In this section, to show the effectiveness of our proposal, we compare MIMETIC-ALL with two ML algorithms commonly used in the state of the art (see Sec. 2.2), namely the Decision Tree (DT) and the Random Forest (RF). Specifically, ML algorithms have been trained and evaluated using `scikit-learn` Python Framework, adopting predefined hyper-parameters and using 100 estimators for the RF classifier. The aim of the analysis is also to assess separately the benefit of (*a*) the Context Inputs and (*b*) the proposed multimodal architecture (being able to process them wisely and thus achieve satisfactory TC performance). Indeed, unlike MIMETIC-ALL, for ML-classifiers the absence of multi-modality implies that the different inputs cannot be handled separately. Therefore, the three inputs have been treated as a single (monolithic) input obtained from their concatenation.

Table 5.4 reports the overall performance of models, trained using APP×ACT class labels and *using all input types* (i.e., `PAY`, `SEQ`, and `Context`). An assessment of computational complexity complements the table. This is done via the memory requirements ("Size")[8], training time ("Time"), and the number of parameters ("Params.") columns. It is worth noting that, in the case of MIMETIC-ALL, since the training methodology adopted allows parallelizing the pre-training of the two modalities, we have calculated the resulting training time by adding the time needed to perform the pre-training of the "slower" modality and the time needed to perform the successive fine-tuning of the whole architecture. As shown, DT results in the worst performance w.r.t. all classification tasks. Specifically, the highest gap against MIMETIC-ALL concerns Joint- and Act-TC tasks, which corresponds to a worsening of approximately −14% (resp. −14%) and −12% (resp. −12%) in terms of F-measure (resp. Accuracy), respectively. Furthermore, the gap is reduced to roughly −3% in terms of both F-measure and Accuracy w.r.t. App-TC task. Finally, comparing RF with MIMETIC-ALL, while the performance related to App-TC is quite similar, there is a loss of up to 2% (resp. 1%) regarding Joint-TC and Act-TC in terms of both F-measure (resp. Accuracy).

Finally, looking at the complexity of the considered model (last three

[8]Sizes do not account for gzip, zlib, etc., compression thus roughly reflect the memory required to load those models for inference.

Table 5.4. Accuracy and F-measure comparison of DL-based Mimetic-All against ML-based traffic classifiers when using all input types (i.e., `PAY`, `SEQ`, and `Context`). Models are trained using App×Act class labels. Results are in the format *avg.* (±*std.*) obtained over 10-folds. For each metric (column), the best and the worst models are highlighted in green and red, respectively. The training time of Mimetic-All is calculated by pre-training the individual modalities in parallel. Size refers to the file size obtained as a result of pickle serialization. For DT and RF number of parameters is reported in the form *total nodes*/*avg. depth* while for Mimetic-All it refers to the number of trainable parameters.

Classifier	DL	Joint-TC		App-TC		Act-TC		Size [MB]	Time [*min*]	Params. [k]
		Accuracy [%]	F-measure [%]	Accuracy [%]	F-measure [%]	Accuracy [%]	F-measure [%]			
DT	○	67.96 (±1.15)	66.05 (±1.36)	95.91 (±0.62)	96.35 (±0.56)	70.12 (±1.15)	68.89 (±1.34)	1.8	0.013	7.6/32
RF	○	80.57 (±0.74)	77.94 (±0.86)	98.92 (±0.14)	99.14 (±0.12)	81.33 (±0.79)	80.41 (±0.82)	131.8	0.57	557/30
Mimetic-All	●	81.60 (±0.89)	79.96 (±0.93)	99.22 (±0.13)	99.34 (±0.12)	82.07 (±0.85)	81.25 (±1.03)	11.8	78	1368

columns of Tab. 5.4), we noticed that although RF performs similarly to MIMETIC-ALL, the memory requirement (viz. size) of the resulting model is 11× larger. This outcome supports the observations made in [98] where it was highlighted that ML algorithms are *unrealistic upper bounds* since they do not scale up when dealing with more complex TC problems (i.e., increased number of classes and larger training data). However, we found that MIMETIC-ALL requires more training time when compared to both DT and RF. Therefore, we can conclude that MIMETIC-ALL provides the best balance between TC performance and complexity.

Takeaways

RQ: *How does* MIMETIC-ALL *perform compared with state-of-the-art ML Traffic Classifiers?*

MIMETIC-ALL *is able to outperform both ML and DL algorithms when addressing the joint traffic classification of the app and activity performed by the user. Although* MIMETIC-ALL *requires more training time, it represents the best trade-off in terms of performance and complexity.*

Chapter 6

Fine-Grained Traffic Prediction using Multitask Deep Learning

In this chapter, we address packet-level network traffic prediction using multitask DL. The workflow we employ to address this prediction problem is depicted in Fig. 6.1.

In detail, in Sec. 6.1, we provide a formal statement of the prediction task we have tackled and the corresponding solution we have devised. Hence, the section includes: (a) the description of the traffic parameters of interest; (b) the specification of the multitask DL architectures used to predict them; (c) the details about the adopted procedures for training these architectures. Then, in Sec. 6.2, we provide a description of the de-

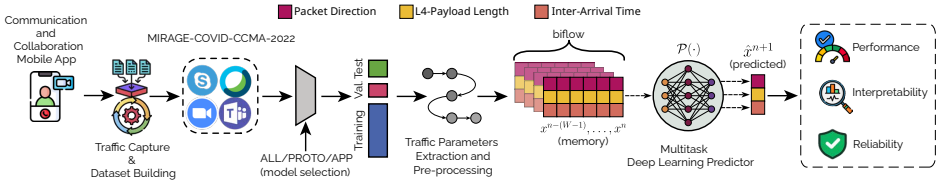


Figure 6.1. Workflow of the methodology defined for predicting the traffic of *communication-and-collaboration mobile apps* via multitask DL approaches and explaining/assessing the forecasting outcomes via XAI techniques.

vised procedure to forecast traffic at a coarser (but arbitrary) granularity by exploiting packet-level predictions. In Sec. 6.3, we detail the experimental setup employed for assessing the performance of the prediction models. Finally, in Sec. 6.4, we comprehensively evaluate our methodology by analyzing traffic prediction performance on a subset of *four* popular CC apps from different perspectives.

6.1 Multitask Deep Learning for Packet-Level Traffic Prediction

Our goal is to utilize DL approaches to predict network traffic generated by CC apps at the *finest granularity*, i.e., at the packet level. To achieve this, we employ the widely-used bidirectional flow (biflow) as *traffic object* for our prediction task. A biflow embodies all the packets that share the same 5-tuple `(IP src, IP dst, port src, port dst, protocol)` in both upstream and downstream directions [23].

Predicting traffic at the packet level is the finest granularity for predicting traffic and is still a challenging problem. A model that can predict at the packet and the biflow level allows for both spatial and temporal flexibility. Indeed, in the former case, packet-level prediction can be used to aggregate based a given activity, app, protocol (UDP or TCP), or server-side. In the latter case, different temporal granularities (as shown in Sec. 6.2) can be obtained from the same model by only changing some parameters at runtime.

Specifically, given a biflow up to its n^{th} packet, we aim to predict P *traffic parameters* associated to the $(n{+}1)^{th}$ packet. In this case, the desired *output* of our DL architecture is represented by the vector $\boldsymbol{x}^{n+1}$. These predictions are based on the previous values of the same traffic parameters, which are stored in a *memory window* of size W. Then, the observations $\boldsymbol{x}^{n}, \ldots, \boldsymbol{x}^{n-(W-1)}$ are used as *input* to the DL architecture.

It is important to note that we construct the input using an *incremental windowing* approach that utilizes a sliding memory window with a unit stride. This allows us to incrementally add samples to the window until it reaches the maximum size of W. In cases where the prediction task has accumulated memory that is less than or equal to W, we apply left zero-padding to reach the desired window size. This enables predictions

to be made on the initial part of the biflow (see later Sec. 6.4.3) and on biflows shorter than W.

Furthermore, we leverage *multitask architectures* that simultaneously address multiple prediction tasks, with each task focusing on one of the P parameters under consideration[1]. Therefore, we are targeting the design of a *single* DL model in the form:

$$\hat{\boldsymbol{x}}^{n+1} = \mathscr{P}(\boldsymbol{x}^{n}, \boldsymbol{x}^{n-1}, \ldots, \boldsymbol{x}^{n-(W-1)}). \tag{6.1}$$

where $\hat{\boldsymbol{x}}^{n+1}$ denotes the prediction vector associated to $\boldsymbol{x}^{n+1}$.

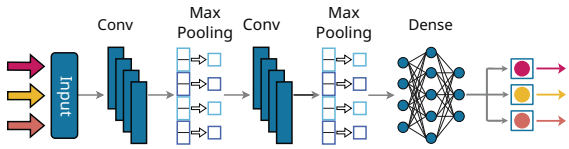
6.1.1 Traffic Parameters

We aim to predict three traffic parameters ($P = 3$) for the packets within a biflow. These parameters are: (i) the *direction* (`DIR`), which is a binary value indicating whether the packet is in the downstream or upstream direction; (ii) the *payload length* (`PL`), representing the size of the transport-layer payload measured in bytes; (iii) the *inter-arrival time* (`IAT`) refers to the time interval between the arrival of two consecutive packets. Herein, we focus on predicting such $P = 3$ parameters since most network performance problems (e.g., loss, delay, jitter) generally occur at the packet level. Therefore, predicting the size, direction, and arrival time of the next packet can improve resource allocation within buffers and bandwidth [13]. These aspects are crucial for ensuring the quality of service and a seamless communication experience, mainly when dealing with CC apps [83].

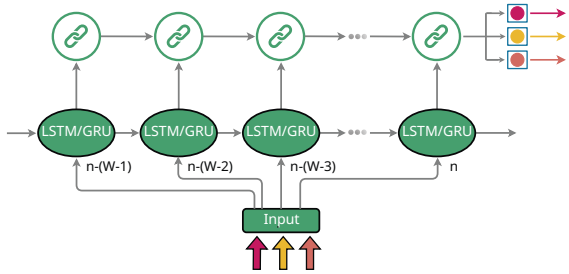
6.1.2 DL Architectures

We utilize three categories of DL models: convolutional, recurrent, and a combination of both. The hyperparameters for these models are determined based on the current state-of-the-art research [23, 27, 99].

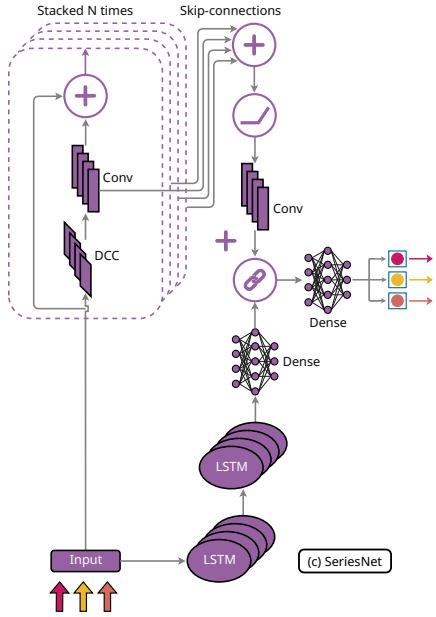
[1] A study by Montieri et al. [23] showed that multitask architectures outperformed single-task solutions in prediction performance and complexity. The results showed that multitask architectures outperformed single-task solutions in both aspects. Specifically, the multitask architectures exhibited better prediction performance, yielding more accurate results and reducing the overall training time. This finding suggests that employing a single deep learning architecture to predict each traffic parameter individually is less effective and more computationally expensive than utilizing multitask architectures.



(a) CNN



(b) RNN(GRU/LSTM)



(c) SeriesNet

Figure 6.2. CNN (a), RNN (GRU/LSTM) (b), and SeriesNet (c) multitask architecture. All DL architectures end with a number of dense layers (aka *heads*) equal to the number of parameters to be predicted (i.e., $P = 3$).

The DL architectures employed are depicted in Fig. 6.2. We underline that all DL architectures end with a number of dense layers (aka *heads*) equal to the number of parameters to be predicted (i.e., P, with $P = 3$ in Fig. 6.1), each using a Sigmoid activation function.

More in detail, for the convolutional model, we adopt a *1D-CNN* architecture (see Fig. 6.2a). It consists of two sequential convolutional layers with 32 and 64 filters, respectively, and a kernel size of 5. Each convolutional layer is followed by a max-pooling layer with a pool size of 3. The architecture further includes a dense layer with 128 neurons. All these layers employ the Rectified Linear Unit (ReLU) activation function.

On the other hand, we also consider two recurrent architectures: a Gated Recurrent Unit (*GRU*) and a Long Short-Term Memory (*LSTM*) network (see Fig. 6.2b). Both recurrent models are unidirectional and consist of 200 units, where the activation function employed is the Sigmoid.

In addition to the individual convolutional and recurrent architectures, we also employ a composite architecture, namely an extended version of the *SeriesNet* (see Fig. 6.2c). This DL model is based on Dilated Causal Convolution (DCC). More in detail, the overall architecture comprises 7 DCC layers. Each DCC has 32 filters, a dilation size of 2, and applies the Scaled Exponential Linear Unit (SELU) activation function. Furthermore, each DCC includes a residual connection that connects the input to the output. The last 2 DCC layers incorporate a dropout rate of 0.8. The sum of the parameterized skip connections from each DCC is passed through a ReLU activation function and used as input to a 1×1 convolutional layer. Finally, the output of this layer is concatenated with the stacking of two LSTM layers, each containing 200 units and a dense layer with 128 neurons. Although we restrict our analysis to the aforementioned DL models, we remark that *the methodology devised in this Thesis (i.e., multitask packet-level prediction and tunable coarser-grained traffic prediction) is quite general and can be straightforwardly extended to other more sophisticated prediction models.*

6.1.3 Loss Specification and Training Details

Our main objective is to predict P traffic parameters for the next packet, which are stored in the vector $\boldsymbol{x}^{n+1}$. Consequently, the DL architecture is trained to minimize a *weighted* sum of the losses associated

with the P prediction tasks considered, namely:

$$\mathscr{L}\left(\boldsymbol{\theta}_{\text{shared}}, \{\boldsymbol{\theta}_p\}_{p=1}^{P}\right) \triangleq \sum_{p=1}^{P} \lambda_p \mathscr{L}_p\left(\boldsymbol{\theta}_{\text{shared}}, \boldsymbol{\theta}_p\right). \tag{6.2}$$

where the weight λ_p indicates the importance level of the p^{th} task in the overall multitask objective function[2]. The shared parameters $\boldsymbol{\theta}_{\text{shared}}$ are associated with the layers that are common to all tasks, while the parameters $\boldsymbol{\theta}_p$ are *specific* to the p^{th} task. Moreover, due to the multitasking nature of these architectures, the specific loss function $\mathscr{L}_p(\cdot)$ to optimize depends on the prediction task being addressed. Specifically, we train the DL architectures to minimize the *binary cross-entropy* loss function for predicting the binary `DIR`. For the prediction of non-binary parameters, such as `PL` and `IAT`, we minimize the *Mean Squared Error (MSE)* between the predicted values and the actual traffic parameters associated with the $(n+1)^{th}$ packet of a given biflow. We selected the MSE because it aligns better with the Internet traffic prediction context and because of its simplicity. In fact, MSE is a quadratic function that penalizes larger errors more significantly, which is useful in practical settings where network resources are allocated based on the values provided by the predictor. In such cases, larger errors can cause under or over-provisioning of resources, with under-provisioning being the most unfavorable scenario, as it can significantly impact the overall performance of the network. Moreover, unlike other complex loss functions such as Huber and Quantile, which require tuning of additional hyper-parameters, MSE does not introduce any new parameters. In this study, we employ different training strategies that vary based on how traffic information is grouped. These strategies allow the models to capture traffic characteristics beyond those specific to an individual app, enabling a more comprehensive understanding of the traffic patterns. Notably, such strategies directly affect the number and complexity of DL models a network operator needs to develop, train, and deploy in the network. As a consequently, we examine the following *three training strategies* corresponding to different aggregation levels:

[2]Based on the optimization of the weights carried out in [23], herein we set $\lambda_1 = \lambda_2 = 0.45$ and $\lambda_3 = 0.10$, where $p = 1$, $p = 2$, and $p = 3$ are associated to the `DIR`, `PL`, and `IAT` prediction task, respectively.

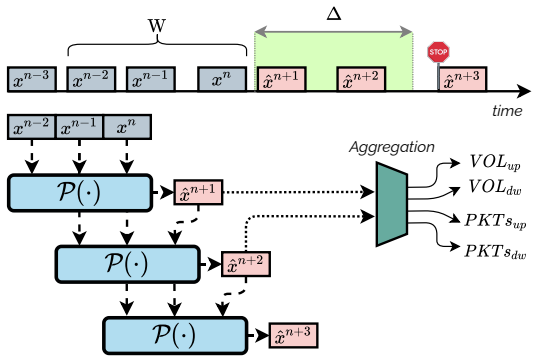
- **per-app models (`APP`)**: a separate model is created for each app. This means that there is a specific predictor $\mathscr{P}(\cdot)$ associated with each app. The models are trained using traffic data labeled with the corresponding app information (e.g., the Android package name).
- **per-proto models (`PROTO`)**: the models are designed to consider traffic based on the protocols used. However, the models do not differentiate between individual apps within each protocol.
- **overall-model (`ALL`)**: a single model is created to encompass all the apps. In other words, a single predictor $\mathscr{P}(\cdot)$ is utilized for all the apps. The model is trained using the entire traffic dataset without considering the specific information about the individual app (or protocol) that generated the traffic.

We remark that the training strategies described do not pose any constraint on the model used for traffic prediction. Hence, each of them can be applied to all the DL traffic predictors considered in this work. However, note that to be practically deployed, per-app models require the presence of an upstream traffic classifier to properly route network traffic to the proper prediction model, namely a more complex network setup compared to the other training strategies.

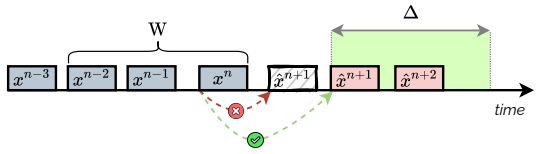
6.2 From Fine-Grained To Aggregate Traffic Prediction

This section defines a first approach to capitalize on the benefits of having a multitask packet-level predictor.

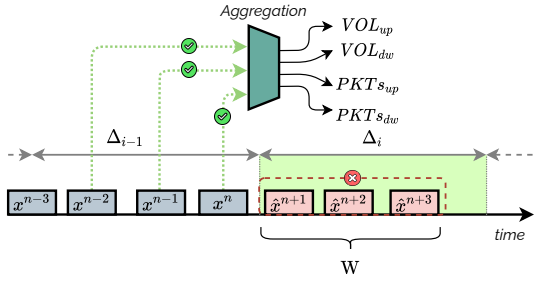
Specifically, our goal is to exploit the packet-level predictions to forecast traffic aggregates within a future time interval (Δ), also known as the time horizon. To achieve this, we employ the *recursive* methodology depicted in Fig. 6.3, which iteratively utilizes the packet-level predictions (i.e., related to `DIR`, `PL`, and `IAT`) to predict the traffic to a coarser granularity (i.e., the number of packets and data volume in both upstream and downstream directions) for the next Δ. This is obtained by *repeatedly* using the fine-grained predictor $\mathscr{P}(\cdot)$. It is worth noting that the main strength of this approach lies in its *flexibility*, which makes it possible to



(a) Base Procedure.



(b) Correction on fine-grained prediction.



(c) Correction on coarse-grained prediction.

Figure 6.3. Proposed *recursive* approach we used to predict aggregates of traffic (i.e., number of packets and traffic volume in both upstream and downstream directions) over a time horizon Δ: we exploit a fine-grained (i.e., at the packet level) predictor $\mathscr{P}(\cdot)$ with a memory window of size $W(=3$ in the figure) (a). Correction on fine-grained prediction (b): the process ensures that all predicted packets fall within the prediction interval Δ. Correction on coarse-grained prediction (c): the process assumes that the traffic in the next Δ_i is the same as that in the previous Δ_{i-1} when the memory (of size W) consists of only predicted packets.

tune the Δ parameter at the operational stage (i.e., at run-time). This avoids the burden of learning a DL traffic predictor for each Δ of interest.

More in detail, in our procedure (depicted in Fig. 6.3a), after each Δ, we take the sequence of the most recent W packets $\{\boldsymbol{x}^n, \boldsymbol{x}^{n-1}, ..., \boldsymbol{x}^{n-(W-1)}\}$, and recursively obtain a series of predictions by means of $\mathscr{P}(\cdot)$, denoted as $\hat{\boldsymbol{x}}^j$, such that: $\sum_j \hat{x}^j_{\texttt{IAT}} \leq \Delta$. To do this, due to the fixed memory of $\mathscr{P}(\cdot)$, at step $j > 1$, the prediction $\hat{\boldsymbol{x}}^j$ is obtained by removing the oldest packet from memory and adding the prediction obtained at the previous step $\hat{\boldsymbol{x}}^{j-1}$ as the last observed packet.

In addition, to handle any critical situations that might occur, we apply two corrections to both fine- and coarse-grained predictions. Correcting the fine-grained predictions (depicted in Fig. 6.3b) ensures that all predicted packets fall within the next Δ. To this end, we enforce that the first predicted packet ($\hat{\boldsymbol{x}}^{n+1}$) arrives at the beginning of the next Δ interval if the corresponding predicted `IAT` would erroneously place such packet before the beginning of the interval. On the other hand, with the correction on the coarse-grained predictions (depicted in Fig. 6.3c), we avoid the error accumulation of the recursive procedure when leveraging a memory encompassing only predicted packets. Accordingly, in such cases, the predicted aggregate traffic is taken as that observed in the previous Δ_{i-1} interval. At the end of the above procedure, the set of `PL`s and `DIR`s of the predicted packets are used to compute the volume and number of packets in both upstream and downstream directions (shown as *Aggregation* in Fig. 6.3). In the following, we refer to these predicted traffic aggregates as $\texttt{VOL}_{\texttt{up}}$, $\texttt{VOL}_{\texttt{dw}}$, $\texttt{PKTs}_{\texttt{up}}$, and $\texttt{PKTs}_{\texttt{dw}}$.

6.3 Experimental Setup

In this section, we provide a comprehensive overview of the experimental setup, including details about the apps' and related activities' selection rationale (Sec. 6.3.1), pre-processing operations on the collected dataset (Sec. 6.3.2), and the evaluation metrics (Sec. 6.3.3) employed for assessing the performance of the prediction models.

6.3.1 Apps' and Activities' Selection Rationale

Nowadays, CC apps, such as those used for business meetings, classes, and social interaction, are massively exploited in everyday life after their adoption was fueled due to the pandemic years [1]. Therefore, we have specifically selected a subset of *four* CC apps from the MIRAGE-COVID-CCMA-2022 dataset based on their popularity [100]: `Skype`, `Teams`, `Webex`, and `Zoom`. As mentioned in Sec. 3, our experimentation focused on specific activities associated with these apps, including:

- Audio-call (`ACall`): a two-way audio transmission between two participants without any video component.
- Chat (`Chat`): a conversation between two participants, exchanging textual messages and/or multimedia content such as images or GIFs.
- Video-call (`VCall`): multiple attendees who can transmit both video and audio; this category encompasses various scenarios, including video calls between two or more attendees and webinars or live events with multiple participants.

By selecting these CC apps and their corresponding activities, we aim to capture and analyze the usage patterns and network traffic characteristics associated with different modes of communication.

6.3.2 Data Processing

In the following analyses, we utilize real traffic from the four apps as provided by the MIRAGE-COVID-CCMA-2022 dataset. We perform some preprocessing steps to obtain the P packet parameters from the raw packet sequences. Firstly, we eliminate packets with zero payloads. Next, we recalculate the `IAT`s, ensuring a minimum precision of 1μs. To remove the influence of outliers, we saturate the recalculated `IAT`s to the 99^{th}-percentile value, which is determined as 197.90 ms based on their distributions. Finally, we apply a min-max scaler to normalize `PL` and `IAT` within the range of $[0, 1]$. This scaling procedure is a common practice when utilizing DL models, as described in [23]. Starting from the collected traffic, we applied the incremental windowing approach, described in Sec. 6.1 to construct the input for the DL architectures. More in detail,

the resulting dataset consists of 806 k samples for `Skype`, 1.9 M samples for `Teams`, 2.5 M samples for `Webex`, and 1.2 M samples for `Zoom`.

6.3.3 Evaluation Procedure and Metrics

The evaluation of traffic prediction strategies is conducted using a robust stratified five-fold cross-validation setup: 80% of the biflows are allocated for the training/validation set, with 80% of this subset being the actual training set and 20% assigned as the validation set; the remaining 20% of biflows are designated as the test set for evaluation purposes. The validation set is utilized to implement the early-stopping technique effectively, which helps prevent overfitting during training. Consequently, we calculate the average value and standard deviation of each evaluation metric over the five folds.

To assess the prediction performance of the binary `DIR` traffic parameter, we employ the *G-mean* metric:

$$\text{G-mean} \triangleq \sqrt{\rho_{dir}^{\text{dw}}\, \rho_{dir}^{\text{up}}}. \tag{6.3}$$

where $\rho_{dir}^{\text{dw}} \triangleq \Pr(\hat{x}_{dir} = \text{DW} \mid x_{dir} = \text{DW})$ and $\rho_{dir}^{\text{up}} \triangleq \Pr(\hat{x}_{dir} = \text{UP} \mid x_{dir} = \text{UP})$, and the variable x_{dir} represents the sequence of actual `DIR`s, while the variable $\hat{x}_{dir}$ represents the predicted `DIR`s. The terms DW and UP indicate the downstream and upstream directions, respectively. The probability of accurately predicting the `DIR` of downstream/upstream packets ($\rho_{dir}^{\text{dw}}/\rho_{dir}^{\text{up}}$) is computed by dividing the number of correct predictions in a specific direction by the total number of samples associated with that true direction. Conversely, to evaluate the prediction performance of `PL` and `IAT`, we employ the *Root Mean Squared Error (RMSE)* metric:

$$\text{RMSE}_p \triangleq \sqrt{\frac{1}{\bar{N}} \sum_{j=1}^{\bar{N}_B} \sum_{n=1}^{\bar{N}_j - 1} \left[\hat{x}_p^{n+1}(\bar{B}_j) - x_p^{n+1}(\bar{B}_j)\right]^2}. \tag{6.4}$$

with $\bar{N}$ being the total number of predictions, $x_p^{n+1}(\bar{B}_j)$ the value of the p^{th} traffic parameter (with $p \in \{\texttt{PL}, \texttt{IAT}\}$) observed for packet $n+1$ from the j^{th} biflow $\bar{B}_j$ (of length $\bar{N}_j$), and $\hat{x}_p^{n+1}(\bar{B}_j)$ the corresponding value

provided by the prediction model.

To provide a meaningful reference point for comparison, we compare the performance of DL models against that of a packet-level baseline predictor. Specifically, the latter makes predictions by assuming that the next observation value is equal to the current observation value: $\hat{x}^{n+1}_{\texttt{RLP}} \triangleq x^n$. We refer to such a baseline as `Repeat-Last-Packet` (`RLP`). Similarly, for the prediction of traffic aggregates (i.e., number of packets and traffic volume), we consider `Repeat-Last-Aggregate` (`RLA`) as a reference baseline. The `RLA` predictor forecasts traffic aggregates within a time interval Δ as equal to the aggregates of traffic observed during the previous time interval.

6.3.4 Implementation Details

For implementing and testing the DL architectures described in Sec. 6.1.2 we exploit the model provided by `Keras` (https://keras.io) Python API running on top of `TensorFlow 2` (https://www.tensorflow.org/). Input data are formatted in Parquet and optimally managed via `Apache PyArrow` (https://arrow.apache.org/). Data pre- and post-processing have been performed mainly by means of `numpy` (https://numpy.org/) and `pandas` (https://pandas.pydata.org/) libraries. For the evaluation metrics reported in Sec. 6.3, we use the implementation of `scikit-learn` (https://scikit-learn.org/). Finally, the graphical data representation has been obtained using `matplotlib` (https://matplotlib.org/) and `seaborn` (https://seaborn.pydata.org/) libraries. To ensure a fair comparison among the various architectures, we set uniform values to all adjustable hyperparameters related to the training process of the models. We train all the models for a total of 100 epochs using the Adam optimizer, with a learning rate of 0.001 and a batch size of 32. To avoid overfitting, we use a validation-based early-stopping technique, where we set the patience and min_delta parameters to 4 and 0.0001, respectively. *To foster the replicability and reproducibility of our analysis, we have publicly released the code of the DL architectures leveraged herein, along with pre-processed data and example usages.*[3]

[3] https://github.com/IdioGuarino/AFTER

6.4 Experimental Evaluation

In the following, we first investigate traffic prediction performance for all selected apps and activities, attained by different multitask DL models, to understand whether it is better to train a single model for all apps or a specific one for each of them (Sec. 6.4.1). Then, we deepen the impact of transport-layer protocols on packet-level prediction performance (Sec. 6.4.2).

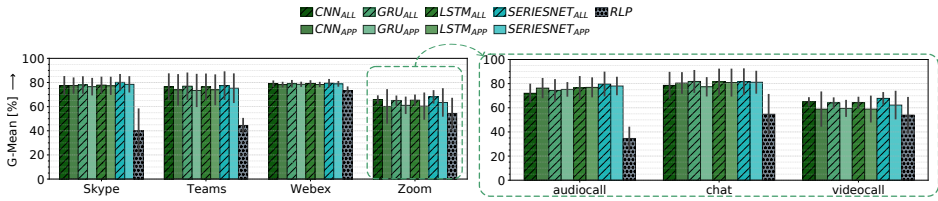
Next, we delve into the results, investigating: (*i*) how well traffic parameters can be estimated during the initial biflow lifetime via per-packet-index performance (Sec. 6.4.3); (*ii*) how much correct/wrong `DIR` estimates affect `PL` and `IAT` predictions via the conditional distributions of the prediction errors (Sec. 6.4.4).

Finally, we analyze the suitability of the packet-level traffic prediction approach for forecasting aggregate traffic characteristics (i.e., number of packets and traffic volume) in a time interval of fixed but arbitrary duration (Sec. 6.4.5).

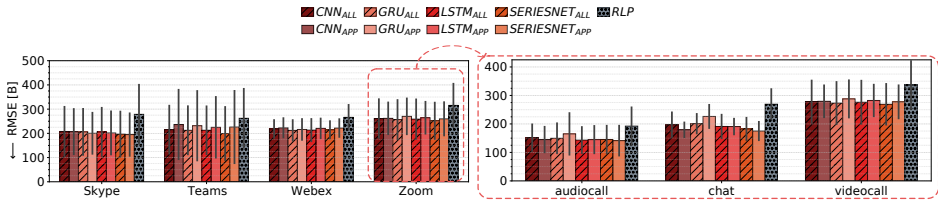
6.4.1 Do we need a dedicated model for each app?

Preliminary results—not reported for brevity—showed that $W = 10$ constitutes the best trade-off between the complexity of the model (growing with W) and the effectiveness of the prediction. Therefore, we provide an overview of the performance of the multitask DL models used for predicting `DIR`, `PL`, and `IAT` by adopting a memory window size $W = 10$. Specifically, we evaluate the effect of having one single overall model for all apps instead of one separate model for each of them. To this end, Fig. 6.4 summarizes the performance of all DL models trained according to different strategies (i.e. `ALL` vs. `APP`) with respect to all the apps considered. In addition, for each app, we also provide the performance achieved by the corresponding `RLP` *baseline*.

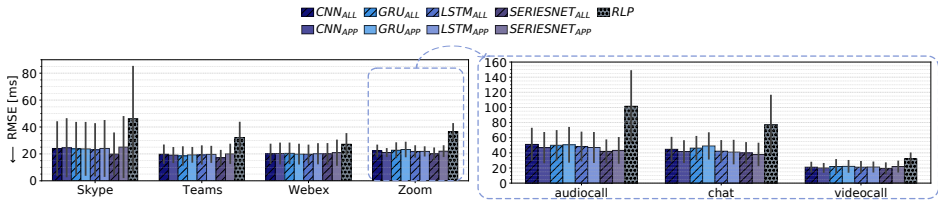
In detail, DL models always outperform `RLP`, especially on `Skype`: we observe the largest gap on all traffic parameters (i.e., $\approx -40\%$ G-mean on `DIR`, and $\approx +84$ B and $\approx +26$ ms RMSE on `PL` and `IAT`, respectively). Conversely, we observe that for the same parameters, the smallest gap is obtained on `Webex` (i.e., $\approx -6\%$ G-mean, and $\approx +54$ B and $\approx +7$ ms RMSE on `PL` and `IAT`, respectively).



(a) `DIR`.



(b) `PL`.



(c) `IAT`.

Figure 6.4. Prediction performance of CNN, LSTM, GRU, SeriesNet, and `RLP` on `DIR` (a), `PL` (b), and `IAT` (c). Results on the *left* refer to the prediction of traffic generated by *each app regardless of the specific activity*. Results on the *right* refer to *`Zoom` when performing a specific activity* (b, d, f)—i.e. `Chat`, `ACall`, and `VCall`. Memory size is set to $W = 10$ and both `APP` and `ALL` training strategies are considered. The arrow close to y-axis shows the desired trend. Results are in the form *avg.* $\pm$ *std.* obtained over 5-folds.

On the other hand, looking at the training strategies, we note that the overall model (`ALL`) outperforms or equals per-app models (`APP`). Additionally, by considering the prediction of a specific traffic parameter, we observe that both the examined strategies achieve $\approx 80\%$ G-mean when predicting `DIR` (Fig. 6.4a). Performance on `Zoom` represents an exception, showing a G-mean of $\approx 65\%$. A similar outcome is also observed for `PL` prediction (Fig. 6.4b) where `Zoom` exhibits a higher RMSE of $\approx +50$ B

compared to the other apps. A slightly different—although more stable—performance picture is evident for IAT prediction (Fig. 6.4c), for which Skype exhibits the worst RMSE (i.e., ≈+5 ms than the other apps) and also presents the highest variability. Further investigations (not reported for brevity) have shown that this phenomenon can be attributed to the higher variability of IATs corresponding to ACall and VCall activities.

Since the error on DIR has the highest impact on the use of fine-grained (viz. packet-level) prediction results, we analyze the performance of the most "problematic" app when predicting the DIR parameter in more depth. Hence, we dissect the predictions related to Zoom traffic according to the specific activity performed by the user (i.e., ACall, Chat, and VCall) in the *right column* of Fig. 6.4.

As depicted in Figs. 6.4a–6.4b, the worst DIR and PL prediction performance is obtained on VCall, where we observe the lowest G-mean (≈62%) and the greatest RMSE (≈280 B), respectively. An opposite trend can be observed when moving to the IAT prediction (Fig. 6.4c), which exhibits an RMSE on VCall less than half of that on Chat and 2.5× lower than that on ACall. Also in this case, all DL models always outperform RLP, especially for the prediction of DIR and IAT on ACall (i.e. ≈ −46% G-mean and ≈+60 ms RMSE, respectively) and of PL on Chat (i.e. ≈+94 B RMSE).

Finally, in examining the influence of the particular DL architecture utilized, it becomes evident that it has a minimal impact on the predictive performance, regardless of whether we consider the entire traffic generated by a given application or analyze the traffic based on different activities[4]. Despite the comparable performance, the complexity of the considered architectures varies significantly. We quantify it with the number of trainable parameters: more trainable parameters result in a longer training time. CNN is the least complex architecture with 27.5 K parameters, while GRU, LSTM, and SeriesNet are significantly more complex and have +96.1 K (GRU), +135.3 K (LSTM), and +502.3 K (SeriesNet) more trainable parameters.

[4]Similar considerations can also be drawn for Skype, Teams, and Webex whose per-activity performance figures are not shown for the sake of brevity.

Takeaways

RQ: *It is better to use a single shared model or a dedicated model for each CC app to predict their fine-grained traffic?*

Training a single model on the entire traffic of CC apps is sufficient, as there is no significant benefit in training specialized models for each individual app. This finding has significant practical implications as it eliminates the need to train, deploy, and manage multiple models.

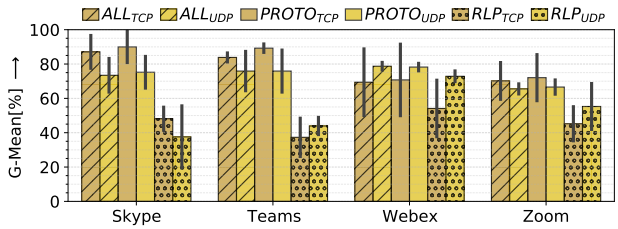
`Zoom` is the hardest app to predict, especially in terms of `PL` and `DIR`, while other apps show better and more consistent performance. Among the `Zoom`-related activities, `VCall` is the most difficult to predict in terms of `PL` and `DIR`.

Despite the similar performance of the considered DL architectures, they exhibit considerable differences in terms of computational complexity, with CNN being the least complex.

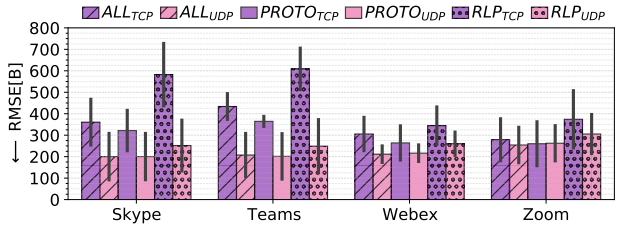
6.4.2 Do we need a dedicated model for TCP and UDP protocols?

We previously showed in Sec. 6.4.1 that training a CNN architecture on all apps (`ALL` strategy) achieves the best trade-off between performance and computational complexity. Additionally, Sec. 6.3 revealed that CC apps utilize both TCP and UDP protocols in their operations. Indeed, while they tend to establish numerous TCP connections (viz. biflows), most data (in terms of both packets and volume) are transmitted through UDP biflows. This is generally because TCP is mainly used for various control and data management purposes. At the same time, UDP handles real-time communications related to audio or video traffic in the case of CC apps, which require low latency and fast transmission (cf. Fig. 4.1).

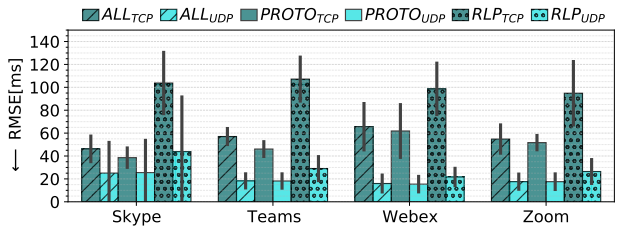
Hence, here we investigate the benefits of having a separate model for each transport protocol (ref. `PROTO` strategy) against employing a single model to handle both TCP and UDP traffic (ref. `ALL` strategy). In Fig. 6.5, we present the performance results of these models for each



(a) DIR.



(b) PL.



(c) IAT.

Figure 6.5. Prediction performance of CNN and RLP on DIR, PL, and IAT w.r.t. UDP and TCP protocols. For CNN, the results obtained by utilizing the ALL training strategy are compared with those achieved by using the PROTO strategy. Results refer to the prediction of traffic generated by all apps with a memory $W = 10$. Results are in the form $avg. \pm std.$ obtained over 5-folds.

app, distinguishing between specific protocols and training strategies. For the sake of completeness, we also include the breakdown of performance achieved by RLP predictor.

Overall, our results show that the effectiveness of the two strategies depends on the app and the protocol: while for UDP traffic, the performance is unaffected by the training strategy, for TCP traffic, training a specific

model for the protocol results in a considerable performance improvement for all prediction tasks. In particular, the improvement on DIR prediction is up to +5% of G-mean, and up to ≈−70 B and ≈−11 ms of RMSE for PL and IAT, respectively.

By breaking results down into specific apps and protocols, we observe that prediction of DIR performs better on TCP traffic for nearly all apps except Webex. However, this trend reverses when predicting PL and IAT, where models for TCP traffic exhibit poorer performance.

Furthermore, our results reveal that CNNs consistently outperform RLP predictor for all predicted parameters, irrespective of the protocol and training strategy. The difference is particularly pronounced for TCP traffic generated by Skype and Teams, with CNNs outperforming the RLP predictor by approximately ≈+40% of G-mean for DIR and ≈−240 B and ≈−60 ms of RMSE for PL and IAT, respectively. Similar trends are also observed for Webex and Zoom, where the difference between CNNs and the RLP predictor is up to +27% for DIR and up to −120 B and −40 ms on PL and IAT, respectively. When analyzing UDP traffic, the performance gap between CNNs and the RLP predictor is still evident but less prominent. For Skype, CNN models outperform the RLP predictor by ≈+40% for DIR, ≈−50 B for PL, and ≈−20 ms for IAT.

Takeaways

RQ: *Is it better to use a dedicated model for each transport layer protocol or a single shared model to predict fine-grained CC app traffic?*

The effectiveness of a specific training strategy (i.e., `ALL` and `PROTO`) varies depending on the transport-level protocol. For all apps, training a dedicated model for TCP traffic (i.e., the `PROTO` strategy) generally leads to notable improvements in performance across all prediction tasks. However, the same approach does not show similar benefits for UDP traffic, as the training strategy does not have a noticeable impact.

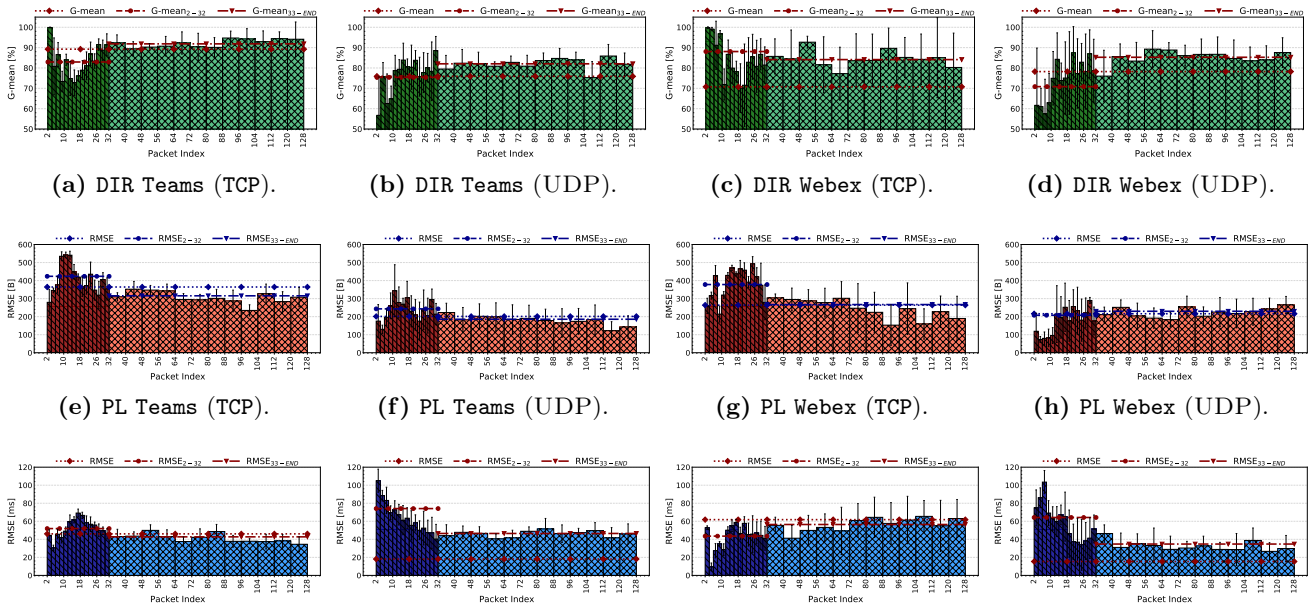
Hereinafter, we focus on the CNN model, individually trained for each transport layer protocol (i.e., the `PROTO` variant). This approach yields the least amount of error, with the trade-off being the necessity for just two distinct models (one for TCP and one for UDP).

6.4.3 How does prediction performance vary along a biflow?

Following the results shown in Sec. 6.4.2, we aim to evaluate whether (and how) the prediction performance of CNNs trained using the `PROTO` strategy varies along a biflow. Hence, we provide a *per-packet-index performance* analysis, where the evaluated metrics (i.e. G-mean for `DIR` and RMSE for `PL` and `IAT`) are computed considering packets at the same position in biflows, namely sharing the same index or falling within the same interval of indexes. Specifically, we focus on the head of the biflows (i.e., the first 128 packets), showing aggregated performance each 2 packets until the 32^{nd} and each 8 packets for the remaining segment.

Hence, Figs. 6.6 reports the results for `Teams` and `Webex` (`Skype` and `Zoom` show similar patterns, thus they are omitted for brevity) with respect to both `TCP` and `UDP` traffic.

In detail, for `Teams`, the analysis witnesses that predicting all traffic parameters is more challenging for packets ranging from 2 to 32 for both `TCP` and `UDP` traffic. In fact, the G-mean resulting from the prediction



(a) `DIR` `Teams` (`TCP`). **(b)** `DIR` `Teams` (UDP). **(c)** `DIR` `Webex` (`TCP`). **(d)** `DIR` `Webex` (UDP).

(e) `PL` `Teams` (`TCP`). **(f)** `PL` `Teams` (UDP). **(g)** `PL` `Webex` (`TCP`). **(h)** `PL` `Webex` (UDP).

(i) `IAT` `Teams` (`TCP`). **(j)** `IAT` `Teams` (UDP). **(k)** `IAT` `Webex` (`TCP`). **(l)** `IAT` `Webex` (UDP).

Figure 6.6. Per-packet index performance of CNN_{proto} trained on `TCP` (*odd columns*) and UDP (*even columns*), in terms of G-mean for `DIR` and RMSE for `PL` and `IAT` of the first 128 packets. Results refer to `Teams` (*columns* $1-2$) and `Webex` (*columns* $3-4$) and are in the format $avg \pm std$ obtained over the 5-folds. Horizontal lines report the overall G-mean/RMSE, the G-mean/RMSE of the first 32 packets (G-mean/RMSE$_{2-32}$), and the G-mean/RMSE of the remaining ones (G-mean/RMSE$_{33-END}$) of each biflow.

of the DIR of the packets from 2 to 32 (i.e., G-mean$_{2-32}$) is lower than that observed on the remaining part of the biflow (i.e., G-mean$_{33-\text{END}}$), on average: −7% and −6% on TCP and UDP, respectively. Similarly, the prediction error incurred for PL and IAT on the initial part of the biflows (i.e., RMSE$_{2-32}$) is higher than the error on the remaining part (i.e., RMSE$_{33-\text{END}}$), on average: up to +120B (resp. +40B) and +8ms (resp. +28ms) on PL and IAT for TCP (resp. UDP), respectively. Conversely, for Webex, packets from 2 to 32 are easier (resp. harder) to be predicted in terms of DIR and IAT on TCP (resp. UDP) traffic. This results in an average gain (resp. loss) of up +4% (resp. −16%) of G-mean and −12ms (resp. +28ms) of RMSE, respectively. Finally, when looking at PL, the gap between the first 32 packets and the remaining ones results in an increased (resp. reduced) error, on average, of about +80B (resp. −20B) of RMSE.

Takeaways

RQ: *How does prediction performance vary along TCP and UDP biflows of CC apps?*

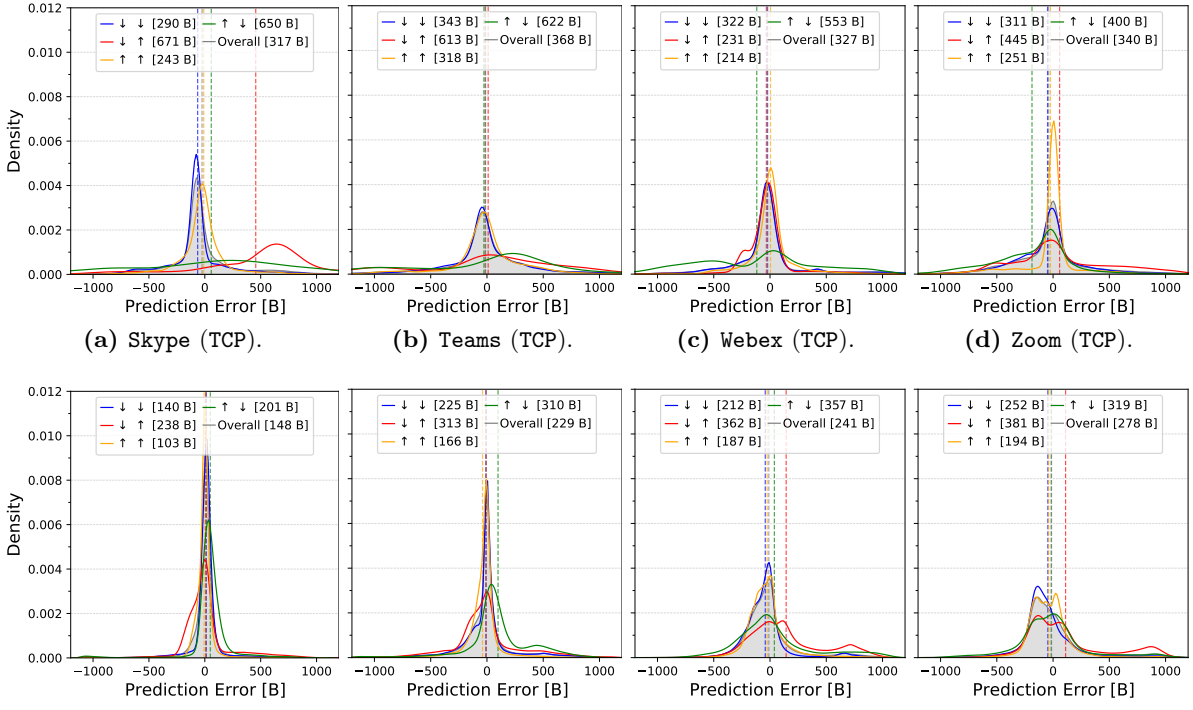
A consistent discrepancy is found between the lower prediction capabilities achieved for the beginning part and the higher capabilities attained for the rest of the biflows.

For some apps (e.g., Teams), the prediction of traffic parameters is more challenging for the first packets of a biflow, regardless of the type of traffic (i.e., TCP or UDP). In contrast, for other apps (e.g., Webex), while DIR and IAT prediction of the first packets is easier for TCP, it is more difficult for UDP traffic.

Overall, the position of packets within biflows significantly affects prediction performance.

6.4.4 What kind of errors do the models make?

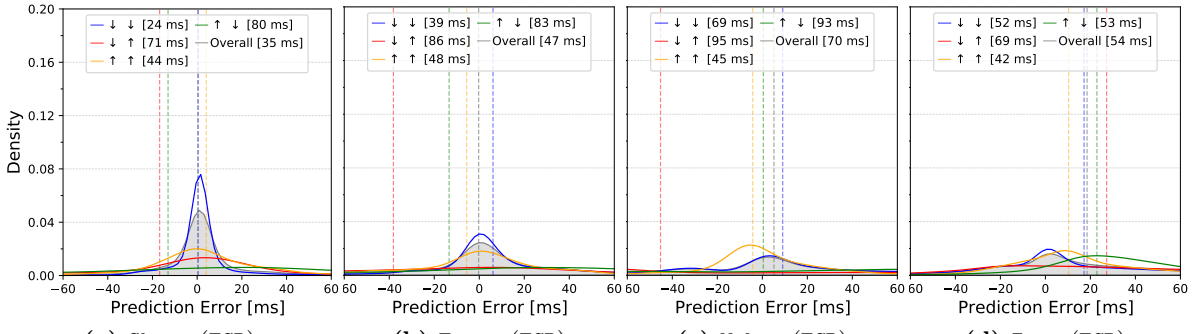
Herein, we aim at characterizing the errors the prediction models make. Figs 6.7 and 6.8 report the prediction error ($\hat{x}_{(\cdot)}^{n+1} - x_{(\cdot)}^{n+1}$) associated with PL



(a) Skype (TCP). **(b)** Teams (TCP). **(c)** Webex (TCP). **(d)** Zoom (TCP).

(e) Skype (UDP). **(f)** Teams (UDP). **(g)** Webex (UDP). **(h)** Zoom (UDP).

Figure 6.7. Probability density of prediction error on PL related to CNN_{proto} trained on TCP (*first row*) and UDP (*second row*) traffic. The probabilities are conditioned to different combinations of (predicted, true) direction predictions, and overall (e.g., ↑ ↑ represents correct prediction of upstream direction, ↓ ↑ an erroneous prediction of downstream for an actually upstream packet). The number in brackets is the RMSE for the specific combination and overall.



(a) Skype (TCP). **(b)** Teams (TCP). **(c)** Webex (TCP). **(d)** Zoom (TCP).

(e) Skype (UDP). **(f)** Teams (UDP). **(g)** Webex (UDP). **(h)** Zoom (UDP).

Figure 6.8. Probability density of prediction error on IAT related to CNN_{proto} trained on TCP (*first row*) and UDP (*second row*) traffic. The probabilities are conditioned to different combinations of (predicted, true) direction predictions, and overall (e.g., ↑ ↑ represents correct prediction of upstream direction, ↓ ↑ an erroneous prediction of downstream for an actually upstream packet). The number in brackets is the RMSE for the specific combination and overall.

and IAT, respectively, for all the apps (grey curves). Specifically, we report the error related to TCP and UDP traffic for each app when using CNNs trained leveraging the PROTO strategy. By construction, negative values in the distribution are related to under-estimation (i.e., the prediction is *lower* than the actual value) for either PL or IAT, while positive ones report over-estimation (i.e., the prediction is *higher* than the actual value).

Overall, the errors span almost the whole theoretical $(-1470, 1470)$ B range for PL, while they lie in the range $(-198, 196)$ ms for IAT. Errors with small magnitudes are much more frequent than those with high magnitudes, particularly for UDP traffic. In almost all the cases, the bias of the distribution is placed around 0 at a distance always ≤ 40 B (resp. ≤ 9 B) and ≤ 2 ms (resp. ≤ 19 ms) for PL and IAT, respectively, on TCP (resp. UDP). It is also worth noting that the RMSE on TCP is always higher than the one on UDP for all apps and traffic parameters (i.e., PL and IAT). For PL prediction, the RMSE varies between 317 B (resp. 148 B) and 368 B (resp. 278 B) for Skype and Zoom (resp. Teams), respectively, on TCP (resp. UDP) traffic. In contrast, for IAT prediction, the RMSE varies between 35 ms (resp. 12 ms) and 70 ms (resp. 17 ms) for Skype and Teams (resp. Webex), respectively, on TCP (resp. UDP) traffic.

To analyze the relationship between the predictions on DIR and the other two metrics (PL and IAT), Figs. 6.7 and 6.8 report also the breakdown of the error distribution conditioned on true and predicted DIR—i.e., $(\hat{x}_{(\cdot)}^{n+1} - x_{(\cdot)}^{n+1})|\hat{x}_{\text{dir}}, x_{\text{dir}}$.

Observing the impact of correct/wrong predictions of DIR on the error incurred for the other two metrics, it is evident that when the models make mistakes in predicting DIR, higher errors are recorded for both PL and IAT regardless of the specific app. Moreover, the above finding is exacerbated mainly on TCP traffic.

Looking closer at PL on TCP traffic (see Figs. 6.7a-6.7d), we notice that for Teams, incorrect predictions about DIR do not significantly affect PL. However, this does not hold for the other apps, where it generally leads to an under- or over-estimation of PL, on average, depending on the specific app and the actual DIR. More in detail, the observed bias equals to 456 B and -118 B (resp. -187 B) for Skype and Webex (resp. Zoom) when $(\hat{x}_{\text{dir}}, x_{\text{dir}}) = (\downarrow, \uparrow)$ and $(\hat{x}_{\text{dir}}, x_{\text{dir}}) = (\uparrow, \downarrow)$, respectively. Notably, for Skype and Teams, the probability of committing an error in magnitude

greater than 200 B occurs in 87% (resp. 78%) and 64% (resp. 75%) of the cases when $(\hat{x}_{\text{dir}}, x_{\text{dir}}) = (\downarrow, \uparrow)$ (resp. $(\hat{x}_{\text{dir}}, x_{\text{dir}}) = (\uparrow, \downarrow)$). Overall, it is observed that PL over-estimation is more probable than the under-estimation for all apps, regardless of the type of error on DIR prediction.

Regarding IAT (ref. Figs. 6.8a- 6.8d), incorrect DIR predictions significantly affect IAT for almost all apps, especially when $(\hat{x}_{\text{dir}}, x_{\text{dir}}) = (\downarrow, \uparrow)$. In this case, the absolute value of the bias has a minimum and a maximum of -17 ms (Skype) and -45 ms (Zoom), respectively. Interestingly, for Zoom the absolute value of the bias is at least 10 ms $(\hat{x}_{\text{dir}}, x_{\text{dir}}) = (\uparrow, \uparrow)$ regardless of whether the direction is predicted well or incorrectly, corresponding to a minimum RMSE on of 42 ms. Finally, we observe that in most cases, there is an over-estimation (resp. under-estimation) of IAT when the actual DIR is downstream—i.e., $(\hat{x}_{\text{dir}}, x_{\text{dir}}) = (\cdot, \downarrow)$. In contrast, there is an over-estimation (resp. under-estimation) when the model correctly (resp. incorrectly) predicts the upstream direction, i.e., $(\hat{x}_{\text{dir}}, x_{\text{dir}}) = (\uparrow, \uparrow)$ (resp. $(\hat{x}_{\text{dir}}, x_{\text{dir}}) = (\downarrow, \uparrow)$).

Moving on UDP, we notice that, while for Skype wrongly predicting DIR has no remarkable impact on PL, the same does not apply to the other apps, where over-estimation of PL is observed, on average (ref. Figs. 6.7e- 6.7h). In fact, the observed bias equals to 99 B (resp. 111 B) for Teams (resp. Zoom) when $(\hat{x}_{\text{dir}}, x_{\text{dir}}) = (\uparrow, \downarrow)$ (resp. $(\hat{x}_{\text{dir}}, x_{\text{dir}}) = (\downarrow, \uparrow)$). For Webex the bias is 144 B and 39 B when $(\hat{x}_{\text{dir}}, x_{\text{dir}}) = (\downarrow, \uparrow)$ and $(\hat{x}_{\text{dir}}, x_{\text{dir}}) = (\uparrow, \downarrow)$, respectively. Conversely, concerning IAT (ref. Figs. 6.8e- 6.8h) remarkable under-estimation of the parameter is observed when $(\hat{x}_{\text{dir}}, x_{\text{dir}}) = (\downarrow, \uparrow)$ for Skype and Teams. The corresponding bias is -3 ms and -7 ms, respectively.

Overall, PL under-estimation is more frequent than over-estimation for Zoom, Webex, and Teams, while the opposite holds for Skype. This tendency is also confirmed when restricting the observation to cases where DIR is not mistaken. On the contrary, the probability of over-estimating IAT is higher than under-estimating it for all the apps. Errors on PL (both over- and under-estimation) larger than 200 B appear in less 25% of the cases for all the apps (less than 10% of the cases for Skype). Similarly, errors on IAT with magnitude larger than 10 ms appear for less than 30% of the cases.

Takeaways

RQ: *What kinds of errors do models make in predicting TCP and UDP traffic of CC apps?*

Errors related to incorrect `DIR` *prediction lead to higher errors for both* `PL` *and* `IAT`*, particularly for* `TCP` *traffic.*

On `TCP` traffic, the impact of incorrect `DIR` predictions on `PL` varies between apps, leading to over- or under-estimation. For `IAT`, incorrect `DIR` predictions significantly affect most apps, particularly when the predicted direction is downstream.

On `UDP` traffic, the impact of incorrect `DIR` predictions is observed primarily in `PL`, leading to over-estimation for several apps. However, in the case of `IAT`, for some apps, a substantial under-estimation occurs when the actual `DIR` is upstream, and the model wrongly predicts the downstream direction.

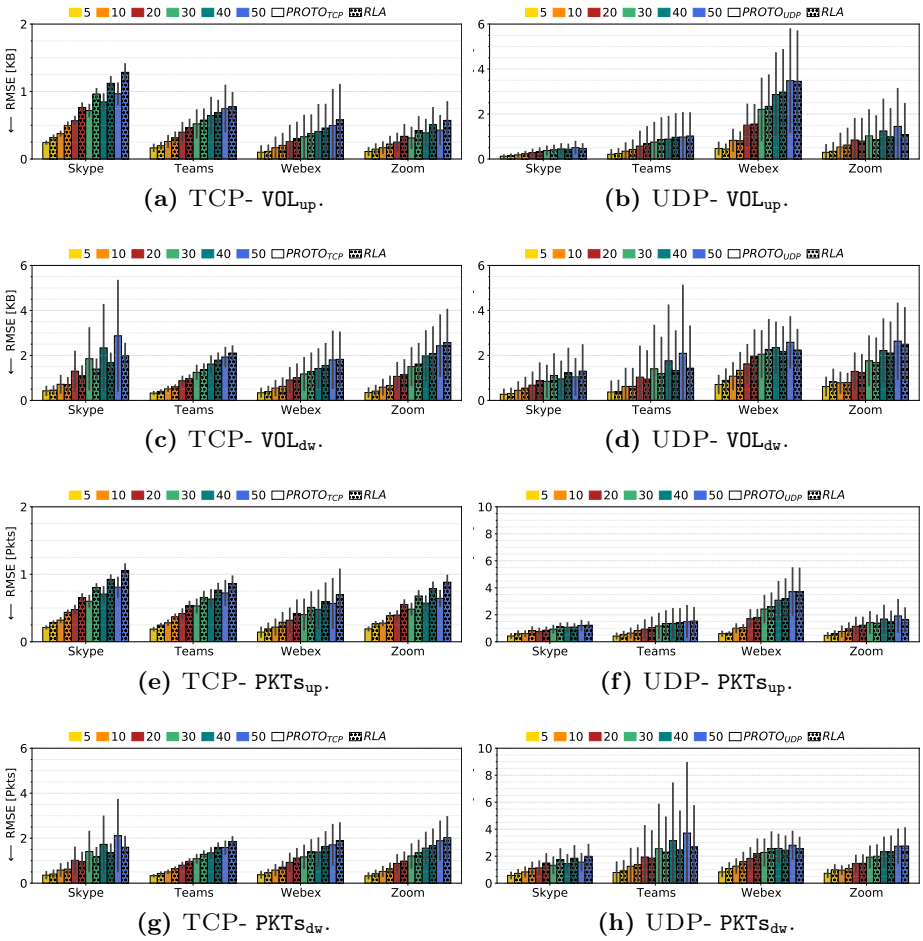
Overall, while the probability of under-estimating `PL` is higher than over-estimating it for almost all the apps, the opposite trend holds for `IAT`.

6.4.5 Is Packet-Level Prediction suitable to Predict Traffic Aggregates?

Herein, we analyze the suitability of packet-level predictors to forecast aggregate traffic (i.e., the number of packets and the traffic volume) in a given time interval Δ. Therefore, applying the methodology outlined in Sec. 6.2, we compare the performance of a CNN trained leveraging the `PROTO` strategy (i.e., two dedicated models designed for TCP and UDP traffic) against a simple `RLA` predictor.

Fig 6.9 depicts the performance—in terms of RMSE—for each protocol when predicting the number of packets (viz. $\texttt{PKTs}_*$) and the volume (viz. $\texttt{VOL}_*$), for both upstream and downstream directions. The performance is reported for different time horizons, namely $\Delta \in \{5, 10, 20, 30, 40, 50\}$ ms.

At a high level, we observe that the error increases with Δ. The greater



(a) TCP- $\texttt{VOL}_{\texttt{up}}$. **(b)** UDP- $\texttt{VOL}_{\texttt{up}}$.

(c) TCP- $\texttt{VOL}_{\texttt{dw}}$. **(d)** UDP- $\texttt{VOL}_{\texttt{dw}}$.

(e) TCP- $\texttt{PKTs}_{\texttt{up}}$. **(f)** UDP- $\texttt{PKTs}_{\texttt{up}}$.

(g) TCP- $\texttt{PKTs}_{\texttt{dw}}$. **(h)** UDP- $\texttt{PKTs}_{\texttt{dw}}$.

Figure 6.9. Prediction performance (RMSE) of CNN trained on `TCP` (*left column*) and UDP (*right column*) traffic for $\texttt{VOL}_{\texttt{up}}$ (a, b), $\texttt{VOL}_{\texttt{dw}}$ (c, d), $\texttt{PKTs}_{\texttt{up}}$ (e, f), and $\texttt{PKTs}_{\texttt{dw}}$ (g, h) as $\Delta \in \{5, 10, 20, 30, 40, 50\}$ *ms*. CNN performance is compared against `RLA` predictor. Results refer to the prediction of traffic generated by all apps with a memory $W = 10$.

unpredictability of traffic over extended time intervals could be attributed to its greater variance and uncertainty [101] and also to the growth of prediction errors accumulated through the recursive procedure adopted. This variability is particularly evident in VoIP traffic, encompassing both audio and video, as it tends to produce *micro-bursts* that are difficult to predict——i.e., a notable surge in data transmission within a brief duration, followed by relatively quieter periods.

We also note that this error varies depending on three factors: (i) traffic direction, (ii) protocol, and (iii) app considered. Taking a closer look, we observe that the errors in the upstream direction increase gradually and tend to stay within a relatively narrow range (averaging around $\approx$1.3 KB and $\approx$ 1 packet) for most apps and protocols, except for `Webex` on UDP traffic. In contrast, the downstream direction shows a more abrupt and highly variable change, with errors reaching higher average levels (up to $\approx$3 KB and $\approx$4 packets).

When analyzing the performance on TCP traffic (see *left* column of Fig. 6.9), CNN proves to be a superior choice for predicting all features compared to `RLA`, regardless of the prediction interval Δ. This advantage is particularly evident when considering the traffic direction for `Skype`, `Zoom`, and `Teams`. Specifically, for `Skype` (resp. `Zoom`), the mean RMSE of CNN is at least $\approx$23% (resp. $\approx$24%) and $\approx$23% (resp. $\approx$27%) lower for $\texttt{VOL}_{\texttt{up}}$ and $\texttt{PKTs}_{\texttt{up}}$, respectively. Similarly, for `Teams`, the difference is at least 8% and 27% lower for $\texttt{VOL}_{\texttt{dw}}$ and $\texttt{PKTs}_{\texttt{dw}}$, respectively. Interestingly, the only scenario where this trend is reversed is observed on `Skype` in the downstream direction when $\Delta \geq 30$ ms (see Figs. 6.9c and 6.9g). In such cases, the RMSE of the CNN on $\texttt{VOL}_{\texttt{dw}}$ and $\texttt{PKTs}_{\texttt{dw}}$ is up to +45% and +32% higher than that of `RLA`.

However, when moving on UDP traffic (see *right* column of Fig. 6.9), no consistently predominant approach can be identified. Nonetheless, the results indicate that CNN is the most favorable option for most scenarios, particularly when predicting the number of packets. Specifically, compared to `RLA`, we find that CNN provides a more accurate packet number estimation for applications such as `Skype`, `Teams`, and `Webex` in the upstream direction (Fig. 6.9f), as well as for `Skype` and `Zoom` in the opposite direction (Fig. 6.9h). However, it is important to note that for certain apps (e.g., `Teams` and `Zoom` in the downstream and upstream directions,

respectively), while CNN outperforms the `RLA` predictor on short prediction intervals (i.e., $\Delta \leq 20$ ms), the trend reverses on longer ones.

Lastly, when examining the results on traffic volumes (see Figs. 6.9b and 6.9d), we notice a more diverse pattern between CNN and `RLA` methods. In the case of `Teams` (`Skype`), the CNN consistently surpasses the `RLA` approach, exhibiting a reduced error ranging from -4% (-10%) to -18% (-24%) on the upstream (downstream) direction. In the other scenarios, the CNN remains the superior option for predicting aggregate traffic within relatively short time intervals (i.e., $\Delta \leq 30$ ms). However, CNN is less effective over longer time horizons. Nonetheless, it is worth noting that having a more accurate predictor over short time intervals facilitates constant monitoring of network performance and real-time optimization. These aspects are essential when dealing with CC apps, especially when the user uses these to perform VoIP calls (i.e., audio and video), which require a good level of QoS and, at the same time, tend to cause intense activity on the network.

Takeaways

RQ: *Is it possible to predict traffic aggregates by exploiting a fine-grained prediction approach?*

The proposed approach outperforms `RLA` in most scenarios, especially when used to predict aggregate traffic over relatively short time intervals (e.g., $\Delta \leq 30$ ms*).* Within this range, the proposed approach yields an average error of up to ≈ 1.3 KB (≈ 1.6 KB) and 1 (2) packet(s) on TCP (UDP) traffic protocol. Overall, the error is limited to $\approx$3 KB ($\approx$3.5 KB) and 2 (4) packets on TCP (UDP).

These findings are significant as they indicate that the proposed approach represents a first step toward developing more responsive networks. Accurate short-term predictions facilitate real-time optimization of network capacity, enabling effective bandwidth adaptation and resource allocation as needed. This can potentially enhance the network's performance and responsiveness in dynamic and ever-changing traffic conditions.

Chapter 7

Empowering Trustworthiness on Deep Learning Solutions

The black-box nature of DL models makes it difficult for network operators to trust them, especially when the results of their decisions could significantly impact the network. In this scenario, XAI can assist network operators in making improved decisions regarding integrating AI into their networks by offering transparency, building trust, identifying problems, and ensuring compliance with regulations [63].

Then, in this chapter, we inspect the performance of models specifically designed to classify and predict the traffic generated by CC apps, using XAI techniques.

In detail, the chapter focuses on (a) the interpretation of these models using post-hoc techniques (Sec. 7.1) and (b) analyzing their reliability through calibration analysis (Sec. 7.2). Each section first describes the XAI technique used and then applies it to study the interpretability/reliability of DL models.

7.1 Interpretability via Deep SHAP

Hereinafter, we present the methodology employed to investigate the *interpretability* of the results obtained from a generic probabilistic classification task using DL architectures. We recall that such an interpretability analysis provides a means to *explain* the model, determining whether the

predictions are more influenced by specific parts of the input traffic and uncovering potential biases. It can also enable the improvement of model performance and the assessment of robustness and vulnerabilities (e.g., how much the model is susceptible to adversarial attacks). Additionally, from a *transparency* viewpoint, interpretability techniques can facilitate the validation of the prediction outcome by providing insights into the internal mechanisms of DL architectures, thus making the resulting decisions more trustworthy.

We highlight that the same methodology can be applied to the `DIR` prediction task, which can be seen as a binary classification task.

In detail, to start interpreting DL architectures, we adopt a simpler explanation model, denoted as $g(\cdot)$, designed to closely approximate the original model $f(\cdot)$. To explain the predictive behavior of a DL-based traffic classifier, we use the model $f(\cdot)$ as the soft output associated with the generic i^{th} class, i.e., $p_i(\cdot)$. This allows us to determine which inputs contribute the most to the confidence probability value that is associated with a given class. In the following, our focus lies on *local methods*, which explain the model $f(\boldsymbol{x})$ within the neighborhood of a specific instance $\boldsymbol{x}$. It is worth noting that, in the case of TC, this type of explanation is termed a per-flow explanation, while in the case of TP, it is referred to as an explanation of the input packet sequence. These local explanations utilize *simplified inputs* $\boldsymbol{x'}$, which are mapped to the original inputs $\boldsymbol{x}$ through the mapping $\boldsymbol{x} = \boldsymbol{h_x}(\boldsymbol{x'})$.

Herein, we adopt the *Additive Feature Attribution (AFA)* functional form as the explanation model $g(\cdot)$ for our analysis. The AFA model is defined as:

$$g(\boldsymbol{z}^{'}) = \phi_0 + \sum_{m=1}^{M} \phi_m \, z_m^{'}. \tag{7.1}$$

where $\boldsymbol{z}^{'} \in \{0,1\}^M$, M represents the number of simplified inputs, and $\phi_m \in \mathbb{R}$. This specific class of explanation models assigns an "effect" ϕ_m to each input, indicating its contribution. Consequently, the output of the original model $f(\boldsymbol{x})$ can be approximated by summing the effects of all input attributions.

A common method for computing AFA solutions is using *Shapley values*, originating from cooperative game theory. These values quantify the

contribution of a particular player, denoted as m, to the overall payoff achieved by the entire coalition $\mathscr{C}$, denoted as $v(\mathscr{C})$. Specifically, the overall payoff is obtained by first evaluating the payoff of all possible subsets $\mathscr{S} \subset \mathscr{C}$ of cooperating players, including m. Secondly, the effect of excluding player m from each $\mathscr{S}$ on the payoff is evaluated, namely $v(\mathscr{S}) - v(\mathscr{S} \setminus m)$. The final m^{th} Shapley value is then obtained by taking the (weighted) average of such differences over all the $\mathscr{S}$. When applying this method to explain a DL-based model, the input data is mapped to the players of the cooperative game. At the same time, the output of the DL architecture, represented by $f(\boldsymbol{x})$, corresponds to the payoff function. However, the exact computation of Shapley values becomes exponentially complex as the input size, denoted as M, increases. To address this challenge, we employ an approximation method known as *SHapley Additive exPlanation (SHAP)*, which efficiently calculates the Shapley values. This approximation eliminates the need to retrain the models by approximating the Shapley values through the conditional expectation $f(\boldsymbol{h}_{\boldsymbol{x}}(\boldsymbol{z}^{'})) \approx \mathbb{E}\{f(\boldsymbol{z})|\boldsymbol{z}_{\mathscr{S}}\}$, where $\mathscr{S}$ represents the set of non-zero indices within $\boldsymbol{z}^{'}$.

Specifically, in our study, we utilize Deep SHAP [102], an adaptation of the *DeepLIFT* algorithm designed for evaluating SHAP values on neural architectures. DeepLIFT employs a compositional approximation of SHAP values using the output expectation as the reference value. It also utilizes explicit Shapley equations for consistent linearization. The reference value is a user-defined parameter chosen to be an uninformative background value for the m^{th} input.

To be more specific, we use Deep SHAP to explain the soft-output associated with the predicted class, denoted as $\hat{p}(\boldsymbol{x})$, to assess the importance of inputs selected from raw traffic data $\boldsymbol{x}$ used to feed the DL model. To be more specific, in TC case, the inputs pertain to `PAY`, `SEQ`, or `context` inputs, while $\hat{p}(\boldsymbol{x})$ refers to the specific app and activity. On the other hand, in TP case, inputs refer to the sequence of the most recent W input packets, while $\hat{p}(\boldsymbol{x})$ refers to the direction of the next packet (i.e., *upstream* or *downstream*). For a given input sequence $\boldsymbol{x}$, we interpret the SHAP value ϕ_m as the importance value of the m^{th} traffic parameter composing $\boldsymbol{x}$ in forming the confidence p_i associated with the i^{th} class.

It is worth noting that since $\phi_m \in \mathbb{R}$, which means that values can

be negative, we should interpret the importance values as follows: positive values increase the confidence in the i^{th} class compared to its average value, while negative values decrease it. Additionally, the sum of the SHAP values equals the soft-output value ($p_i(\boldsymbol{x})$) minus the *base output*. The *base output* represents the average of the same soft-output value obtained from the samples associated with the background set, i.e., $\mathbb{E}\{p_i\}$. Finally, explanations obtained from local methods are combined to generate *global explanations*. Due to the variability of soft outputs, the absolute importance range of the m^{th} input may vary from sample to sample.

Therefore, our global explanation approach involves the initial computation of *range-normalized* SHAP values:

$$\widetilde{\phi}_m \triangleq \phi_m \,/\, \sum_{m=1}^{M} \phi_m. \tag{7.2}$$

By using $\widetilde{\phi}_m$ instead of ϕ_m, we can focus on the relative importance of each input and derive importance measures that are independent of the specific confidence levels of the architecture. This normalization ensures consistent aggregation over the test samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ and removes dependence on the architecture's peculiar confidence levels, which can be generally higher or lower.

In the following, we utilize DEEP SHAP to provide explanations regarding both the classification and prediction of the traffic of CC apps. Firstly, we explain the behavior of the MIMETIC-ALL architecture, which is used to classify the traffic at both app and activity levels (Sec. 7.1.1). Secondly, we explain how CNNs trained with the `PROTO` strategy behave when predicting the direction of the next packet (Sec. 7.1.2).

7.1.1 TC: How do inputs affect joint App and Activity classification?

Following that shown in Sec.5.3.1, we now discuss the interpretability of the results of MIMETIC-ALL when dealing with the Joint-TC task. More in detail, relying on DEEP SHAP, we first analyze the relative contribution of each modality (viz. branch), then thoroughly investigate the inputs corresponding to each of them. To be more specific, our emphasis is on the overall correctly classified tested samples [71]. This choice enables us to

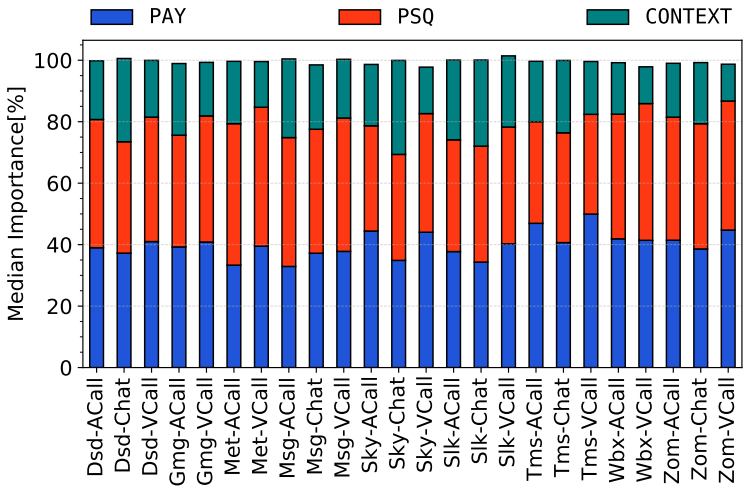


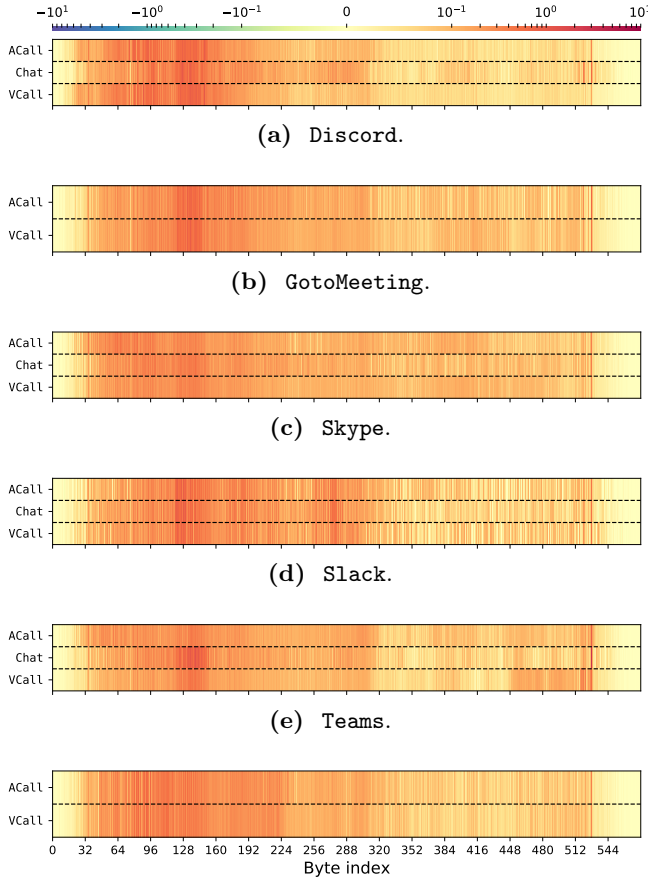
Figure 7.1. Median importance of each Input (i.e., `PAY`, `SEQ`, and `Context`) on Mimetic-All w.r.t. the Joint-TC task. Each combination of app and activity is separately analyzed according to the correctly classified samples.

concentrate on the correct functioning of a deep learning-based traffic classifier and interpret its counterintuitive (yet correct) decisions afterward.

As a result, Fig. 7.1, reports the median importance of each modality (i.e., bPAY, bSEQ, and bContext) that compose Mimetic-All w.r.t. to each (*app*, *activity*) combination.

Overall, it is evident that bPAY and bSEQ branches are more important than that of bContext. In fact, the importance of both bPAY and bSEQ ranges from 30% and 50%, while for bContext it is in the range $[12, 30]$%. Notably, for 5 out of 9 apps (i.e., `Discord`, `GotoMeeting`, `Slack`, `Webex`, and `Zoom`), both bPAY and bSEQ contribute with similar importance regardless of activity. However, for `Skype`, while bPAY and bSEQ have similar importance for `Chat`, this does not hold for `ACall` and `VCall`, where the former modality is more important. Moreover, we observe that for `Teams` (resp. `Meet` and `Messenger`), bPAY is more (resp. less) important than bSEQ regardless of activity. Finally, focusing on bContext, we observe that in almost all cases, its importance is ≥ 20%. However, when dealing with `VCall` for `Webex` and `Zoom`, its importance falls in the range $[12, 15]$%.

Hereinafter, we inspect the relative importance of the inputs corresponding to each modality.



(a) `Discord`.

(b) `GotoMeeting`.

(c) `Skype`.

(d) `Slack`.

(e) `Teams`.

(f) `Webex`.

Figure 7.2. Median importance (in Symlog-scale) of `PAY` Inputs (identified by their position) for bPAY-branch of Mimetic-All. Results refer to exemplifying apps—i.e., `Discord` (a), `GotoMeeting` (b), `Slack` (c), and `Teams` (d)—, based on the activity-type w.r.t. Joint-TC task. Each combination of app and activity is separately analyzed according to the correctly classified samples.

PAY Input Fig. 7.2 shows the importance of PAY inputs (i.e., the first $N_b = 576$B of each biflow) on bPAY branch by reporting the activity breakdown for different apps, including Discord, GotoMeeting, Skype, Slack, Teams, and Webex[1]. More in detail, for each combination (app, activity), we report the median importance value of each byte (identified by its index) that composes PAY input.

As shown, the bytes that compose the PAY input generally contribute positively to the classification task, but their importance varies by app and activity. However, the presence of different regions highlights that not all bytes have the same importance. In fact, bytes from 32^{nd} to 320^{th} generally are the most important, indicating that they are crucial to correctly classifying the app and activity. Conversely, the first 32 and the last 48 bytes correspond to very low importance, suggesting that they are of little relevance to the classification task under consideration. It is worth noting the presence of some differences between apps and activities in terms of the most important bytes. While Skype and Webex show a fairly uniform distribution of importance across bytes, other applications have some *hot spots*. These hot spots are located around bytes from the 120^{th} to 160^{th} for all apps and from the 272^{nd} to 270^{th} specifically for Slack.

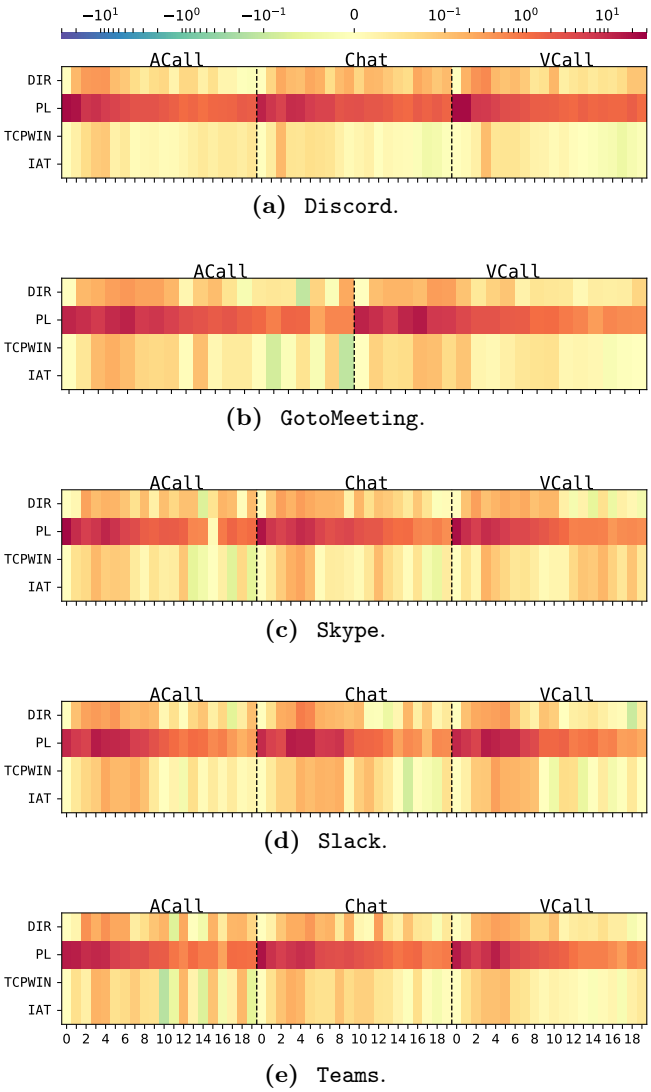
SEQ Input Now we discuss the importance of SEQ inputs on bSEQ branch. These inputs consist of four header fields extracted from the first $N_p = 20$ packets of each biflow. These fields include packet direction (DIR), transport-level payload-length (PL), TCP window size (TCPWIN), and inter-arrival time (IAT).

To better understand the importance of these fields, Fig. 7.3 shows the median importance value for each element of the 4×20 matrix representing the SEQ input. The graph provides a breakdown of the activity of different apps, such as Discord, GotoMeeting, Skype, and Teams[2].

Overall, we note that the greatest importance is attributed to the fields of the first 10 packets. Furthermore, it is worth noting that PL not only consistently contributes positively to the correct classification of the spe-

[1]For brevity, Meet, Messenger, and Zoom are omitted as they yield similar results to Webex, Skype, and Zoom, respectively.

[2]We omitted the results for Meet, Messenger, Webex, and Zoom, as their outcomes, are similar to those of GotoMeeting, Discord, GotoMeeting, and Teams, respectively



(a) `Discord`.

(b) `GotoMeeting`.

(c) `Skype`.

(d) `Slack`.

(e) `Teams`.

Figure 7.3. Median importance (in Symlog-scale) of `SEQ` Inputs (identified by the respective packet index) for bSEQ-branch of MIMETIC-ALL. Results refer to exemplifying apps—i.e., `Discord` (a), `GotoMeeting` (b), `Skype` (c), `Slack` (d), and `Teams` (d)—, based on the activity-type w.r.t. Joint-TC task. Each combination of app and activity is separately analyzed according to the correctly classified samples.

cific app and activity but is also the most important feature. This finding highlights the ability of the embedding layer to enhance the information derived from the sequence of PL that is used to feed bSEQ modality. Additionally, we find that DIR represents the second most important field, and in nearly all cases, it also contributes positively to the final decision.

On the contrary, the importance of TCPWIN and IAT is generally less significant, and in some instances, it might have a detrimental impact on the classification outcome. In fact, for ACall in GotoMeeting, Skype, and Teams, we observe that TCPWIN and IAT of certain packets within the second half (i.e., from 11^{th} to 20^{th}) exhibit negative values, which is not the case for other activities. This suggests that these fields could potentially lead to erroneous model decisions. Similar findings are also noticeable for Slack regarding Chat and VCall.

Context Herein, we analyze the importance of Context on bContext branch. Each target biflow is associated with Context comprising nine features extracted from its contextual traffic, i.e., the traffic carried by the biflows that run in parallel. These features include the amount of transmitted bytes (V_*), the number of transmitted packets (P_*), the bit-rate (BR_*), and the packet-rate (PR_*) in both upstream and downstream direction (we use $* = u$ and $* = d$ to denote upstream and downstream, respectively). The features also include the number of contextual biflows (#CF).

Fig. 7.4 depicts the median importance value of each feature composing Context, providing a per-activity breakdown for all apps.

Overall, we observe that in all apps and activities, the number of packets exchanged in both directions always positively impacts the final outcome. It is also worth noting that these features have a higher importance value than others, indicating that the number of packets exchanged is the most relevant factor in determining the final outcome. Additionally, we observe that the number of contextual biflows (i.e., #CF) assumes significant relative importance, particularly when classifying VCall in the cases of Discord, GotoMeeting, Meet, Teams, Webex, and Zoom.

In contrast, we note that features related to traffic volume (i.e., V_*) may lead the model to give the wrong result. For instance, VOL_{up} has negative importance on ACall (resp. Chat) for Discord, GotoMeeting, Messenger,

Slack, and Zoom (resp. Discord, Slack, Teams, and Zoom)[3]. In other cases, such as GotoMeeting, Slack, and Zoom, VOL_{up} also negatively impacts on VCall. However, the opposite holds for Meet, Skype, and Webex where VOL_{up} always has positive importance.

Finally, focusing on byte- and packet-rate (i.e., BR_{up} and BR_{dw}, and PR_{up} and PR_{dw}, respectively), we observe that the importance varies greatly depending on the type of app, activity, and feature considered. Particularly, byte-rate and packet-rate generally have positive and negative importance, respectively, on the classification of VCall in the case of Discord, Meet, and Slack. The same holds for ACall in the case of Slack. In contrast, an opposite trend can be observed for Chat in the case of Skype.

Takeaways

RQ: *How do* MIMETIC-ALL*'s traffic inputs affect joint App and Activity classification?*

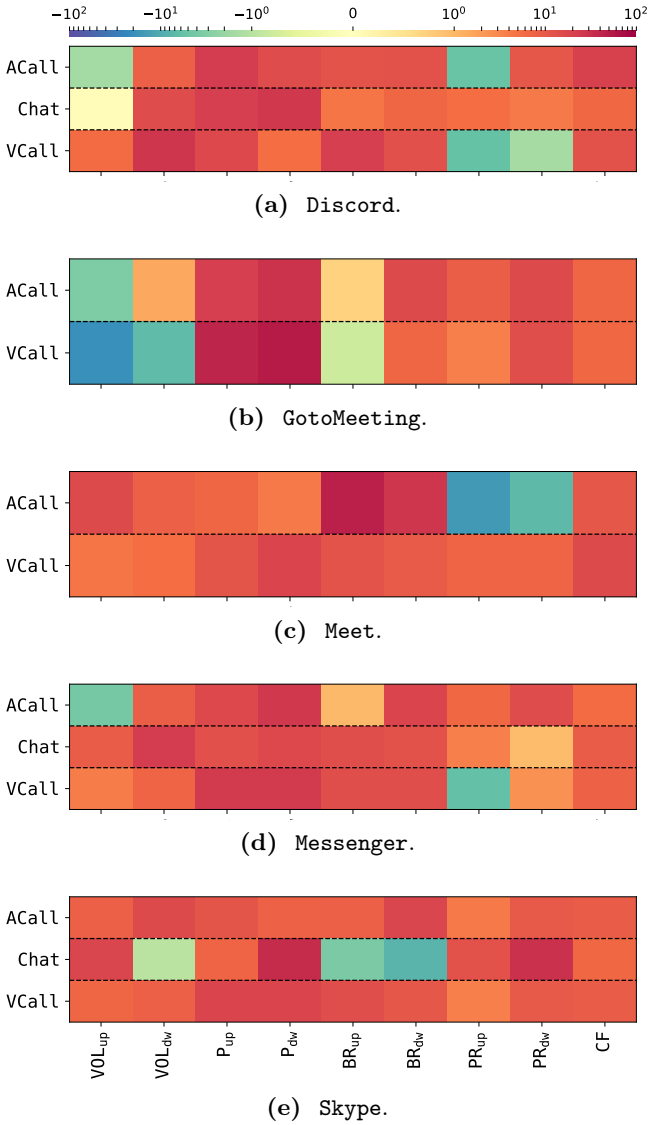
When analyzing different branches of MIMETIC-ALL, bPAY and bSEQ were found to be more important than bCONTEXT for Joint-TC. bPAY and bSEQ ranged from $30-50\%$, while bCONTEXT ranged from $12-30\%$.

For bPAY modality, bytes $32-320$ are crucial, while the first 32 and last 48 bytes are insignificant.

For bSEQ modality, the first ten packets are most important. PL is the crucial feature. DIR is second. TCPWIN and IAT are less significant and may negatively impact classification.

For bCONTEXT modality, the number of exchanged packets (i.e., PKTs_{up} and PKTs_{dw}) and contextual biflows (i.e., #CF) are crucial. However, traffic volumes (i.e., VOL_{up} and VOL_{dw}) can sometimes lead to incorrect results. The importance of byte- and packet-rates (BR_{up}, BR_{dw}, PR_{up}, and PR_{dw}) varies depending on the app and activity.

[3]Similar observations can also be made for VOL_{dw} concerning Chat (resp. VCall) in the case of Skype and Teams (resp. GotoMeeting).



(a) `Discord`.

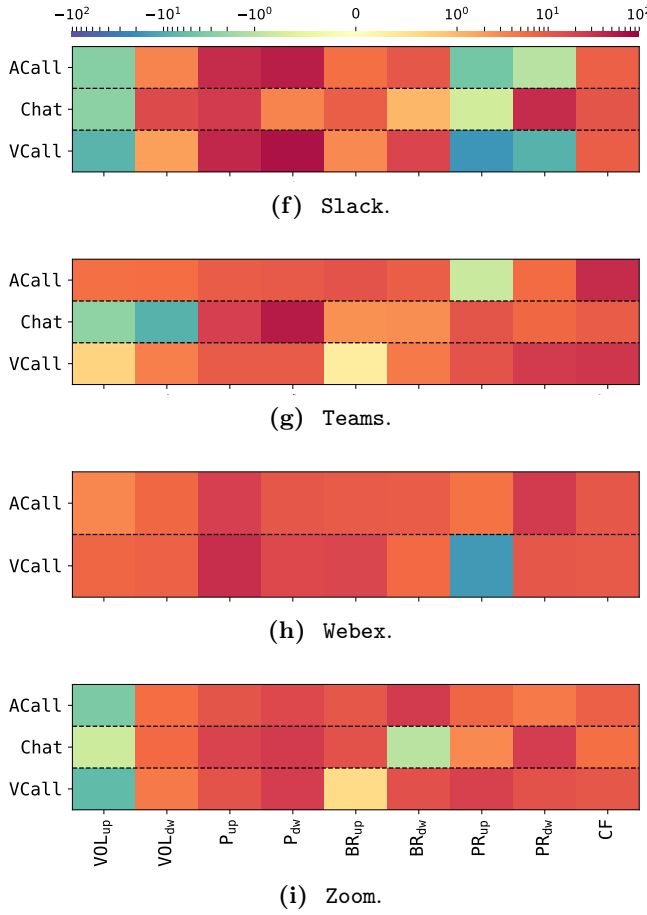
(b) `GotoMeeting`.

(c) `Meet`.

(d) `Messenger`.

(e) `Skype`.

Figure 7.4. Median importance (in Symlog-scale) of Context inputs for bContext-branch of Mimetic-All. Results refer to `Discord` (a), `GotoMeeting` (b), `Meet` (c), `Messenger` (d), and `Skype` (e) based on the activity-type w.r.t. Joint-TC task. Each combination of app and activity is separately analyzed according to the correctly classified samples.



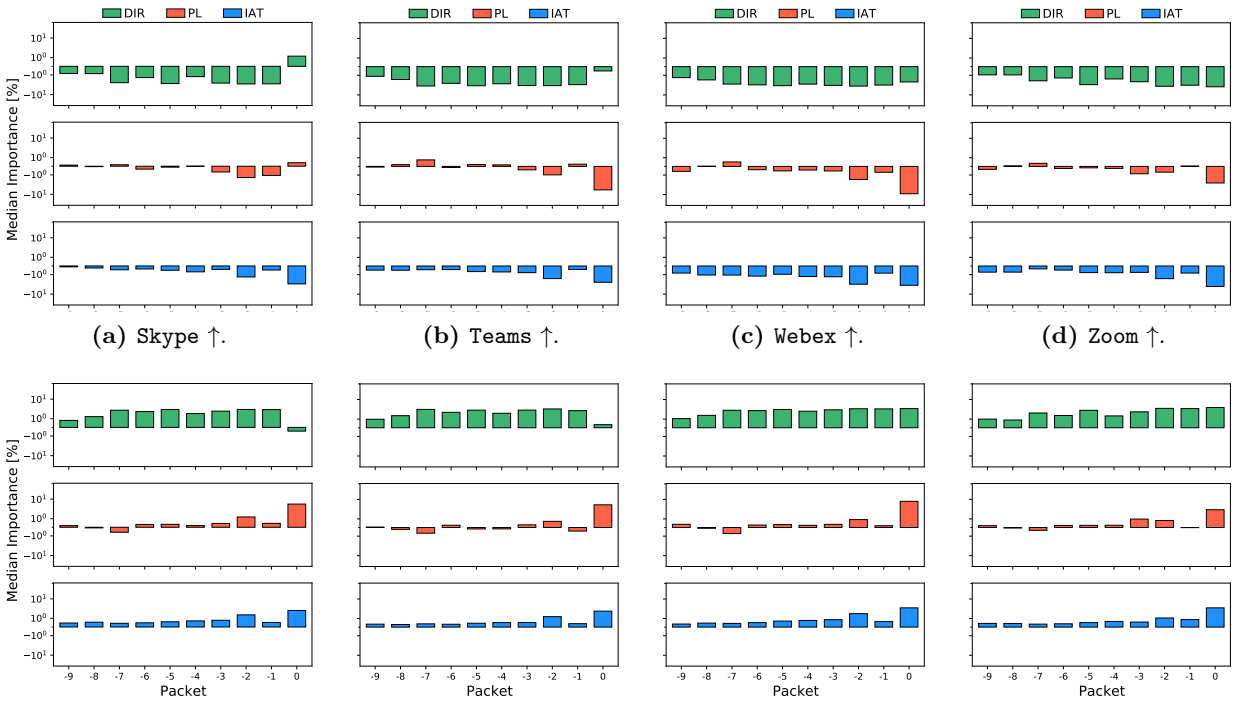
(f) `Slack`.

(g) `Teams`.

(h) `Webex`.

(i) `Zoom`.

Figure 7.4. Median importance (in Symlog-scale) of Context inputs for bCONTEXT-branch of MIMETIC-ALL. Results refer to `Slack` (f), `Teams` (g), `Webex` (h), and `Zoom` (i) based on the activity-type w.r.t. Joint-TC task. Each combination of app and activity is separately analyzed according to the correctly classified samples.



(a) Skype ↑. (b) Teams ↑. (c) Webex ↑. (d) Zoom ↑.

(e) Skype ↓. (f) Teams ↓. (g) Webex ↓. (h) Zoom ↓.

Figure 7.5. Median importance (in Symlog-scale) of each packet parameter (i.e., `DIR`, `PL`, and `IAT`) on the prediction of `DIR` of CNN trained on `TCP` for `Skype`, `Teams`, `Webex`, and `Zoom`. The x-axis reports the packet index in chronological order, with 0 for the last observed packet, −1 the previous one, and so forth. Each app is separately analyzed according to the (true) direction of packets, as upstream (↑) and downstream (↓) .

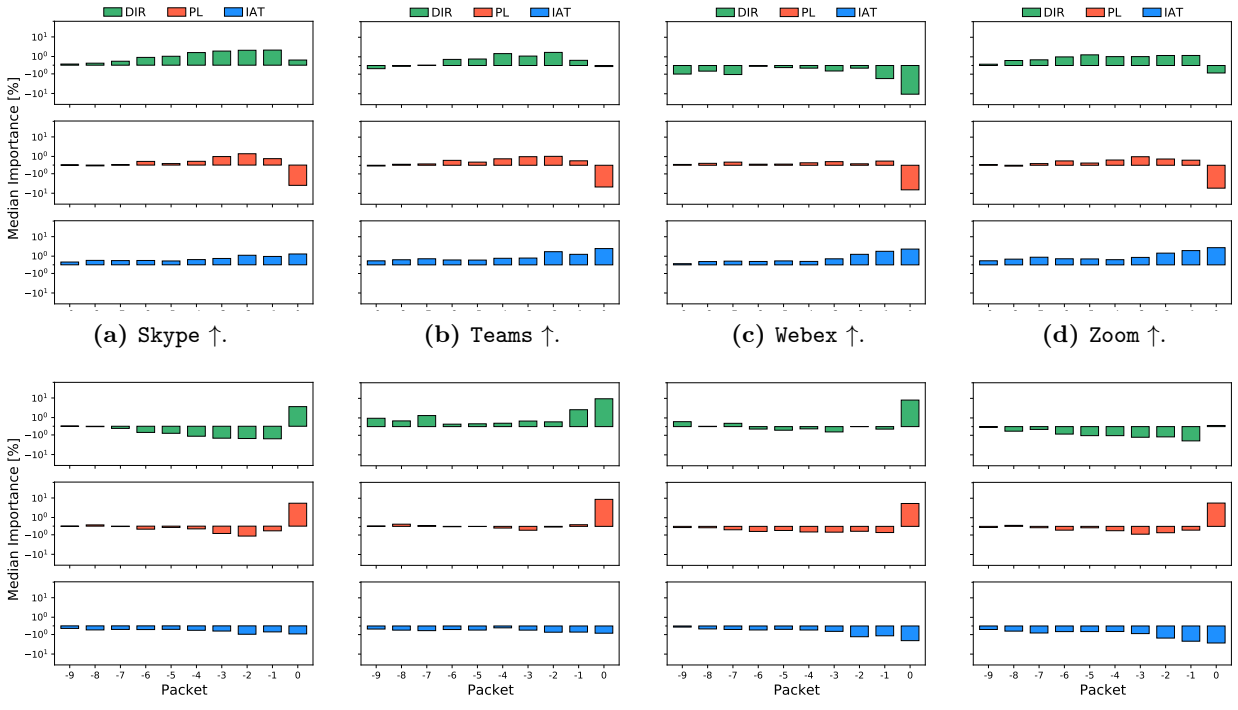


Figure 7.6. Median importance (in Symlog-scale) of each packet parameter (i.e., `DIR`, `PL`, and `IAT`) on the prediction of `DIR` of CNN trained on UDP for `Skype`, `Teams`, `Webex`, and `Zoom`. The x-axis reports the packet index in chronological order, with 0 for the last observed packet, -1 the previous one, and so forth. Each app is separately analyzed according to the (true) direction of packets, as upstream (↑) and downstream (↓) .

7.1.2 TP: How do inputs affect DIR prediction?

Herein, relying on Deep SHAP, we examine the relative influence of the three traffic parameters—i.e., DIR, PL, and IAT—extracted from the last W observed packets of each biflow on the prediction of DIR. To be more precise, we focus on samples whose DIR is correctly classified [71]. This selection allows us to focus on the accurate behavior of a DL-based traffic predictor and interpret its counter-intuitive (while right) decisions *a posteriori*.

As a result, Fig. 7.5 (resp. Fig. 7.6) depicts the median importance values across the last 10 packets (since $W = 10$) of each biflow, that are used to feed the model corresponding to TCP (resp. UDP) traffic to obtain the prediction of the next packet. Each figure provides a detailed breakdown of Skype, Teams, Zoom, and Webex according to the true direction of the next packet—i.e., upstream (↑) and downstream (↓).

On TCP traffic, we observe similar patterns for all apps. In particular, as depicted in Figs. 7.5a–7.5d (resp. 7.5e–7.5h), when the model correctly predicts the *upstream* (resp. *downstream*) direction, DIR and IAT of the last 10 observed packets always has a negative (resp. positive) effect on the prediction of the direction. This does not hold for the most recent packet of Skype, which tends to have the opposite effect on the prediction. Also, while the relative importance of DIR is fairly evenly distributed among all observed packets, for IAT, the greatest importance is attributed to the most recent packets. In general, we observe that DIR is the input that most significantly impacts the decision of the predictor. In contrast, we notice a polarity reversal between the less recent (i.e., the left side) and most recent (i.e., the right side) packets for PL. The former has a positive impact on the upstream direction and a negative impact on the downstream direction. Conversely, the latter has a negative impact on the upstream direction and a positive impact on the downstream direction.

Then, when looking at UDP traffic, for almost all apps, all observed input sequences positively contribute to the prediction when the model correctly predicts the *upstream* direction (↑, see Figs. 7.6a–7.6d), with the most recent packets usually having greater importance. Interestingly, for Webex, unlike PL and IAT, the DIR of the last 10 observed packets always has a negative effect on the prediction of the *upstream* direction. At the same time, when we examine DIR and PL of the last observed packet,

we find that it has no effect or even works against a proper direction prediction (negative score). This holds especially for `PL`, where for all apps, the last observed packet has a non-negligible negative importance—which also corresponds to the highest values in magnitude—suggesting that this generally leads the model to predict the *downstream* direction instead of the correct *upstream* one.

Conversely, moving to the prediction of *downstream* direction (↓, see Figs. 7.6h–7.6g), we notice an opposite trend, where the `DIR` and `PL` of the last packet positively impact the prediction. Furthermore, their importance (in terms of magnitude) is significantly greater than that of the other packets. Based on this observation, we can infer that the `DIR` and `PL` of the last packet lead the model to predict the *downstream* direction accurately.

Lastly, it is worth noting that, unlike all other apps, in the case of `Teams`, the `DIR` of all observed packets positively impacts the prediction of the downstream direction of the next packet.

Takeaways

RQ: *How do the last* 10 *packets affect the prediction of the direction of the next packet?*

The direction of the next packet can be predicted with reasonable accuracy by looking at the direction (viz. `DIR`), payload-length (viz. `PL`), and inter-arrival time (viz. `IAT`) of the last few packets. However, the importance of these parameters varies depending on the type of traffic (i.e., `TCP` or `UDP`) and the direction of the packet to be predicted (i.e., upstream or downstream).

In general, `DIR` is the most important traffic parameter for predicting the direction of the next packet, while `PL` and `IAT`, although important, impact the model decision less significantly.

Finally, input parameters of the last observed packet can be either highly beneficial (downstream case) or detrimental (upstream case) to predict the direction of the next packet properly.

7.2 Reliability via Calibration Analysis

Other than the performance of the considered DL-based traffic models, it is of great importance to evaluate the *reliability* to soft-estimates associated with the output provided by such models. Reliability evaluation is fundamental in many critical scenarios. It constitutes a building block of XAI since it assesses the degree of trustworthiness in providing prediction outputs with high confidence (or not).

Formally speaking, given an input sample $\boldsymbol{x}$ to the DL-based traffic model under analysis, we analyze the reliability of the confidence vector $\boldsymbol{p}(\boldsymbol{x}) = p_1(\boldsymbol{x}), \cdots, p_L(\boldsymbol{x})$ and of the confidence associated to the predicted class $\hat{p}(\boldsymbol{x}) = \max_{i=1,\cdots,L} p_i(\boldsymbol{x})$, where L is the number of classes.

In what follows, we introduce a graphical visualization and three metrics to assess calibration [103]. Indeed, a *confidence-calibrated* classifier is such that for each sample, the confidence of the predicted class $\hat{p}$ equals $P\{\hat{\ell} = \ell \mid \hat{p}\}$, where ℓ and $\hat{\ell}$ are the true and predicted labels, respectively. That is, when reporting a confidence of, e.g., 80%, the predictor actually reaches 80% accuracy—i.e., the confidence value was neither excessively optimistic nor pessimistic.

To visualize the above property for varying $\hat{p}$, we rely on the so-called *reliability diagrams*, which show the accuracy as a function of the confidence (i.e., $P\{\hat{\ell} = \ell \mid \hat{p}\}$) and compare it with the ideal $P\{\hat{\ell} = \ell \mid \hat{p}\} = \hat{p}$ identity line: a perfectly-calibrated classifier entails a reliability diagram corresponding to the identity function.

These diagrams are evaluated by partitioning the predictions into M equally-spaced bins and computing their accuracy. Let B_m be the set of evaluated samples such that the confidence associated with the predicted label falls within the interval $I_m \triangleq (\frac{m-1}{M}; \frac{m}{M}]$, the accuracy and confidence associated with the bin are:

$$\mathrm{acc}(B_m) = |B_m|^{-1} \sum_{n \in B_m} 1(\hat{\ell}_{(n)} = \ell_{(n)}). \tag{7.3}$$

$$\mathrm{conf}(B_m) = |B_m|^{-1} \sum_{n \in B_m} \hat{p}(n). \tag{7.4}$$

where $\ell_{(n)}$, $\hat{\ell}_{(n)} \triangleq \arg\max_{1,\cdots,L} p_i(s)$, and $\hat{p}(s) \triangleq \max_{1,\cdots,L} p_i(s)$ are the

true label, the predicted label, and the predicted confidence of the s^{th} sample, respectively. Since the confidence values vary in $[1/L, 1]$, we consider the starting point of the confidence interval to be $1/L$.

Accordingly, to obtain concise metrics of the deviation from a perfect calibration, we integrate the above diagrams with the *Expected Calibration Error (ECE)*, defined as

$$\text{ECE} \triangleq \mathbb{E}_{\hat{p}}\{|P\{\hat{\ell} = \ell \mid \hat{p}\} - \hat{p}|\}. \tag{7.5}$$

and the *Maximum Calibration Error (MCE)*, defined as

$$\text{MCE} \triangleq \max_{\hat{p}} \left|\Pr\left\{\hat{\ell} = \ell | \, \hat{p}\right\} - \hat{p}\right|. \tag{7.6}$$

The former metric represents the expected absolute deviation between the confidence and the confidence-conditional accuracy, whereas the latter is the maximum absolute deviation from the identity line [103]. They can be approximately calculated as

$$\text{ECE} \approx \sum_{m=1}^{M} \frac{|B_m|}{N} |\text{acc}(B_m) - \text{conf}(B_m)|. \tag{7.7}$$

and

$$\text{MCE} \approx \max_{m=1,\ldots,M} |\text{acc}(B_m) - \text{conf}(B_m)| \, . \tag{7.8}$$

respectively. The above expressions are based on the overall number of tested samples S and the averaged confidence within the bin B_m. The latter equals $\text{conf}(B_m) = |B_m|^{-1} \sum_{s \in B_m} \hat{p}_{(s)}$, where $\hat{p}(s) \triangleq \max_{i=1,\cdots,L}\{p_i(s)\}$ denotes the predicted confidence of the s^{th} sample.

The ECE and MCE metrics only consider the confidence in the predicted label, while ignoring the other scores in the softmax distribution. A *stronger* definition of calibration requires the probabilities of all the classes in the softmax distribution to be calibrated, namely to have p_i equal to $\Pr\{\ell = i | \, p_i\}$ for $i = 1, \ldots, L$, i.e., having 80% confidence for the i^{th} label leads to 80% probability of observing that app. A concise metric that relies on the above stronger calibration definition is the *Class-Wise Expected*

Calibration Error (CW-ECE) [104], defined as

$$\text{CW-ECE} \triangleq \frac{1}{L}\sum_{i=1}^{L}\mathbb{E}_{p_i}\left\{\left|\Pr\left\{\ell = i \,|\, p_i\right\} - p_i\right|\right\}. \tag{7.9}$$

Such a metric is evaluated as the class-wise sum

$$\text{CW-ECE} = \frac{1}{L}\sum_{i=1}^{L}\text{CW-ECE}_i. \tag{7.10}$$

where

$$\text{CW-ECE}_i \approx \sum_{m=1}^{M}\frac{|B_{m,i}|}{N}\left|\varrho(B_{m,i}) - \text{conf}(B_{m,i})\right|. \tag{7.11}$$

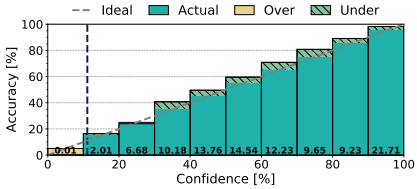
In the latter definition, $B_{m,i}$ denotes the set of samples whose prediction for the i^{th} label p_i falls within the m^{th} bin, and $\text{conf}(B_{m,i})$ (resp. $\varrho(B_{m,i})$) the corresponding bin-averaged confidence probability (resp. the proportion of samples labeled as the i^{th} label).

In what follows, we analyze the calibration of previous DL models used to either classify or predict traffic of CC apps. Firstly, we assess the calibration of the Mimetic-All architecture w.r.t. the Joint-TC task (Sec. 7.2.1).

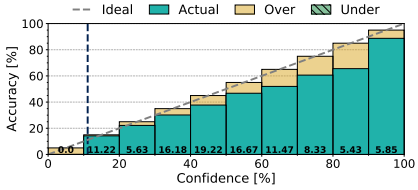
Secondly, we perform a similar analysis on CNNs trained with the `PROTO` strategy w.r.t. the `DIR` prediction task (Sec. 7.2.2). Regarding `DIR` prediction, it is worth noting that there are only two possible labels (i.e., $L = 2$) for both the true (viz. x_{dir}^{n+1}) and the predicted directions (viz. $\hat{x}dir^{n+1}$): *upstream* (viz. *up*) and *downstream* (viz. *dw*). This means that `DIR` prediction can be treated as a binary classification task. As a result, the confidence vector simplifies to $\boldsymbol{p}(\boldsymbol{x}) = [p_{up}(\boldsymbol{x})\, p_{dw}(\boldsymbol{x})]$, and the confidence score becomes $\hat{p}(\boldsymbol{x}) = \max\{p_{up}(\boldsymbol{x}), p_{dw}(\boldsymbol{x})\}$.

7.2.1 TC: How do `Context` affect reliability?

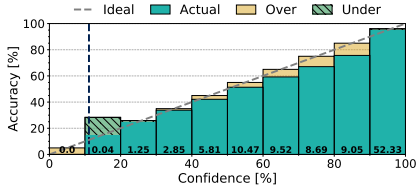
In this section, after demonstrating the effectiveness of Mimetic-Enhanced when tackling the Joint-TC task through the inclusion of `Context`, we evaluate its reliability [105] by means of a calibration analysis as described in Sec. 7.2. To this end, in Tab. 7.1, we report the ECE, MCE, and



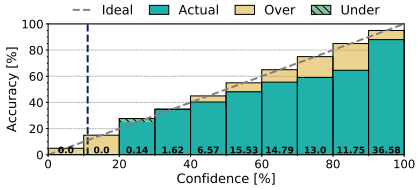
(a) bCONTEXT.



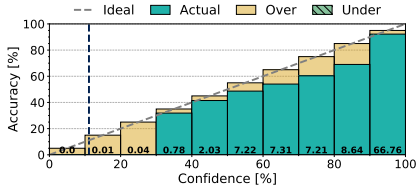
(b) bPAY.



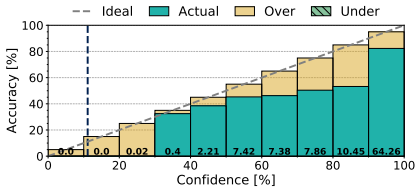
(c) MIMETIC-CONPAY.



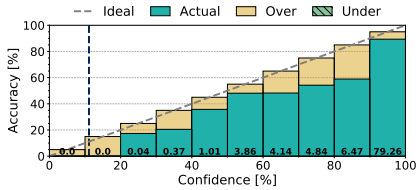
(d) bSEQ.



(e) MIMETIC-CONSEQ.



(f) MIMETIC-ENHANCED.



(g) MIMETIC-ALL.

Figure 7.7. Impact of Context inputs on calibration: reliability diagrams for APP×ACT-TC for classifiers characterized by different combinations of modalities (the input types fed to each classifier are shown in Tab. 5.2). Graphs in column allow to compare calibration without (upper row) and with (lower row) the Context inputs. Confidence is divided into 10 bins, and it is $\geq \frac{1}{L}$ (vertical dashed line), with L being the number of classes. Over and under gap represent an over-confident (optimistic) and under-confident (pessimistic) miscalibration pattern, respectively. The number at the bottom of each bar reports the percentage of samples within the corresponding bin.

Table 7.1. ECE, MCE, and Cw-ECE comparison of DL-based traffic classifiers when combining different mixes of modalities for Joint-TC. The input types fed to each classifier are shown in Tab. 5.2. For each metric (column), the best and the worst calibrated apps are highlighted in green and red, respectively.

Classifier	ECE [%]	MCE [%]	Cw-ECE [%]
bPAY	8.21 (±1.60)	19.79 (±4.14)	1.02 (±0.08)
bSEQ	11.08 (±1.86)	29.40 (±15.67)	1.12 (±0.13)
bContext	4.09 (±0.82)	9.41 (±2.50)	0.78 (±0.04)
Mimetic-Enhanced	17.99 (±0.87)	32.28 (±3.34)	1.60 (±0.06)
Mimetic-ConPay	4.11 (±1.03)	31.55 (±25.38)	0.63 (±0.05)
Mimetic-ConSeq	8.29 (±0.0.66)	24.05 (±4.36)	0.86 (±0.06)
Mimetic-All	11.5 (±0.94)	33.48 (±14.46)	1.07 (±0.06)

Cw-ECE values (in percentage form) obtained considering each single-modality branch (i.e., bPAY, bSEQ, and bContext) and their multi-modal combinations (i.e., Mimetic-Enhanced, Mimetic-ConPay, Mimetic-ConSeq, and Mimetic-All).

First of all, we observe that the calibration of each classifier varies greatly depending on the considered branches (viz., input types).

Focusing on individual inputs, we observe that using the sole `Context` (viz. bContext) results in the best-calibrated model with an ECE (resp. MCE) that is reduced by 2× (resp. 2×) and 2.7× (resp. 3×) compared to `PAY` (viz. bPAY) and `SEQ` (viz. bSEQ), respectively. Also, although more limited, the reduction of Cw-ECE is 1.3× and 1.4×, respectively.

In contrast, considering the sole `SEQ` input (i.e., bSEQ) only or combined with `PAY` (i.e., Mimetic-Enhanced), results in a model with an ECE $\in [11.8, 17.99]\%$ (resp. MCE $\in [29.4, 31.55]\%$), representing the classifiers with lower calibration (viz. reliability). Finally, it is interesting to note the benefit on model calibration due to the use of `Context`. In all cases where these are included, we obtain a calibration gain corresponding to a reduced ECE (resp. Cw-ECE) of up to -6.5% (resp. -0.5%). However, these benefits are not observable when looking at MCE. When considering `PAY` (i.e., Mimetic-ConPay) or both `PAY` and `SEQ` (i.e., Mimetic-All), the inclusion of `Context` leads to an increase of $+11.8\%$ and $+1.2\%$ of MCE, respectively.

Then, to visualize in detail how $P\{\hat{\ell} = \ell \mid \hat{p}\}$ varies with $\hat{p}$, in Fig. 7.7 we show the accuracy as a function of the confidence by means of the so-called *reliability diagrams*. Therein, each diagram is compared with the ideal $P\{\hat{\ell} = \ell \mid \hat{p}\} = \hat{p}$ identity line: a perfectly-calibrated classifier entails a reliability diagram corresponding to the identity function.

From inspection of the results, when considering the `PAY` and `SEQ` individually or in combination (see Figs. 7.7b, 7.7d, and 7.7f), the samples tend to be more evenly distributed within the bins. In contrast, when these inputs are combined with `Context` (see Figs. 7.7c, 7.7e, and 7.7g), we see that the samples tend to converge in the last bin (i.e., $[90\%, 100\%]$). This, in turn, means that the confidence of these classifiers *increases systematically*. Nonetheless, since the addition of `Context` also implies a *lower* ECE *for all the classifiers*, such increasingly-optimistic behavior is not paid at the expense of a decreased calibration. This clearly highlights a structural improvement of the classifiers' performance due to information originating from context biflows.

Takeaways

RQ: *How do `Context` affect the reliability of* MIMETIC-ENHANCED *when dealing with joint App and Activity classification?*

Classification reliability (viz. calibration) varies significantly based on the input types considered. Using `Context` alone leads to the best-calibrated model compared to other inputs (i.e., `PAY` and `SEQ`). Combining `Context` with other inputs consistently improves calibration without compromising accuracy, demonstrating a structural enhancement in classifier performance due to contextual information.

7.2.2 TP: How reliable is the prediction of `DIR`?

Herein, we analyze the effectiveness of CNNs that are trained using the `PROTO` strategy. We remark that this approach results in two separate models, one for `TCP` and another for `UDP` traffic. Our goal is to evaluate the reliability of these models in predicting `DIR`, using the calibration metrics outlined in Sec. 7.2. Specifically, we focus on the ECE, MCE, and

Table 7.2. ECE, MCE, and Cw-ECE related to DIR-prediction when using $\text{CNN}_{\text{PROTO}}$ with a memory size set to $W = 10$. Results refer to the prediction of traffic generated by each app, regardless of the specific activity. For a given protocol, for each metric (column) the best and worst calibrated apps are highlighted in green and red, respectively. Results are in the form $avg. \pm std.$ obtained over 5-folds.

Protocol	App	ECE [%]	MCE [%]	Cw-ECE [%]
TCP	Skype	1.83 (±1.80)	9.83 (±1.80)	2.13 (±1.80)
	Teams	1.64 (±0.71)	8.15 (±2.05)	2.82 (±1.48)
	Webex	3.20 (±1.60)	10.52 (±4.80)	5.93 (±3.40)
	Zoom	5.88 (±4.05)	12.55 (±5.79)	6.83 (±5.16)
UDP	Skype	3.72 (±2.80)	6.70 (±4.81)	5.16 (±2.24)
	Teams	5.22 (±4.26)	27.40 (±18.71)	7.34 (±7.72)
	Webex	3.48 (±1.73)	14.05 (±18.09)	5.99 (±2.19)
	Zoom	5.99 (±1.73)	10.36 (±2.58)	7.21 (±1.44)

Table 7.3. ECE, MCE, and Cw-ECE related to DIR-prediction when using $\text{CNN}_{\text{PROTO}}$ with a memory size set to $W = 10$. Results refer to the prediction of traffic generated by each activity, regardless of the specific app. For a given protocol, for each metric (column) the best and worst calibrated activities are highlighted in green and red, respectively. Results are in the form $avg. \pm std.$ obtained over 5-folds.

Protocol	Activity	ECE [%]	MCE [%]	Cw-ECE [%]
TCP	ACall	2.85 (±1.46)	6.08 (±2.60)	3.70 (±1.33)
	Chat	2.38 (±1.34)	9.12 (±4.14)	2.98 (±1.42)
	VCall	4.90 (±4.43)	9.17 (±4.46)	6.50 (±5.76)
UDP	ACall	2.33 (±0.74)	12.66 (±18.71)	3.56 (±0.84)
	Chat	8.14 (±2.51)	25.84 (±7.82)	9.28 (±2.447)
	VCall	3.56 (±1.93)	23.04 (±22.07)	4.31 (±1.81)

CW − ECE metrics. This evaluation helps us to determine whether these models are accurate and trustworthy in their predictions. To this end, in Tabs. 7.2 and 7.3, we report for each model (viz. traffic) the above values (in percentage form) obtained by considering the app and the activity performed by the user, respectively.

Focusing on the app on TCP, we note that in the case of Skype and

`Teams`, the model has the best calibration when used to predict the direction of the next packet with respect to all the metrics. This result is in agreement with Fig. 6.4a in which it is shown that the model achieves the best performance in the case of such apps (i.e., $\approx 90\%$ of `G-mean`). Conversely, the model is less calibrated, particularly in the case of `Zoom` for which all metrics are significantly higher (up to $3\times$ of ECE and Cw-ECE). Also, it is interesting to note that although `Webex` and `Zoom` exhibit similar performance, in the latter case, the model is less calibrated (i.e., up to $\approx 2\times$ of ECE). Then, moving on to `UDP`, we note that in the case of `Skype`, the model exhibits the best overall calibration when considering all metrics simultaneously. Once again, this result is in agreement with Fig. 6.4a, where `Skype` corresponds to the best performance on UDP traffic (i.e. $\approx 80\%$ of `G-mean`). In contrast, the calibration corresponding to the remaining apps varies greatly depending on the metric considered. In fact, when analyzing ECE, it is worth noting that the calibration is lowest when dealing with `Zoom` (i.e., the most difficult to predict in terms of `DIR`). On the other hand, when looking at MCE and Cw-ECE, the lowest calibration is associated with `Teams`. Finally, moving to the activity (see Tab. 7.3), we note that on `TCP` (resp. `UDP`) traffic, while for `ACall` and `Chat` (resp. `ACall` and `VCall`) the model has a good calibration—except for `Chat` (`VCall`) in terms of MCE—, this does not hold for `VCall` (resp. `Chat`) to which corresponds an ECE, an MCE, and an $\mathrm{CW}-\mathrm{ECE}$ that are up to $2\times$ ($3.5\times$), $1.5\times(2\times)$, and $2\times(2.6\times)$ higher, respectively.

Then, to visualize in detail how $P\{\hat{\ell} = \ell \mid \hat{p}\}$ varies with $\hat{p}$, in Figs. 7.8–7.9, we show the accuracy as a function of the confidence by means of the so-called *reliability diagrams*. Therein, we remark that each diagram is compared with the ideal $P\{\hat{\ell} = \ell \mid \hat{p}\} = \hat{p}$ identity line: a perfectly-calibrated classifier entails a reliability diagram corresponding to the identity function.

Looking at the results obtained per app, we note that on `UDP`, the samples are uniformly distributed across the bins. Moreover, as highlighted in Figs. 7.8b, 7.8b, and 7.8f, for `Skype`, `Teams`, and `Webex` we observe a near-ideal behavior with a slight under- and over-confidence affecting the first and second half of bins, respectively. At the same time, referring to Figs. 7.8h, in the case of `Zoom`, a more pronounced over-confidence affecting the final bins can be observed. Then, when moving on `TCP`, we observe that

for all apps (except for Zoom), a significant proportion of samples, ranging from 44% to 75%, fall into the last bin, corresponding to a nearly-ideal behavior. On the other hand, the model tends to be notably less confident for samples falling into bins $2-7$. In the case of Zoom, we observe a uniform distribution of samples across the bins, for which we observe pronounced over-confidence for those that fall in bins $3-9$.

Finally, looking at the specific activity (see Fig. 7.9), for ACall and VCall, while the related prediction tasks present a near-ideal behavior for UDP, in the case of TCP they correspond to more pronounced under- and over-confidence with respect to the first and the second half of the bins, respectively. Moreover, extending the analysis to Chat, we note opposite behaviors depending on the specific traffic type. In fact, while for TCP we observe good confidence levels, in the case of UDP the model is significantly over-confident for all samples, regardless of the bins in which they fall.

Takeaways

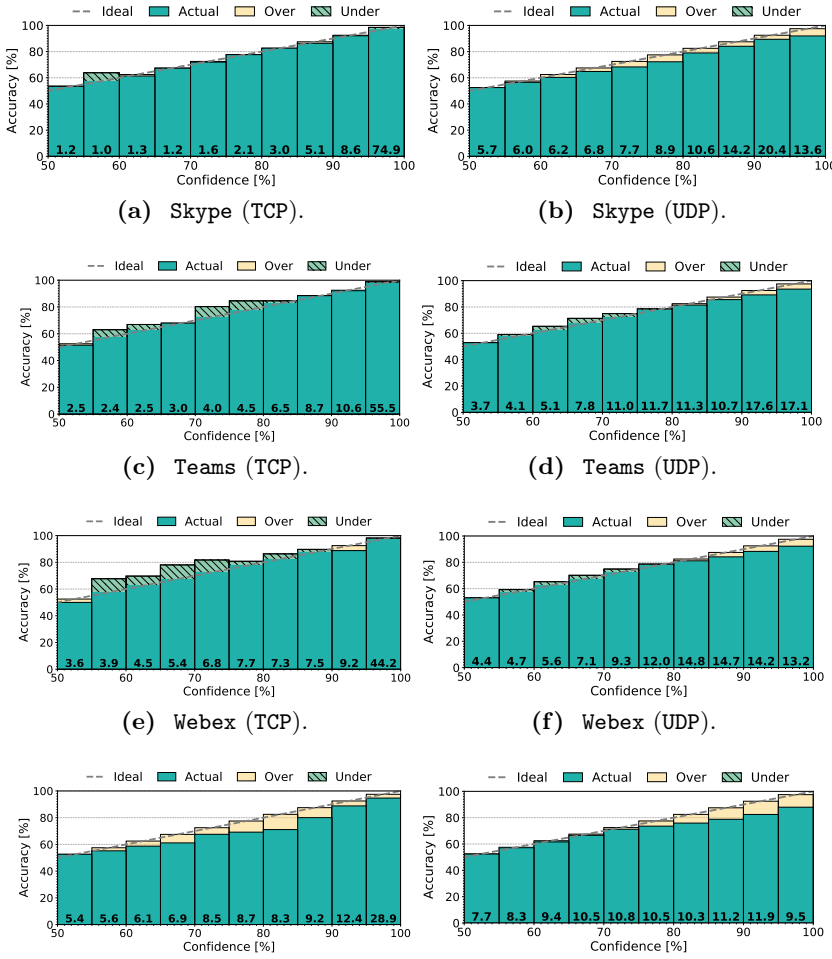
RQ: *How reliable is the* DIR *prediction of CNNs trained with the* PROTO *strategy?*

Generally, predictors show almost excellent calibration, which is affected by the type of traffic, the app, and user activity.

On TCP traffic, Skype and Teams exhibit the best calibration, consistent with their high performance. In contrast, the lowest calibration is related to Zoom—i.e., the hardest to be predicted.

On UDP traffic, Skype corresponds to the best overall calibration, aligned with its strong performance. However, calibration varies for other apps based on the metric used.

User activities show varying levels of calibration, with ACall and Chat (resp. VCall) exhibiting good calibration on TCP (resp. UDP) traffic. In contrast, Chat (resp. VCall) shows significantly lower calibration performance.



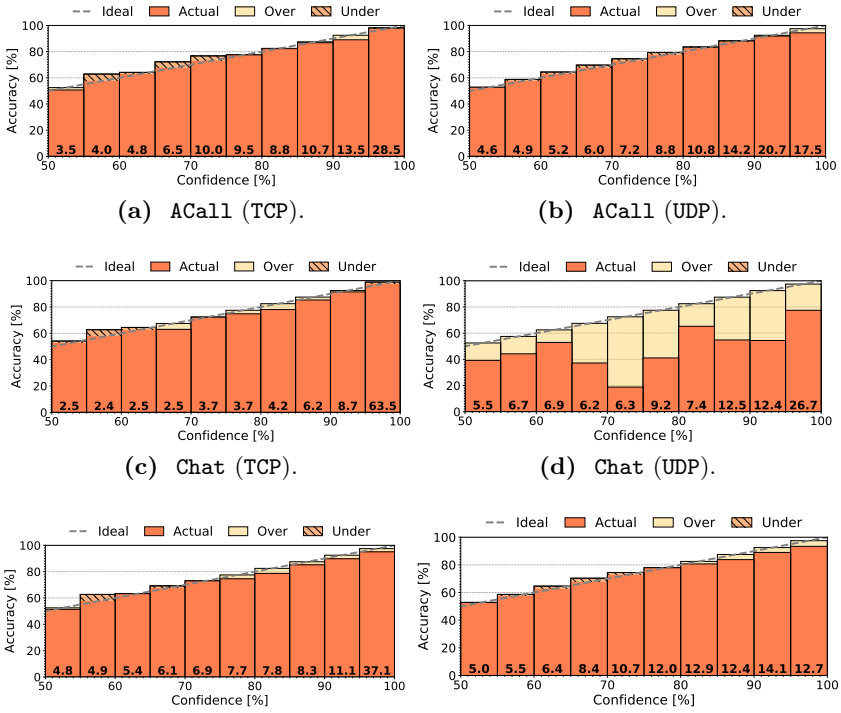
(a) Skype (TCP). **(b)** Skype (UDP).

(c) Teams (TCP). **(d)** Teams (UDP).

(e) Webex (TCP). **(f)** Webex (UDP).

(g) Zoom (TCP). **(h)** Zoom (UDP).

Figure 7.8. Reliability diagrams related to DIR-prediction task for Skype (a), Teams (b), Webex (c), and Zoom (d) when using $\text{CNN}_{\text{PROTO}}$ trained on TCP (*left side*) and UDP (*right side*) traffic. As the DIR-prediction task represents a binary classification problem, the confidence interval varies in the range [50%, 100%]. Confidence is divided in 10 bins. Over and under gap represent an over-confident (optimistic) and under-confident (pessimistic) miscalibration pattern, respectively. The number at the bottom of each bar reports the percentage of samples within the corresponding bin.



(a) `ACall` (`TCP`). **(b)** `ACall` (`UDP`).

(c) `Chat` (`TCP`). **(d)** `Chat` (`UDP`).

(e) `VCall` (`TCP`). **(f)** `VCall` (`UDP`).

Figure 7.9. Reliability diagrams related to `DIR`-prediction task for `ACall` (a), `Chat` (b), `VCall` (c)) when using $\texttt{CNN}_{\texttt{PROTO}}$ trained on `TCP` (*left side*) and `UDP` (*right side*) traffic. As the `DIR`-prediction task represents a binary classification problem, the confidence interval varies in the range [50%, 100%]. Confidence is divided in 10 bins. Over and under gap represent an over-confident (optimistic) and under-confident (pessimistic) miscalibration pattern, respectively. The number at the bottom of each bar reports the percentage of samples within the corresponding bin.

Chapter 8

Conclusions

The COVID pandemic has increased the use of *Communication and Collaboration mobile apps (CC apps)*. With the pandemic becoming endemic in some parts of the world, the persistent use of CC apps has led to a continuous increase in their share in Internet traffic. This has called for improved network monitoring and management tools, especially given the multi-activity nature of such apps. Therefore, we focused on designing and evaluating *Deep Learning (DL)* techniques for classifying and predicting mobile Internet traffic generated by CC apps. These techniques are paramount for managing network infrastructures and services, particularly during and after the COVID pandemic. We relied on a *newly collected dataset*—named MIRAGE-COVID-CCMA-2022 which we publicly released—covering *nine popular CC apps* and *three user activities each* and including over 300 hours of network traffic.

Since understanding the peculiarities of network traffic is crucial for tasks such as traffic classification (TC) and prediction (TP), we began by *characterizing and modeling* it from different perspectives regarding the apps and the corresponding activities. We found that TLS and SSL biflows are dominant, but UDP biflows account for most transmitted packets. We also discovered that the typical inputs used to feed traffic classifiers have shortcomings in capturing the characteristics of activities corresponding to the same app. In contrast, for the same combinations of apps and activities, we delineated distinctiveness in their aggregate traffic (e.g., downstream/upstream bit rates and number of coexisting biflows). Finally, we

provided a modeling analysis via Markov Chains, highlighting similar transition patterns for different apps performing the same activity.

Hence, we focused on *fine-grained TC* of both apps and activities, facing three tasks (App-TC, Act-TC, and Joint-TC) using different training strategies. We compared both single- and multi-modal state-of-the-art DL models and found that they show good performance only for the less tough App-TC. To overcome this limitation, we devised a novel set of Context inputs (viz. `Context`) that aids the effective (early) classification of activities, leading to a novel multimodal DL-based traffic classifier named MIMETIC-ALL. This classifier resulted in the best trade-off between performance and complexity. Moreover, `Context` were also proven as excellent substitutes for payload-based inputs, trading a small loss in App-TC for substantially improved performance in Act-TC and Joint-TC. Also, their use entails a significant reduction of memory footprint and training time, besides the intrinsic greater robustness to future more opaque encryption sublayers.

Additionally, we tackled *packet-level TP* via multi-task DL approaches, defining different training strategies. Our findings revealed that a 1D-CNN trained on traffic of all CC apps represents a better trade-off than per-app models. Moreover, a specialized model for TCP traffic led to significant performance improvements across all prediction tasks. Taking advantage of packet-level prediction approaches, we tackled coarser-granularity prediction (traffic volume and number of packets in a given time interval). Results showed varying behaviors across different apps, activities, and transport protocols for both fine- and coarse-grained prediction.

Finally, we leveraged *XAI approaches* (i.e., DEEP SHAP) to gain insights into the black-box nature of these DL models and data, and assess the reliability of their outcomes (via a calibration analysis). We evaluated the importance of the inputs used by MIMETIC-ALL, showing that Joint-TC is mostly influenced by the payload bytes (`PAY`) and the feature sequences of the first packets of the biflows (`SEQ`). Nevertheless, `Context` still play a crucial role in solving this task, particularly the number of exchanged packets and of contextual biflows exhibited the greatest importance. Notably, we found that `Context` is also beneficial to improve the general calibration of our proposal (as well as the considered variants), thus providing a structural improvement of the classification model. In

addition, we analyzed how the accurate prediction of DIR is affected by the last observed packets used as inputs to DL predictors. Our findings indicated that the most recent packets mainly influence the model predictions, and the input parameters of the last observed packet can be either very informative or confounding for the model. However, the importance of the observed parameters varies depending on the type of traffic (TCP vs. UDP) and the direction of the packet to be predicted (upstream vs. downstream). Upon calibration analysis, the predictor proved to be well-calibrated for predicting DIR, with reliability mainly influenced by traffic type, app used, and user activity.

Bibliography

[1] Sandvine. The Global Internet Phenomena Report 2023., Jan 2023.

[2] Anja Feldmann, Oliver Gasser, Franziska Lichtblau, Enric Pujol, Ingmar Poese, Christoph Dietzel, Daniel Wagner, Matthias Wichtlhuber, Juan Tapiador, Narseo Vallina-Rodriguez, Oliver Hohlfeld, and Georgios Smaragdakis. The lockdown effect: Implications of the COVID-19 pandemic on Internet traffic. In *ACM Internet Measurement Conference (IMC)*, page 1–18, 2020. doi: 10.1145/3419394.3423658.

[3] Andra Lutu, Diego Perino, Marcelo Bagnulo, Enrique Frias-Martinez, and Javad Khangosstar. A characterization of the COVID-19 pandemic impact on a mobile network operator traffic. In *ACM Internet Measurement Conference (IMC)*, page 19–33, 2020. ISBN 9781450381383. doi: 10.1145/3419394.3423655.

[4] Thomas Favale, Francesca Soro, Martino Trevisan, Idilio Drago, and Marco Mellia. Campus traffic and e-Learning during COVID-19 pandemic. *Computer Networks*, 176:107290, 2020. ISSN 1389-1286. doi: 10.1016/j.comnet.2020.107290.

[5] Alisha Ukani, Ariana Mirian, and Alex C. Snoeren. Locked-in during lockdown. In *21st ACM Internet Measurement Conference (IMC)*, pages 480–486, 11 2021. doi: 10.1145/3487552.3487828.

[6] Mehdi Karamollahi, Carey Williamson, and Martin Arlitt. Zoomiversity: A case study of pandemic effects on post-secondary teaching and learning. In Oliver Hohlfeld, Giovane Moura, and Cristel Pelsser, editors, *Passive and Active Measurement*, pages 573–599, Cham, 2022. Springer International Publishing. ISBN 978-3-030-98785-5. doi: 10.1007/978-3-030-98785-5_26.

[7] Albert Choi, Mehdi Karamollahi, Carey Williamson, and Martin Arlitt. Zoom session quality: A network-level view. In Oliver Hohlfeld, Giovane Moura, and Cristel Pelsser, editors, *Passive and Active Measurement*, pages 555–572, Cham, 2022. Springer International Publishing. ISBN 978-3-030-98785-5. doi: 10.1007/978-3-030-98785-5_25.

[8] Massimo Candela, Valerio Luconi, and Alessio Vecchio. Impact of the COVID-19 pandemic on the Internet latency: A large-scale study. *Computer Networks*, 182:107495, 2020. ISSN 1389-1286. doi: 10.1016/j.comnet.2020.107495.

[9] Timm Böttger, Ghida Ibrahim, and Ben Vallis. How the Internet reacted to COVID-19: A perspective from Facebook's edge network. In *ACM Internet Measurement Conference (IMC)*, page 34–41, 2020. ISBN 9781450381383. doi: 10.1145/3419394.3423621.

[10] Erica Stutz, Yuri Pradkin, Xiao Song, and John Heidemann. Visualizing Internet measurements of Covid-19 work-from-home. In *IEEE International Conference on Big Data (Big Data)*, pages 5633–5638, 2021. doi: 10.1109/BigData52589.2021.9671311.

[11] A. Dainotti, A. Pescapè, and K. C. Claffy. Issues and future directions in traffic classification. *IEEE Network*, 26(1):35–40, 2012.

[12] Luis Velasco, Ramon Casellas, Sergio Llana, Lluís Gifre, Ricardo Martínez, Ricard Vilalta, Raúl Muñoz, and Marc Ruiz. A control and management architecture supporting autonomic NFV services. *Photonic Network Communications*, 37(1):24–37, 2019.

[13] Alberto Dainotti, Antonio Pescapé, P. Salvo Rossi, Francesco Palmieri, and Giorgio Ventre. Internet traffic modeling by means of Hidden Markov Models. *Elsevier Computer Networks*, 52(14):2645–2662, 2008. doi: 10.1016/j.comnet.2008.05.004.

[14] Arcangela Rago, Giuseppe Piro, Gennaro Boggia, and Paolo Dini. Multi-task learning at the mobile edge: An effective way to combine traffic classification and prediction. *IEEE Transactions on Vehicular Technology*, 69 (9):10362–10374, 2020. doi: 10.1109/TVT.2020.3005724.

[15] Statista. Number of available applications in the Google Play Store from December 2009 to June 2023., Sep 2023.

[16] Statista. Number of available apps in the Apple App Store from 1st quarter 2015 to 3rd quarter 2022., Aug 2023.

[17] Eric Liang, Hang Zhu, Xin Jin, and Ion Stoica. Neural packet classification. In *Proceedings of the ACM Special Interest Group on Data Communication*, page 256–269. Association for Computing Machinery, 2019. doi: 10.1145/3341302.3342221.

[18] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé. Mobile encrypted traffic classification using Deep Learning: Experimental evaluation, lessons learned, and challenges. *IEEE Trans. Netw. Service Manag.*, 16(2):445–458, 2019. doi: 10.1109/TNSM.2019.2899085.

[19] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, Valerio Persico, and Antonio Pescape. Ai-powered internet traffic classification: Past, present, and future. *IEEE Communications Magazine*, 2023.

[20] Martin Henze, Jan Pennekamp, David Hellmanns, Erik Mühmer, Jan Henrik Ziegeldorf, Arthur Drichel, and Klaus Wehrle. Cloudanalyzer: Uncovering the cloud usage of mobile apps. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 262–271, 2017. doi: 10.1145/3144457.3144471.

[21] Weiwei Jiang. Cellular traffic prediction with machine learning: A survey. *Expert Systems with Applications*, 201:117163, 2022.

[22] Gabriel O Ferreira, Chiara Ravazzi, Fabrizio Dabbene, Giuseppe C Calafiore, and Marco Fiore. Forecasting network traffic: A survey and tutorial with open-source comparative evaluation. *IEEE Access*, 2023.

[23] Antonio Montieri, Giampaolo Bovenzi, Giuseppe Aceto, Domenico Ciuonzo, Valerio Persico, and Antonio Pescapè. Packet-level prediction of mobile-app traffic using multitask Deep Learning. *Computer Networks*, 200:108529, 2021. ISSN 1389-1286. doi: 10.1016/j.comnet.2021.108529.

[24] Ying Zheng, Ziyu Liu, Xinyu You, Yuedong Xu, and Junchen Jiang. Demystifying deep learning in networking. In *Proceedings of the 2nd Asia-Pacific Workshop on Networking*, APNet '18, page 1–7, New York, NY, USA, 2018. Association for Computing Machinery. doi: 10.1145/3232565.3232569.

[25] European Commission. Ethics guidelines for trustworthy ai. https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai, Apr 2019. Online; accessed Oct. 2023.

[26] European Commission. A european approach to artificial intelligence. https://digital-strategy.ec.europa.eu/en/policies/

european-approach-artificial-intelligence, Dec 2023. Online; accessed Dec. 2023.

[27] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. End-to-end encrypted Traffic Classification with one-dimensional convolution neural networks. In *IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 43–48, 2017. doi: 10.1109/ISI.2017.8004872.

[28] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret. Network traffic classifier with convolutional and recurrent neural networks for Internet of Things. *IEEE Access*, 5:18042–18050, 2017. doi: 10.1109/ACCESS.2017.2747560.

[29] Alfredo Nascita, Antonio Montieri, Giuseppe Aceto, Domenico Ciuonzo, Valerio Persico, and Antonio Pescapé. XAI meets mobile traffic classification: Understanding and improving multimodal deep learning architectures. *IEEE Transactions on Network and Service Management*, 18(4): 4225–4246, 2021. doi: 10.1109/TNSM.2021.3098157.

[30] Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher A. Wood. TLS Encrypted Client Hello. Internet-Draft draft-ietf-tls-esni-14, Internet Engineering Task Force, February 2022. URL https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-14. Work in Progress.

[31] Antonia Affinito, Alessio Botta, and Giorgio Ventre. The impact of COVID on network utilization: an analysis on domain popularity. In *IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 1–6, 2020. doi: 10.1109/CAMAD50429.2020.9209302.

[32] Zhanyi Wang, Chuanming Huang, Zhuo Zhang, and Bo Liu. The Applications of Deep Learning on Traffic Identification., 2015.

[33] He Huang, Haojiang Deng, Jun Chen, Luchao Han, and Wei Wang. Automatic multi-task learning system for abnormal network traffic detection. *Int. Journal of Emerging Technologies in Learning*, 13(4):4–20, 2018. doi: 10.3991/ijet.v13i04.8466.

[34] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapè. MIMETIC: mobile encrypted traffic classification using multimodal deep learning. *Elsevier Computer Networks*, 165:106944, 2019. doi: 10.1016/j.comnet.2019.106944.

[35] Chang Liu, Longtao He, Gang Xiong, Zigang Cao, and Zhen Li. FS-Net: A flow sequence network for encrypted traffic classification. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1171–1179, 2019. doi: 10.1109/INFOCOM.2019.8737507.

[36] Shahbaz Rezaei, Bryce Kroencke, and Xin Liu. Large-scale mobile app identification using deep learning. *IEEE Access*, 8:348–362, 2020. doi: 10.1109/ACCESS.2019.2962018.

[37] Yi Zeng, Huaxi Gu, Wenting Wei, and Yantao Guo. *Deep-Full-Range*: A deep learning based network encrypted traffic classification and intrusion detection framework. *IEEE Access*, 7:45182–45190, 2019. doi: 10.1109/ACCESS.2019.2908225.

[38] Ying Zhao, Junjun Chen, Di Wu, Jian Teng, and Shui Yu. Multi-task network anomaly detection using federated learning. In *ACM 10th International Symposium on Information and Communication Technology (SoICT)*, pages 273–279, 2019. doi: 10.1145/3368926.3369705.

[39] 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): 1999–2012, 2020. doi: 10.1007/s00500-019-04030-2.

[40] Xin Wang, Shuhui Chen, and Jinshu Su. Automatic Mobile App Identification From Encrypted Traffic With Hybrid Neural Networks. *IEEE Access*, 8:182065–182077, 2020.

[41] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. DISTILLER: Encrypted traffic classification via multimodal multitask deep learning. *Journal of Network and Computer Applications*, 183: 102985, 2021. doi: 10.1016/j.jnca.2021.102985.

[42] Iman Akbari, Mohammad A. Salahuddin, Leni Ven, Noura Limam, Raouf Boutaba, Bertrand Mathieu, Stephanie Moteau, and Stephane Tuffin. A look behind the curtain: Traffic classification in an increasingly encrypted web. *Proc. ACM Meas. Anal. Comput. Syst.*, 5(1), feb 2021. doi: 10.1145/3447382.

[43] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde. Analyzing android encrypted network traffic to identify user actions. *IEEE Trans. Inf. Forensics Security*, 11(1):114–125, 2016. doi: 10.1109/TIFS.2015.2478741.

[44] B. Saltaformaggio, H. Choi, K. Johnson, Y. Kwon, Q. Zhang, X. Zhang, D. Xu, and J. Qian. Eavesdropping on fine-grained user activities within smartphone apps over encrypted network traffic. In *USENIX Workshop on Offensive Technologies (WOOT)*, pages 69–78, 2016.

[45] Edita Grolman, Andrey Finkelshtein, Rami Puzis, Asaf Shabtai, Gershon Celniker, Ziv Katzir, and Liron Rosenfeld. Transfer learning for user action identication in mobile apps via encrypted traffic analysis. *IEEE Intelligent Systems*, 33(2):40–53, 2018. doi: 10.1109/MIS.2018.111145120.

[46] Fabio Aiolli, Mauro Conti, Ankit Gangwal, and Mirko Polato. Mind your wallet's privacy: identifying bitcoin wallet apps and user's actions through network traffic analysis. In *34th ACM/SIGAPP Symposium on Applied Computing (SAC)*, pages 1484–1491, 2019. doi: 10.1145/3297280.3297430.

[47] Ding Li, Wenzhong Li, Xiaoliang Wang, Cam-Tu Nguyen, and Sanglu Lu. App trajectory recognition over encrypted internet traffic based on deep neural network. *Computer Networks*, 179:107372, 2020. ISSN 1389-1286. doi: 10.1016/j.comnet.2020.107372.

[48] Changbing Li, Chao Dong, Kai Niu, and Zhengzhen Zhang. Mobile service traffic classification based on joint deep learning with attention mechanism. *IEEE Access*, 9:74729–74738, 2021. doi: 10.1109/ACCESS.2021.3081504.

[49] 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, 2018. doi: 10.1109/TIFS.2017.2737970.

[50] Giuseppe Aceto, Giampaolo Bovenzi, Domenico Ciuonzo, Antonio Montieri, Valerio Persico, and Antonio Pescapé. Characterization and Prediction of Mobile-App Traffic Using Markov Modeling. *IEEE Transactions on Network and Service Management*, 18(1):907–925, 2021. doi: 10.1109/TNSM.2021.3051381.

[51] C. Huang, C. Chiang, and Q. Li. A study of deep learning networks on mobile traffic forecasting. In *IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–6, 2017. doi: 10.1109/PIMRC.2017.8292737.

[52] Nipun Ramakrishnan and Tarun Soni. Network traffic prediction using recurrent neural networks. In *17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 187–193, 2018. doi: 10.1109/ICMLA.2018.00035.

[53] Yuxiu Hua, Zhifeng Zhao, Rongpeng Li, Xianfu Chen, Zhiming Liu, and Honggang Zhang. Deep learning with long short-term memory for time series prediction. *IEEE Communications Magazine*, 57(6):114–119, 2019. doi: 10.1109/MCOM.2019.1800155.

[54] Yonghua Huo, Yu Yan, Dan Du, Zhihao Wang, Yixin Zhang, and Yang Yang. Long-term span traffic prediction model based on STL decomposition and LSTM. In *20th IEEE Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–4, 2019. doi: 10.23919/APNOMS.2019.8892991.

[55] Q. He, A. Moayyedi, G. Dán, G. P. Koudouridis, and P. Tengkvist. A Meta-Learning Scheme for Adaptive Short-Term Network Traffic Prediction. *IEEE Journal on Selected Areas in Communications*, 38(10):2271–2283, 2020. doi: 10.1109/JSAC.2020.3000408.

[56] Sajal Saha, Anwar Haque, and Greg Sidebottom. Deep Sequence Modeling for Anomalous ISP Traffic Prediction. In *IEEE International Conference on Communications*, pages 5439–5444, 2022. doi: 10.1109/ICC45855.2022.9838262.

[57] Tiago Prado Oliveira, Jamil Salem Barbar, and Alexsandro Santos Soares. Computer network traffic prediction: a comparison between traditional and deep learning neural networks. *International Journal of Big Data Intelligence*, 3(1):28–37, 2016. doi: 10.1504/IJBDI.2016.073903.

[58] Yini Fang, Salih Ergüt, and Paul Patras. SDGNet: a handover-aware spatiotemporal graph neural network for mobile traffic forecasting. *IEEE Commununications Letters*, 26(3):582–586, 2022. doi: 10.1109/LCOMM.2022.3141238.

[59] Chaoyun Zhang, Marco Fiore, and Paul Patras. Multi-Service mobile traffic forecasting via convolutional long Short-Term memories. In *IEEE International Symposium on Measurements & Networking (M&N)*, pages 1–6, 2019. doi: 10.1109/IWMN.2019.8804984.

[60] S. Tanwir, D. Nayak, and H. Perros. Modeling 3D video traffic using a Markov modulated gamma process. In *IEEE International Conference on Computing, Networking and Communications (ICNC)*, pages 1–6, 2016. doi: 10.1109/ICCNC.2016.7440638.

[61] Athina Kalampogia and Polychronis Koutsakis. H.264 and H.265 Video Bandwidth Prediction. *IEEE Transactions on Multimedia*, 20(1):171–182, 2018. doi: 10.1109/TMM.2017.2713642.

[62] Christos Katris and Sophia Daskalaki. Comparing forecasting approaches for Internet traffic. *Expert Systems with Applications*, 42(21):8172–8183, 2015. ISSN 0957-4174. doi: 10.1016/j.eswa.2015.06.029.

[63] Tianzhu Zhang, Han Qiu, Marco Mellia, Yuanjie Li, Hewu Li, and Ke Xu. Interpreting AI for Networking: Where We Are and Where We Are Going. *IEEE Commununications Magazine*, 60(2):25–31, 2022. doi: 10.1109/MCOM.001.2100736.

[64] K. Amarasinghe, K. Kenney, and M. Manic. Toward Explainable Deep Neural Network Based Anomaly Detection. In *IEEE 11th International Conference on Human System Interaction (HSI)*, pages 311–317, 2018. doi: 10.1109/HSI.2018.8430788.

[65] Arnaud Dethise, Marco Canini, and Srikanth Kandula. Cracking Open the Black Box: What Observations Can Tell Us About Reinforcement Learning Agents. In *ACM Workshop on Network Meets AI & ML (NetAI)*, page 29–36, 2019. doi: 10.1145/3341216.3342210.

[66] Andrea Morichetta, Pedro Casas, and Marco Mellia. EXPLAIN-IT: Towards Explainable AI for Unsupervised Network Traffic Analysis. In *3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks (Big-DAMA)*, page 22–28, 2019. doi: 10.1145/3359992.3366639.

[67] Shahbaz Rezaei, Bryce Kroencke, and Xin Liu. Large-Scale Mobile App Identification Using Deep Learning. *IEEE Access*, 8:348–362, 2020. doi: 10.1109/ACCESS.2019.2962018.

[68] C. Beliard, A. Finamore, and D. Rossi. Opening the Deep Pandora Box: Explainable Traffic Classification. In *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1292–1293, 2020. doi: 10.1109/INFOCOMWKSHPS50562.2020.9162704.

[69] Xin Wang, Shuhui Chen, and Jinshu Su. Real network traffic collection and deep learning for mobile app identification. *Hindawi Wireless Communications and Mobile Computing*, 2020. doi: 10.1155/2020/4707909.

[70] Iman Akbari, Mohammad A Salahuddin, Leni Ven, Noura Limam, Raouf Boutaba, Bertrand Mathieu, Stephanie Moteau, and Stephane Tuffin. A look behind the curtain: traffic classification in an increasingly encrypted web. *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, 5(1):1–26, 2021. doi: 10.1145/3447382.

[71] Alfredo Nascita, Antonio Montieri, Giuseppe Aceto, Domenico Ciuonzo, Valerio Persico, and Antonio Pescapé. XAI meets mobile traffic classification: Understanding and improving multimodal deep learning architectures. *IEEE Transactions on Network and Service Management*, 18(4): 4225–4246, 2021. doi: 10.1109/TNSM.2021.3098157.

[72] Amir Mahdi Sadeghzadeh, Saeed Shiravi, and Rasool Jalili. Adversarial Network Traffic: Towards Evaluating the Robustness of Deep-Learning-Based Network Traffic Classification. *IEEE Transactions on Network and Service Management*, 18(2):1962–1976, 2021. doi: 10.1109/TNSM.2021.3052888.

[73] Kevin Fauvel, Alessandro Finamore, Lixuan Yang, Fuxing Chen, and Dario Rossi. A Lightweight, Efficient and Explainable-by-Design Convolutional Neural Network for Internet Traffic Classification. In *ACM 29th SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023.

[74] Chao Wang, Alessandro Finamore, Lixuan Yang, Kevin Fauvel, and Dario Rossi. Appclassnet: A commercial-grade dataset for application identification research. *SIGCOMM Comput. Commun. Rev.*, 52(3):19–27, 2022. doi: 10.1145/3561954.3561958.

[75] Alberto Dainotti, Ralph Holz, Mirja Kühlewind, Andra Lutu, Joel Sommers, and Brian Trammell. Open collaborative hyperpapers: A call to action. *SIGCOMM Comput. Commun. Rev.*, 49(1):31–33, 2019. doi: 10.1145/3314212.3314218.

[76] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, Valerio Persico, and Antonio Pescapé. MIRAGE: Mobile-app traffic capture and ground-truth creation. In *4th International Conference on Computing, Communications and Security (ICCCS)*, pages 1–8, 2019. doi: 10.1109/CCCS.2019.8888137.

[77] Ieee standard for information technology– local and metropolitan area networks– specific requirements– part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications: Further higher data rate extension in the 2.4 ghz band. *IEEE Std 802.11g-2003 (Amendment to IEEE Std 802.11, 1999 Edn. (Reaff 2003) as amended by IEEE Stds 802.11a-1999, 802.11b-1999, 802.11b-1999/Cor 1-2001, and 802.11d-2001)*, pages 1–104, 2003. doi: 10.1109/IEEESTD.2003.94282.

[78] Ieee standard for information technology– local and metropolitan area networks– specific requirements– part 11: Wireless lan medium access

control (mac)and physical layer (phy) specifications amendment 5: Enhancements for higher throughput. *IEEE Std 802.11n-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009)*, pages 1–565, 2009. doi: 10.1109/IEEESTD.2009.5307322.

[79] Ieee standard for information technology– telecommunications and information exchange between systemslocal and metropolitan area networks– specific requirements–part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications–amendment 4: Enhancements for very high throughput for operation in bands below 6 ghz. *IEEE Std 802.11ac-2013 (Amendment to IEEE Std 802.11-2012, as amended by IEEE Std 802.11ae-2012, IEEE Std 802.11aa-2012, and IEEE Std 802.11ad-2012)*, pages 1–425, 2013. doi: 10.1109/IEEESTD.2013.6687187.

[80] Ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 1: Enhancements for high-efficiency wlan. *IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020)*, pages 1–767, 2021. doi: 10.1109/IEEESTD.2021.9442429.

[81] tcpdump & libpcap. http://www.tcpdump.org/. Online; accessed Oct. 2023.

[82] netstat. https://linux.die.net/man/8/netstat/. Online; accessed Oct. 2023.

[83] Kyle MacMillan, Tarun Mangla, James Saxon, and Nick Feamster. Measuring the performance and network utilization of popular video conferencing applications. In *21st ACM Internet Measurement Conference (IMC)*, page 229–244, New York, NY, USA, 2021. ISBN 9781450391290. doi: 10.1145/3487552.3487842.

[84] Antonio Nistico, Dena Markudova, Martino Trevisan, Michela Meo, and Giovanna Carofiglio. A comparative study of RTC applications. In *2020 IEEE International Symposium on Multimedia (ISM)*, pages 1–8, Naples, Italy, December 2020. IEEE. ISBN 978-1-72818-697-9. doi: 10.1109/ISM.2020.00007. URL https://ieeexplore.ieee.org/document/9327919/. ZSCC: 0000009.

[85] Sandvine. The Global Internet Phenomena Report COVID-19 Spotlight., May 2020.

[86] Lexi Sydow - App Annie. Video Conferencing Apps Surge from Coronavirus Impact., March 2020.

[87] Fondazione CRUI. Covid-19 | strumenti per la didattica digitale. https://www.fondazionecrui.it/primo-piano/corona-virus-strumenti-per-la-didattica-digitale/, May 2020. Online; accessed Oct. 2023.

[88] European University Institute. Software available at the eui. https://www.eui.eu/ServicesAndAdmin/AcademicService/Digital-Education/Software-available-at-the-EUI, Jul 2020. Online; accessed Oct. 2023.

[89] Sandvine. The Global Internet Phenomena Report 2022., Jan 2022.

[90] Lineage os. https://lineageos.org/. Online; accessed Oct. 2023.

[91] Marc Petit-Huguenin, Gonzalo Salgueiro, Jonathan Rosenberg, Dan Wing, Rohan Mahy, and Philip Matthews. Session Traversal Utilities for NAT (STUN). RFC 8489, February 2020. URL https://www.rfc-editor.org/info/rfc8489.

[92] 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. doi: 10.1016/j.jnca.2017.11.007.

[93] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic. Who do you sync you are? smartphone fingerprinting via application behaviour. In *ACM 6th conference on Security and privacy in wireless and mobile networks (WISEC)*, pages 7–12, 2013. doi: 10.1145/2462096.2462099.

[94] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 439–454, 2016. doi: 10.1109/EuroSP.2016.40.

[95] Eline Vanrykel, Gunes Acar, Michael Herrmann, and Claudia Diaz. Leaky birds: Exploiting mobile application traffic for surveillance. In *International Conference on Financial Cryptography and Data Security*, pages 367–384. Springer, 2016. doi: 10.1007/978-3-662-54970-4_22.

[96] Benoit Claise and Brian Trammell. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011, September 2013. URL https://rfc-editor.org/rfc/rfc7011.txt.

[97] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

[98] Idio Guarino, Chao Wang, Alessandro Finamore, Antonio Pescapè, and Dario Rossi. Many or few samples?: Comparing transfer, contrastive and meta-learning in encrypted traffic classification. In *2023 7th Network Traffic Measurement and Analysis Conference (TMA)*, pages 1–10, 2023. doi: 10.23919/TMA58422.2023.10198965.

[99] Zhipeng Shen, Yuanming Zhang, Jiawei Lu, Jun Xu, and Gang Xiao. SeriesNet: A Generative Time Series Forecasting Model. In *IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2018. doi: 10.1109/IJCNN.2018.8489522.

[100] Idio Guarino, Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, Valerio Persico, and Antonio Pescapé. Classification of communication and collaboration apps via advanced deep-learning approaches. In *IEEE 26th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 1–6, 2021. doi: 10.1109/CAMAD52502.2021.9617789.

[101] Christoph Bergmeir and José M Benítez. On the use of cross-validation for time series predictor evaluation. *Information Sciences*, 191:192–213, 2012.

[102] Scott M Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. In *NeurIPS'17*, pages 4765–4774, 2017.

[103] Meelis Kull, Miquel Perello Nieto, Markus Kängsepp, Telmo Silva Filho, Hao Song, and Peter Flach. Beyond temperature scaling: Obtaining well-calibrated multiclass probabilities with Dirichlet calibration. In *Advances in neural information processing systems*, volume 32, pages 1–11, 2019.

[104] Jishnu Mukhoti, Viveka Kulharia, Amartya Sanyal, Stuart Golodetz, Philip HS Torr, and Puneet K Dokania. Calibrating deep neural networks using focal loss. In *34th Conference on Neural Information Processing Systems (NeurIPS)*, pages 15288–15299, 2020.

[105] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *34th International Conference on Machine Learning (ICML)*, page 1321–1330, 2017.

Author's publications

The research conducted in this Thesis led to the publication of several papers throughout my three-year Ph.D. program. It is worth noting that this list does not cover all of my publications but focuses solely on papers related to the primary subject of this Thesis. For each publication, I managed all stages of the research and actively participated in the writing of the manuscript.

International Journal Publications

1. **Idio Guarino**, Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, Valerio Persico and Antonio Pescapè, *Contextual counters and multimodal Deep Learning for activity-level traffic classification of mobile communication apps during COVID-19 pandemic*, Elsevier Computer Networks, Special issue on Machine Learning empowered Computer Networks 219, 2022

2. **Idio Guarino**, Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, Valerio Persico and Antonio Pescapè, *Explainable Deep-Learning Approaches for Packet-level Traffic Prediction of Collaboration and Communication Mobile Apps*, IEEE Open Journal of the Communication Society (OJ-COMS), under review, 2023

International Conference and Workshop Publications

1. **Idio Guarino**, Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, Valerio Persico, Antonio Pescapè, *Characterizing and Modeling Traffic of*

Communication and Collaboration Apps Bloomed With COVID-19 Outbreak, IEEE 6th International Forum on Research and Technology for Society and Industry (RTSI), 2021

2. **Idio Guarino**, Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, Valerio Persico and Antonio Pescapè, *Classification of Communication and Collaboration Apps via Advanced Deep-Learning Approaches*, IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2021

3. **Idio Guarino**, Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, Valerio Persico and Antonio Pescapè, *Fine-Grained Traffic Prediction of Communication and Collaboration Apps via Deep-Learning: a First Look at Explainability*, IEEE International Conference on Communications (ICC), 2023