

# Know your Big Data Trade-offs when Classifying Encrypted Mobile Traffic with Deep Learning

Giuseppe Aceto<sup>1,2</sup>, Domenico Ciuonzo<sup>1</sup>, Antonio Montieri<sup>1</sup>, Valerio Persico<sup>1,2</sup>, Antonio Pescapé<sup>1,2</sup>

<sup>1</sup>University of Napoli “Federico II” (Italy) and <sup>2</sup>NM2 s.r.l. (Italy)

{giuseppe.aceto, domenico.ciuonzo, antonio.montieri, valerio.persico, pescape}@unina.it

**Abstract**—The spread of handheld devices has led to the unprecedented growth of traffic volumes traversing both local networks and the Internet, appointing mobile traffic classification as a key tool for gathering highly-valuable profiling information, other than traffic engineering and service management. However, the nature of mobile traffic severely challenges state-of-art Machine-Learning (ML) approaches, since the quickly evolving and expanding set of apps generating traffic hinders ML-based approaches, that require domain-expert design. Deep Learning (DL) represents a promising solution to this issue, but results in higher completion times, in turn suggesting the application of the Big-Data (BD) paradigm. In this paper, we investigate for the first time BD-enabled classification of encrypted mobile traffic using DL from a general standpoint, (a) defining general design guidelines, (b) leveraging a public-cloud platform, and (c) resorting to a realistic experimental setup. We found that, while BD represents a transparent accelerator for some tasks, this is not the case for the training phase of DL architectures for traffic classification, requiring a specific BD-informed design. The experimental setup is built upon a three-dimensional investigation path in the BD adoption, namely: (i) completion time, (ii) deployment costs, and (iii) classification performance, highlighting relevant non-trivial trade-offs.

**Index Terms**—traffic classification; mobile apps; big data; deep learning; Android apps; iOS apps; encrypted traffic.

## I. INTRODUCTION

Traffic Classification (TC) consists in inferring the application (or service) generating the observed network traffic. Currently, TC is both fueled and challenged by the huge and increasing amount of mobile traffic generated by the widespread use of handheld devices (mobile data volume has grown by  $\approx 88\%$  only between 2017 and 2018 [1]). Hence, the interest in classifying mobile traffic is raising nowadays, for the purpose of e.g., differentiated billing, personalized advertising, cyber-crime detection and prevention, while extracting valuable profiling information in the process [2].

Over time, the popular adoption of dynamic ports and encrypted protocols (clustered to a few well-known ports) [3], has increasingly challenged accurate TC, crippling traditional port-based and Deep Packet Inspection (DPI) techniques [4], still effective only in closed-world scenarios (e.g., enterprise networks) enabled by man-in-the-middle proxies [5]. In the mobile-traffic context, achieving targeted TC performance is further undermined by a successful multi-platform framework-based development and distribution model [6], implying (i) the embedding of common (third-party) network services to implement app features; (ii) the quick proliferation of (similar)

apps to discriminate from; (iii) a fast-paced update cycle of apps, development frameworks, and operating systems.

For TC all these characteristics impair app-fingerprints collection, definition, and update, also possibly reducing the number of training samples available per app, due to limited time between updates. While classifiers based on Machine Learning (ML) have been proposed to cope with the shortcomings of port-based and DPI techniques, they resulted unable to keep the pace of mobile network traffic evolution [7, 8]. The main reason is that standard ML classifiers are underpinned by the design of handcrafted (i.e. domain-expert driven) features, which in TC context usually correspond to statistics extracted from the sequence of packets [7] or exchanged messages [9]. Unfortunately, such process is time-consuming, unsuited to automation, and thus unsuccessful in practical mobile TC.

Recently, a cutting-edge subset of ML techniques, known as Deep Learning (DL) [10], has emerged as the springboard toward the fulfillment of high performance in the dynamic and challenging (encrypted) TC context, allowing to train classifiers directly from input data by automatically distilling structured and complex feature representations [10]. Accordingly, several works recently appeared tackling TC via DL [11–14], but such approach resulted thorny, and generally less well understood than standard ML [15]. Indeed, DL algorithms may generate learning networks with a very dense and complex structure [10], whose training may result in completion times orders-of-magnitude higher than those acceptable according to the constraints of the specific application domain.

The constant repetition of tasks requiring high computational power and strict time constraints is the target of Big-Data (BD) frameworks. Hence, leveraging BD parallelization potential is sought to be a solution to DL-based TC. However, although BD framework embodies a transparent accelerator to *separable* computation tasks (e.g. the test phase of inference systems), this is not the case for non-naturally-parallelizable ones, like the optimization in DL training procedure [16].

This motivated our research, in which *for the first time in literature we investigate and experimentally evaluate the adoption of DL-based network traffic classification strategies as supported by BD frameworks*. In more details, pursuing our analysis along three different (but inter-playing) dimensions—i.e. classification performance, training completion time, and costs—we designed, deployed, and evaluated state-of-art DL networks (1D-CNN and LSTM) for classifying encrypted mobile traffic via BD. In our experimental campaigns, we

ran classification tasks adopting the BD platform of a public-cloud service provider and leveraging human-generated *mobile* and *encrypted* traffic datasets. This provides results related to popular and reproducible setups as well as to real-world traffic. Accordingly, our work is able to deliver a picture detailed at a depth never achieved before, producing interesting outcomes and useful guidelines for both researchers and practitioners willing to harvest the benefits deriving from the *joint* adoption of DL and BD in network traffic analysis.

The rest of the paper is organized as follows. Sec. II briefly reviews the existing literature on ML/DL-based TC and Big Data network analytics; Sec. III describes the reference Big Data-enabled DL framework for mobile TC, focusing on key aspects pertaining to the design phase; Sec. IV describes the experimental evaluation setup considered, with corresponding results discussed in Sec. V; Sec. VI ends our work with conclusions and future avenues of research.

## II. RELATED WORK

In this section we position our contribution against both (a) the existing proposals for mobile TC classification based on ML/DL and (b) the available BD-based solutions to address networking issues.

Various works have tackled **mobile TC** in recent years, mostly via standard ML techniques and often under encrypted-traffic assumption [7, 8]. Also, a number of proposals have lately emerged proving the appeal of DL to Internet TC. However, for the latter only initial design attempts are provided, all related to either *non-mobile* [12–14] or *non-encrypted scenarios* [11] (except for our previous work [15]).

In line with the interest of the scientific community, many works have employed **BD solutions** in the broad field of networking to capitalize the value of network data, notwithstanding the constraints they impose. These works mostly fall in the area of either network security [17–19], or mobile and social networks analytics [20, 21], and (almost) all benefit from distributed computations aimed at reducing the time required for training ML models. Instead, only a few works specifically leveraged BD solutions to focus on *network TC* via ML [22, 23] (with only [23] tackling the mobile case).

Recently, a few frameworks have bloomed for leveraging BD infrastructures to train (and run) DL algorithms in different flavours. However, only a very limited set of works has already adopted BD for addressing networking issues through DL algorithms [21, 24]. Alsheikh et al. [21] focused on an activity recognition based on mobile-device data and evaluated the proposed setup in terms of both speedup efficiency and accuracy. Differently, Abeshu and Chilamkurti [24] envisaged DL adoption in fog-to-things communication scenario for attack detection. Nonetheless, all these works mainly focused on how BD frameworks are able to reduce the completion time of the DL heavy tasks and—to the best of our knowledge—none of them evaluated the detrimental effect of distributing data-analysis tasks across several (loosely coordinated) workers.

To the best of authors’ knowledge, (i) no work has performed TC by means of BD-enabled DL classifiers to date.

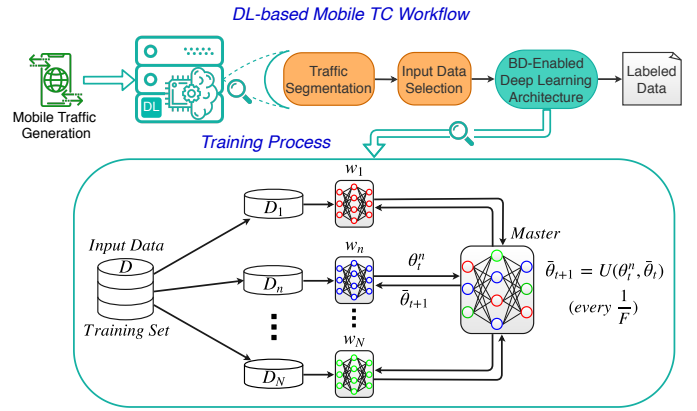


Fig. 1: Scheme for the proposed BD-enabled DL mobile TC solution.

Equally important, (ii) the challenging scenario of encrypted mobile traffic data has been only touched tangentially within the BD framework, even considering (classic) ML techniques. Finally, (iii) the validation leveraging *human-generated traffic*—that is of paramount importance toward real-world implementations in mobile contexts—has been often overlooked.

## III. DESIGN OF BD-ENABLED DL-BASED MOBILE TC

A basic scheme for the proposed BD-enabled DL mobile classification solution is reported in Fig. 1. Its related design choices can be categorized in those strictly concerning the TC workflow (that are BD-independent) and those related to the training mechanisms enforced by the DL architectures when deployed on a BD framework (that are BD-dependent, by definition). The former are refreshed in Sec. III-A, while the latter are discussed in Sec. III-B.

### A. DL-based Mobile TC Workflow

In order to design a DL system for TC, milestone design choices should be made about: (i) the *traffic object*, i.e. the traffic aggregate atom which induces the segmentation criterion; (ii) the *type(s) of input data*, i.e. the number and the sets of input selected from each traffic object to feed the DL architecture; (iii) the *DL architecture* (e.g. the composition instance of elementary learning layers) coping with input constraints originating from the design choices concerning the *type of input data*. We briefly discuss these aspects in the following, pointing to [15] for a more detailed analysis.

The **traffic object** defines how raw traffic is segmented into multiple discrete units. Most related works considered either *flows* or *biflows* [4], with the latter choice leading to better performance. In detail, a flow is a stream of packets sharing the same 5-tuple (i.e. source IP and port, destination IP and port, and transport-level protocol) taking into account their directions. Differently, in a biflow the source and destination (IP, port) pairs can be swapped. Another appealing choice is the so-called *Service Burst* (SB), recently proposed in mobile TC [7], and defined by aggregating packets with an inter-packet time smaller than a given “burst” threshold and then grouping those that belong to biflows with the same transport

protocol and destination (IP, port). Still, SBs have not seen their direct application to security and policy enforcement so far, as opposed to established (bi-)flows.

The next step after segmentation is to extract for each TC object the corresponding unbiased **input set(s)**, especially those suited for “early” TC [4] (i.e. using only the first portion of traffic aggregate to take a decision). The most relevant types of input data [15] of a generic TC object ingested by DL architectures may be roughly grouped within two categories: (i) the first  $N_b$  bytes of the payload at transport level or higher [11, 12]; (ii) selected informative data fields of the first  $N_p$  packets [14]. In the first case, the payload data being fed to the DL architecture is represented in binary format, arranged in a byte-wise fashion and normalized so as to constrain it within  $[0, 1]$ . In the second case, the type of input data is represented by selected protocol fields (not pertaining to the explicit inspection of encrypted payload, e.g. the packet size) of the first  $N_p$  packets.<sup>1</sup>

Finally, the **DL architectures** are topped with a softmax layer providing inference among  $L$  possible apps, and are obtained by composition of elementary layers [10], whose common choices are *dense*, *convolutional*, *pooling*, and *recurrent layers*. Dense layers are the simplest atoms of DL architectures, consisting of a linear transformation and an entry-wise activation. Convolutional and pooling layers are the basic blocks of Convolutional Neural Networks (CNNs), made of a set of translation-invariant filters (to extract features of a certain region) and down-sampling intermediate representations (to reduce complexity and mitigate overfitting), respectively. Recurrent layers present “loopy” unit connections and have in Long Short-Term Memory (LSTM) and gated recurrent unit their most popular variants: they are in charge of “recalling” values (via a state vector) over time and accept as input a vector sequence, whereas they output either the final state or its entire time-evolution. An exhaustive evaluation of the DL architectures is out of the scope of this work. We refer to [15] for their selection, the choice of the parameters, and an in-depth discussion of the related aspects.

### B. Training DL-based Mobile TC architectures on Big Data

The learning process for DL architectures may be slow and computationally demanding, since they consist of many hidden layers, millions of parameters and require a high number of training samples. BD solutions are meant to offer a way to address these issues, providing processing frameworks able to parallelize computation tasks by splitting the information base and distributing it across  $N$  **cooperating working nodes** (*workers*) coordinated by a single central node (*master*).

Specifically, BD-enabled DL relies on *data parallelism* and *federated learning* [16] to reduce the overall training time of the considered DL architecture, by capitalizing the peculiarity of BD paradigm. In essence, the workers  $w_1, \dots, w_N$  are given  $N$  distinct partitions  $\mathcal{D}_1, \dots, \mathcal{D}_N$  of the training set  $\mathcal{D}$  to

<sup>1</sup>We remark that instances longer (resp. shorter) than the considered fixed-length ( $N_b$  or  $N_p$ ) data inputs are truncated (resp. zero-padded) to the designed length of bytes ( $N_b$ ) or packets ( $N_p$ ).

learn independent replicas of the (same) given DL architecture. Clearly, deploying a higher number of workers allows to enhance the parallelization (the higher  $N$ , the smaller the size of the partitions  $\mathcal{D}_1, \dots, \mathcal{D}_N$  assigned to the workers). On the other hand, each worker is able to learn only a “data-partial” DL model, being the outcome of its limited-view training partition, in principle. Additionally, since learning is based on (sophisticated versions of) stochastic gradient descent, the process of the  $n^{th}$  worker is naturally iterative and performed over  $N_{\text{epo}}$  “epochs”, composed of different mini-batches (scanning the whole  $\mathcal{D}_n$ ), with the model at time  $t$  completely specified by the parameter set  $\theta_t^n$ .

In federated learning different workers are federated by the master to optimize a central DL model (specified at time  $t$  by the parameter set  $\bar{\theta}_t$ ) exploiting their DL model replicas by minimizing a single (common) loss function  $\mathcal{L}(\cdot)$  (for TC a categorical cross-entropy [10]) and implicitly capitalizing the whole training set  $\mathcal{D}$ . This is achieved by *periodically synchronizing* the state of each worker with the (centralized) view of the master, whose model is incrementally updated leveraging the information provided by the workers. The master is in charge of the coordination mechanism and has the responsibility to incorporate model updates periodically coming from the workers (*worker commits*), and to serve requests of the most updated central model (*worker pulls*). Between subsequent commits each worker learns *independently* on its training partition. The **worker update frequency**  $F$  at which the workers execute a commit is thus a design parameter. Such frequency ranges from one update per mini-batch to exchanges after several epochs, the higher (resp. lower) values leading to tighter (resp. looser) coupling.

Additionally, depending on the **communication protocol** governing the exchange of commits/pulls between the workers and master, BD-enabled DL approaches can be categorized into two main groups: *synchronous* and *asynchronous*. In the former case, commits from the workers are aligned through a synchronization barrier, and the pull operation puts all the nodes in the same state  $\bar{\theta}_{t+1}$  after the master aggregation. In the latter case, commits from the workers are handled in a first-come first-served fashion by the master, which provides the updated central model  $\bar{\theta}_{t+1}$  based on the message from the worker. Although the latter solution can incur the side-effect that some workers are computing (and committing) updates based on old central-model states (because the master incorporates updates into the central model asynchronously), it is more time-efficient because it does not include locking mechanisms (that make all workers wait for the slowest one: the so-called *straggler* issue) and works well also with heterogeneous hardware.

Lastly, the **federated-optimization algorithm** is another degree of freedom of the BD DL-based TC system proposed. It is defined by both local workers computation and master update policy and is tightly coupled to the communication protocol choice. Precisely, for each update of the central model, in the synchronous (resp. asynchronous) case the master uses all the commits at once (resp. one commit at a time).

Accordingly, the adoption of BD framework to support the learning process of DL architectures is expected to greatly reduce the time required for its training on the *whole*  $\mathcal{D}$ . However this benefit comes at a cost: since no node has the chance of working on the whole dataset, the DL architecture resulting from this training procedure represents a *sub-optimal solution* to the TC problem, exposing performance possibly worse than that of a centralized solution (with much longer processing periods but working on the  $\mathcal{D}$  training set as a whole). Hence, next section investigates the dependence of DL training in mobile TC on the non-transparent BD accelerator.

#### IV. EVALUATION SETUP

In this section we detail the setup designed and adopted for the experimental evaluation. First, in Sec. IV-A we describe the mobile TC problems addressed and the corresponding datasets leveraged for the evaluation. Then, in Sec. IV-B we specify the BD-enabled DL TC architecture deployed and the tools we adopted. Finally, in Sec. IV-C we introduce the performance metrics to investigate the proposed TC system along different dimensions induced by BD solutions.

##### A. Classification problems and description of the datasets

Our evaluation resorts to two datasets, either recommended or produced by a global mobile solution provider<sup>2</sup>, associated to different mobile and encrypted TC tasks (a summary is given in Tab. I), to understand if and how the performance of the different mobile TC problems are impacted by the BD infrastructure. These datasets contain traffic from apps running on both *Android* and *iOS* devices (covering the two most popular mobile OSes), and have been collected by *human users* instead of relying on bot-generated traffic, as opposed to recent works on mobile TC [7]. For the sake of a consistent assessment of almost all DL-based TC works published so far [12, 14, 15], we have chosen to operate at the *biflow* level. Finally, the ground truth has been obtained by labeling each traffic trace with the generating app (running each app separately limited the presence of background traffic).

The first (*binary*,  $L = 2$ ) dataset (*FB/FBM*) was collected in the ARCLAB laboratory at the University of Naples “Federico II”. In detail, the capture sessions were run on a Xiaomi Mi5 and refer to either Facebook (FB) or Facebook Messenger (FBM) traffic data. This dataset allows to evaluate the capability of the classifiers to discriminate between two apps with extremely similar fingerprints, for e.g., *billing differentiation*. *More than 100 users* were requested to perform various activities for both the apps, including login/registration/logged-use cases (to explore diversity). Overall, the dataset contains  $> 34k$  instances (see Tab. I), with 19.3k (resp. 15.0k) biflows generated by FB (resp. FBM), with a 56% (resp. 44%) share, guaranteeing also a good balance between FB/FBM samples.

The second (*multi-class*,  $L > 2$ ) dataset (*iOS*), contains traffic generated by 45 iOS apps. This dataset was directly

handled by the solution provider and is here explored for evaluating TC for e.g. *prioritization purposes*.

We mention that the traces were generated by users with different devices and OS/app versions, and were provided already anonymized.<sup>3</sup> Differently from FB/FBM dataset, in this case we have  $1 \div 48$  traces per app, leading to a non-negligible class imbalance.

##### B. Architecture deployment

Herein we detail the experimental setup designed and implemented to evaluate the performance of the DL-based TC solutions when deployed onto BD architectures.

In line with the strategies usually adopted today by enterprises aiming at achieving both technical and economical advantages, we run all our experimental-evaluation campaigns onto a cloud platform. In detail, we utilized the services of *Microsoft Azure*, one of the market leaders among the cloud providers. The impact of this decision on our analysis is two-fold: (i) some of the following deployment choices depend upon the options commercialized by the provider; (ii) the adoption of a public-cloud platform puts under the spotlight the *economical expenditure* generated by the execution of DL tasks. Though this choice may place constraints on the experimental analysis because of the finite budget available, it allows us to further enrich our study with interesting results along dimensions other than classification performance, such as the *cost* charged to cloud customers for accomplishing model training tasks (see Sec. IV-C).

All the results discussed in Sec. V have been obtained leveraging *Distributed Keras* [16], a distributed DL framework built on top of Apache Spark<sup>4</sup> and Keras<sup>5</sup>. In details, we relied on *Azure Databricks*<sup>6</sup>, which provides analytic services based on an Apache Spark environment optimized for DL. Distributed Keras provides several state-of-art optimization algorithms (based on data-parallelism and federated learning) and is claimed to reduce the time spent for training models with respect to traditional centralized approaches.

Specifically, the inputs for the experimental setup (number of workers  $N$ , worker update frequency  $F$ ) were selected according to budget constraints as well as observed trends, so as to explore satisfactorily the space generated by all their combinations. In detail, we consider deployments with the number of workers ranging from  $N = 2$  to  $N = 16$ , while for  $F$  we have considered values from one update per mini-batch (i.e.  $\approx 139$  updates per epoch in our experimentation) to one update every  $N_{\text{epo}}$  epochs (i.e. one single update per worker).

Furthermore, the setup of master and worker nodes was chosen according to the offers of the cloud provider, by adhering to the default setting which employs the same node configuration for both the master and the workers. In detail, general-purpose DS4v2 nodes (8 vCPUs, 28 GiB RAM, 0.698 €/hour) are used in all our experiments, with better-performing

<sup>3</sup>In detail,  $\approx 85\%$  of iOS traces were captured during 2016.



<sup>4</sup><https://spark.apache.org/>.

<sup>5</sup><https://keras.io/>.

<sup>6</sup><https://azure.microsoft.com/it-it/services/databricks/>.

<sup>2</sup>Due to NDA with the provider we can not report its name, details of its network, detailed information on the datasets, nor release the datasets.

TABLE I: Datasets for the evaluation of BD-enabled DL architectures. Avg. trace duration is  $\approx 5$  mins. **ET** stands for Encrypted Traffic.

Dataset	Type	#Apps	#Traces	#Biflows	ET	OS Version	Collection	Source	Main Aim
	Binary	$L = 2$	> 1100	34.3k	91%	Android 6.0.1	May '17–Mar. '18	Self-generated@UniNa	Billing differentiation
	Multi-class	$L = 45$	419	44.0k	60%	7.0–10.0	Sept. '14–Jan. '17	Mobile solutions provider	Service prioritization

D32sv3 nodes (32 vCPUs, 128 GiB RAM, 2.456 €/hour) leveraged for specific analyses, as detailed later.

Finally, the DL architectures selected are those with the best performance (for each input type, see Sec. III-A) in a centralized deployment [15]: a 1D-CNN [12] (fed with the first  $N_b = 784$  payload bytes of the transport level) and an LSTM [14] (fed with four informative fields<sup>7</sup>, of first  $N_p = 20$  packets in a biflow). These correspond to 5.82M and 52.3k (resp. 5.86M and 56.6k) *training parameters* for FB/FBM (resp. iOS) dataset, respectively. Concerning the optimization algorithm, we adopted the AEASGD (with  $N_{\text{epo}} = 90$ ) [16], being asynchronous and thus able to avoid the straggler issue.

### C. Evaluation Metrics

Here we introduce the metrics adopted to evaluate the DL architectures when deployed on (cloud) BD frameworks. In detail, our experimental analysis resorts to a stable performance-evaluation setup, based on a stratified ten-fold cross-validation. Hence, for each of the metrics discussed in what follows, we report its mean and the standard deviation. Notably, our experimental evaluation is performed along three distinct dimensions: (i) *training completion time*, (ii) *cloud deployment cost*, and (iii) *classification effectiveness*. We are interested in investigating the *trade-offs* existing among these three intertwined dimensions. The metrics defined and adopted for each dimension are detailed in the following.

**Training Completion Time.** Since reducing the processing time required for a task completion is arguably the major driver leading to the adoption of BD architectures, we provide a detailed evaluation of this key aspect, focusing on the *wallclock time*  $T$  required for completing the *training phase* of DL architectures.<sup>8</sup> This analysis is of great interest since mobile TC systems require frequent re-training operations, due to aging of training data as a result of both app and OS updates [7, 15]. Precisely, since (distributed) DL training is performed on multiple epochs [10], we report such information in a normalized way, as Wallclock Time Per-Epoch (WTPE).

**Cloud Deployment Cost.** Cloud services are characterized by pay-as-you-go billing strategies, thus abolishing capital expenditure for configuring and maintain the BD infrastructure. Accordingly, here we consider the total cost  $C$  charged to the cloud customers for running the processing tasks needed for training the DL architecture. Specifically, our *cost evaluation function* is  $C = (\rho N + \rho_M) T$ , where  $N$  denotes the number of workers,  $\rho$  (resp.  $\rho_M$ ) the hourly cost for deploying one worker node (resp. the master), and  $T$  the Training Completion Time.

<sup>7</sup>Packet size, packet direction, TCP window size, inter-arrival time.

<sup>8</sup>We recall that time reduction trends of testing phase are less interesting, due to perfect parallelization.

**Classification effectiveness.** Because BD frameworks do not represent a transparent accelerator for the training phase of DL-based traffic classifiers, to evaluate the effectiveness of the corresponding DL-based TC solutions, the adopted evaluation metrics include common classification measures [4] such as the (macro) *recall* (i.e. the arithmetic average of per-app accuracies) and *F-measure* (i.e. the harmonic mean of per-app precision and recall, arithmetically averaged over all the considered apps). Finally, we also consider *confusion matrices* to identify the most frequent misclassification patterns.

## V. RESULTS AND DISCUSSION

Herein we discuss the results of the experimental campaigns we run deploying the designed system on Azure PaaS to evaluate its performance against two mobile TC tasks (binary and multi-class, see Sec. IV-A), along the three evaluation dimensions (completion time, cost, and classification effectiveness, see Sec. IV-C). For each of these, we assess the impact of different design choices such as the number of workers ( $N$ ), the update frequency ( $F$ ), and the DL architecture.

**Completion Time vs. Number of Workers ( $N$ ).** Figs. 2a and 2d show the WTPE for the two considered DL-based TC architectures on FB/FBM and iOS datasets, respectively, when increasing  $N$  from 2 to 16. Herein, the worker update frequency  $F$  is set to one update per epoch. To stress the overhead incurred by each BD-enabled DL architecture, we consider the corresponding WTPE  $T_1$  needed to run it in a centralized fashion, i.e. when *one worker* is in charge of processing the whole training set. Accordingly, we report the *ideal-WTPE curve*, defined as  $T_1/N$  and corresponding to a *lower-bound* on the achievable WTPE.

The results show an intuitive decreasing trend with  $N$  for both TC tasks (with slightly higher WTPE for iOS, in line with the more complex classification task), thus confirming the appeal of the BD framework which is able to reduce the training time up to  $-91.8\%$  (resp.  $-88.5\%$ ) when a 1D-CNN is used in the case of FB/FBM (resp. iOS), with respect to an analogous centralized deployment. For example, with  $N = 8$  workers,  $\leq 10s$  WTPE is required in both TC tasks. Additionally, the overhead incurred with respect to the theoretical curve also increases for higher values of  $N$  (i.e. the larger  $N$ , the higher the overhead), but remains negligible. Finally, a direct comparison of the two different DL architectures shows that the more complex 1D-CNN benefits more from parallelization with respect to the “lighter” LSTM. **Cost vs. Number of Workers ( $N$ ).** Figs. 2b and 2e show the impact of the number of workers ( $N \in \{2, 4, 8, 16\}$ ) on the training cost of the two considered DL architectures in line with the pay-as-you-go billing model enforced, when addressing mobile TC tasks of FB/FBM and iOS datasets,

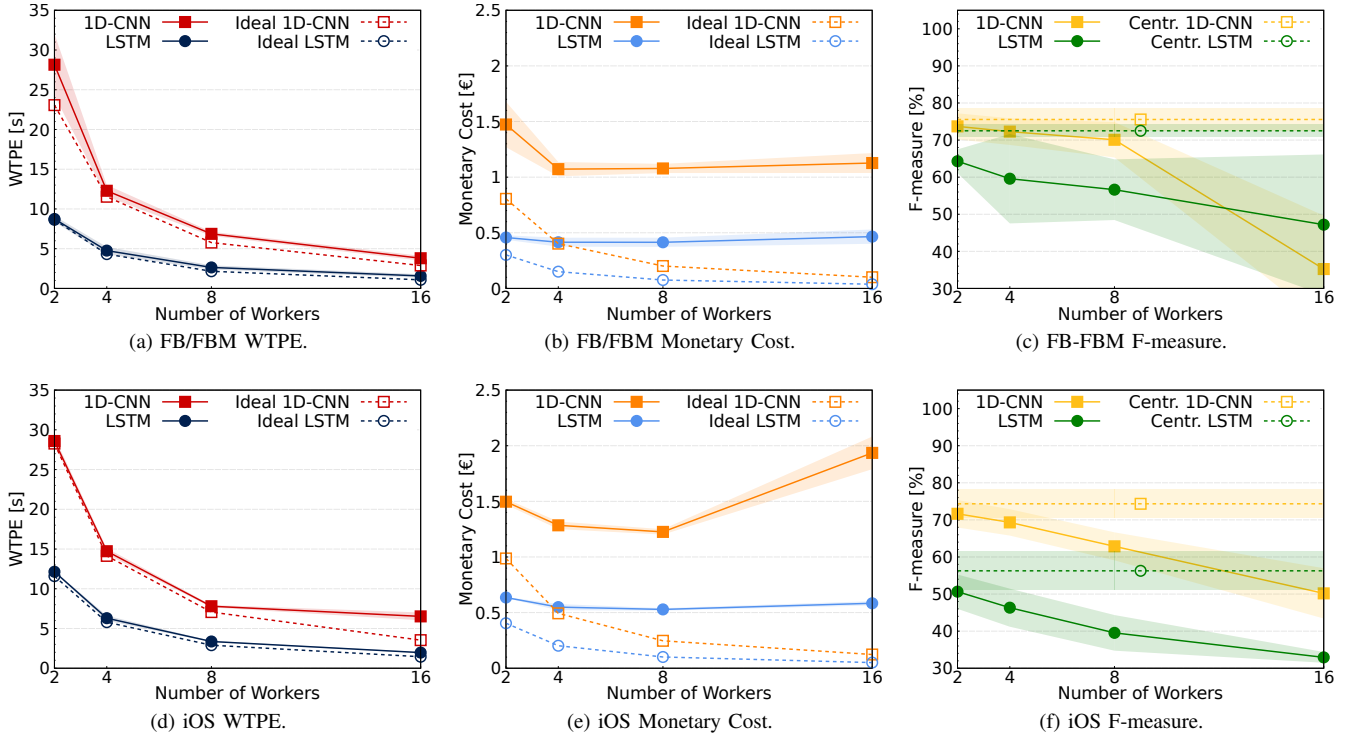


Fig. 2: Impact of no. of workers on WTPE, Monetary cost, and F-measure for FB/FBM dataset (a, b, and c) and for iOS dataset (d, e, and f). Both 1D-CNN and LSTM architectures are considered. Average on 10-folds with  $\pm 3\sigma$  confidence bands are shown.

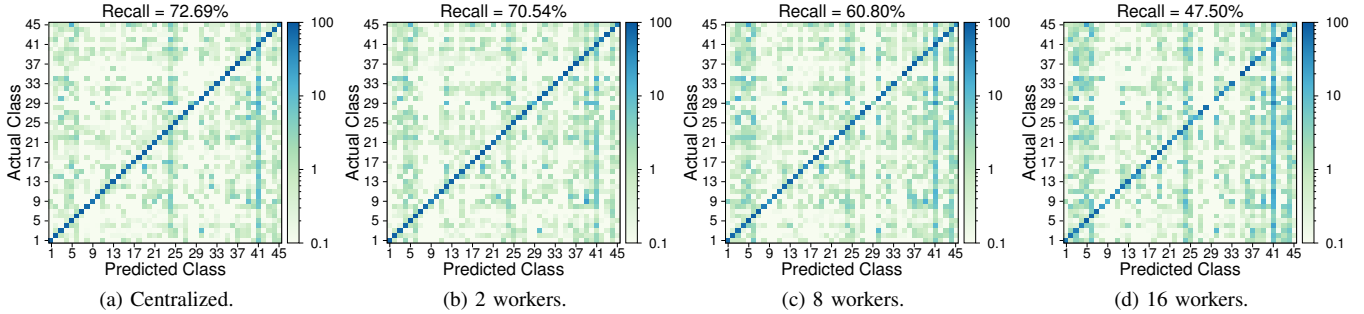


Fig. 3: Confusion matrices of 1D-CNN on iOS dataset for centralized case (a) and BD-solution with  $N \in \{2, 4, 16\}$  workers (b-c-d) ([%] in log scale).

respectively ( $F$  is again set to one update for epoch). To stress the overhead cost incurred by BD-enabled DL architectures, we also report (for each architecture) the *ideal-cost curve* corresponding to  $(\rho N + \rho_M) \cdot (T_1/N)$ , i.e. the cost required to train the DL architecture in the ideal case the BD framework guarantees perfect parallelization, being a *lower-bound* on the achievable cost—with  $\rho_M = \rho$  in our case (see Sec. IV-B). While the *hourly cost* for cloud system deployment linearly increases with  $N$  (namely  $\rho(N + 1)$ ), the resulting total cost  $C$  for completing the training phase is also proportional to the required time  $T$ . As the training time may deviate from its ideal value as shown in the previous analysis for higher values of  $N$  (e.g. only negligible benefits are achieved moving  $N$  from 8 to 16 when using 1D-CNN for iOS), similarly, the resulting monetary cost may increase as the decreased training

time does not always match a balanced gain in terms of hourly cost. Accordingly, while the deployment cost for LSTM (for both mobile TC tasks) and 1D-CNN (for FB/FBM) almost saturates for larger values of  $N$ , this is not the case for 1D-CNN for iOS. In the latter case (see Fig. 2e), deploying a larger number of workers ( $N = 16$ ) leads to significantly higher costs (+54.2% with respect to the case  $N = 8$ ) while the benefit in terms of reduced training time is negligible ( $-2.3\%$ ).

**Classification Effectiveness vs. Number of Workers ( $N$ ).** Figs. 2c and 2f report the effectiveness of the two DL architectures accomplishing binary and multi-class mobile TC tasks respectively, when deployed on clusters where  $N$  ranges from 2 to 16. Experimental results witness (solid lines) how the degree of parallelization hinders the classification performance achieved, with F-measure values significantly decreasing as  $N$

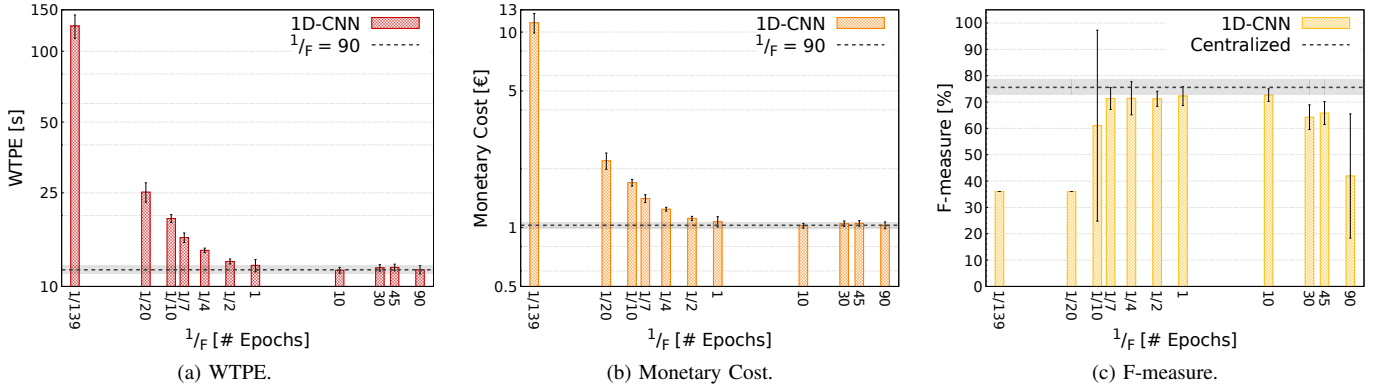


Fig. 4: Impact of no. of epochs (reciprocal of worker update frequency) on WTPE (a), Monetary cost (b), and F-measure (c) for FB/FBM dataset and 1D-CNN (L7-784) architecture. Average on 10-folds with  $\pm 3\sigma$  confidence intervals are shown.

increases. Accordingly, the worst classification performance is observed when relying on a 16-node cluster, namely  $-53.4\%$  (resp.  $-41.5\%$ ) compared to a *centralized solution* when addressing binary (resp. multi-class) classification via 1D-CNN (resp. LSTM). On the other hand, classification performance obtained by 2-node deployments are closer to those attainable by centralized DL implementations (dashed lines). To deepen the above investigation, we show in Fig. 3 the confusion matrices pertaining to the (best performing) 1D-CNN on iOS dataset by investigating the error patterns for  $N \in \{2, 8, 16\}$  in comparison to the centralized case. Confusion matrices show a general degradation with growing  $N$ , with some apps *not recognized in most of the cases*, as also confirmed by the corresponding recall (e.g. 47.50% for  $N = 16$ ).

Such results witness how the adoption of DL deployments leveraging the power of BD frameworks may generate significant performance loss: though current solutions provide ready-to-use implementations with interfaces similar to (if not matching) the centralized counterparts, DL training stage is *not naturally parallelizable*, thus resulting in worse classification results due to reduced training accuracy collectively provided by workers when operating on smaller dataset portions.

**Impact of worker update frequency ( $F$ ).** In Fig. 4, we evaluate the three considered dimensions versus worker update period  $1/F$  (reported in terms of either number or fraction of epochs), with a range  $\frac{1}{F} \in [1/139, 90]$  epochs (i.e. from one update every mini-batch to one update during the whole training phase). For brevity and budget constraints, the analysis focuses on the best performing BD-enabled DL architecture (i.e. 1D-CNN) trained and tested on the binary FB/FBM dataset with  $N = 4$  workers. For both WTPE and cost analyses (Figs. 4a and 4b), we consider as the *lower-bound counterparts* the values obtained considering the loosest coupling between the workers and the master ( $\frac{1}{F} = 90$ ), while for the F-measure the *upper-bound* value of the centralized case. As expected, both WTPE and cost (Figs. 4a–4b) increase with  $F$ . Interestingly, a steep reduction is evident when passing from one update every single mini-batch to one every 7 mini-batches (i.e. from  $\frac{1}{F} = \frac{1}{139}$  to  $\frac{1}{F} = \frac{1}{20}$  epoch) with a  $-80.3\%$

decrease. On the other hand, when the update period goes from  $\frac{1}{F} = \frac{1}{20}$  to  $\frac{1}{F} = 90$  the decrease is only  $-53.2\%$ .

Finally, Fig. 4c shows the classification effectiveness in terms of F-measure. The best performance is obtained with  $\frac{1}{4} \leq \frac{1}{F} \leq 10$  with a significant degradation for  $\frac{1}{F} \leq \frac{1}{10}$  and  $\frac{1}{F} \geq 30$ . Whilst worse performance is expected when the exchange of updates is less frequent (right side of Fig. 4c), this phenomenon is unexpected in the presence of tight coupling (i.e.  $\frac{1}{F} \leq \frac{1}{10}$ , left side of Fig. 4c). To shed light on this evidence, we have performed additional experiments (not shown for brevity) with better-performing worker/master nodes (D32sv3). Results highlight that in this case the F-measure obtained with  $\frac{1}{F} = \frac{1}{7}$  and  $\frac{1}{F} = \frac{1}{10}$  is comparable with the best-performing case, thus not showing any performance decrease due to the tight coupling. Nonetheless, the same performance trend of Fig. 4c is observed for  $\frac{1}{F} \leq \frac{1}{20}$ . This result suggests that a *computational bottleneck* exists at the master, hindering the correct collection of the updates from the workers, hence resulting in a worse-performing DL model.

## VI. CONCLUSIONS AND FUTURE DIRECTIONS

We tackled TC of mobile traffic via DL architectures supported by BD solutions, providing a comprehensive methodological evaluation and comparison, pursued along three different (intertwined) dimensions, i.e. training completion time, costs, and classification performance. Specifically, we designed, deployed, and evaluated TC state-of-art DL networks for classifying encrypted mobile traffic via BD. In the experimental campaign we adopted the BD platform of a leading public-cloud service provider (Microsoft Azure) and leveraged two human-generated mobile and encrypted traffic datasets. Accordingly, our work provided an in-depth analysis never achieved so far, producing interesting outcomes and useful guidelines for harvesting the benefits deriving from the joint adoption of DL and BD, with specific focus on mobile TC.

In detail, although the adoption of the BD framework to support DL architectures significantly reduces the overall training time (with even more significant trends expected in larger datasets), especially in the case of high parallelization, its non-transparent nature has a *direct implication* on

DL classification performance. Indeed, the joint use of data parallelism and federated learning provides a final trained DL architecture representing a sub-optimal solution to the TC task, not reaching the performance of a centralized solution (that takes longer times, but works on the training set as a whole), with more marked effects in the high-parallelization case ( $N = 16$  workers in our experiments). Such performance gap significantly depends also on the worker update frequency  $F$ , and TC “centralized” performance may be approached only through higher frequency values. Sadly, this inherent tradeoff leads to higher computational overhead for the master (viz. more powerful hardware required) and impacts on both time and cost performance. This precludes a wallclock time cut proportional to the number of workers, which reflects on the cost unsuitability, highlighted by a cost-optimal number of workers. Concluding, the above outcomes highlight the dependence of BD-enabled DL-based mobile traffic classifiers, in a non-trivial way, on (a) the degree of parallelization and (b) the communication frequency of the BD architecture supporting the training phase of DL-based traffic classifiers.

The present study motivates several *research directions*: (i) deployment and validation of *advanced* BD-enabled DL-based traffic classifiers, exploiting multimodal data fusion and adopting more sophisticated DL layers (e.g. inception, residual connections, etc.); (ii) accelerated exploitation of massive unsupervised data for transfer learning, granted by BD solutions; (iii) prototyping of BD-enabled DL architectures able to exploit *both* model and data parallelism [16]; (iv) stream-based learning implementations [17] of BD-enabled DL-based traffic classifiers to account for concept drift.

#### REFERENCES

- [1] F. Jejdling et al. Ericsson Mobility Report - Q4 Update EAB-19:001022 Uen, Rev. A. Technical report, Ericsson AB, Stockholm, Sweden, Feb 2019.
- [2] D. Rajashekar, N. Zincir-Heywood, and M. Heywood. Smart phone user behaviour characterization based on autoencoders and self organizing maps. In *IEEE ICDMW'16*.
- [3] A. Razaghpanah, A. A. Niaki, N. Vallina-Rodriguez, S. Sundaresan, J. Amann, and P. Gill. Studying TLS usage in Android apps. In *13th ACM CoNEXT*, 2017.
- [4] A. Dainotti, A. Pescapè, and K. C. Claffy. Issues and future directions in traffic classification. *IEEE Network*, 26(1), 2012.
- [5] H. Yao, G. Ranjan, A. Tongaonkar, Y. Liao, and Z. M. Mao. SAMPLES: Self adaptive mining of persistent lexical snippets for classifying mobile application traffic. In *ACM MobiCom'15*.
- [6] T. Majchrzak and T.-M. Grønli. Comprehensive analysis of innovative cross-platform app development frameworks. In *HICSS'17*.
- [7] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic. Robust smartphone app identification via encrypted network traffic analysis. *IEEE Trans. Inf. Forensics Security*, 13(1), 2018.
- [8] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè. Multi-classification approaches for classifying mobile app traffic. *Elsevier JNCA*, 103, 2018.
- [9] A. Hajjar, J. Khalife, and J. Díaz-Verdejo. Network traffic application identification based on message size analysis. *Elsevier JNCA*, 58, 2015.
- [10] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.
- [11] Z. Wang. The Applications of Deep Learning on Traffic Identification., 2015.
- [12] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *IEEE ISI'17*.
- [13] M. Lotfollahi, R. Shirali, M. J. Siavoshani, and M. Saberian. Deep packet: a novel approach for encrypted traffic classification using Deep Learning. *preprint arXiv:1709.02656*, 2017.
- [14] 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, 2017.
- [15] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè. Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges. *IEEE Trans. Netw. Service Manag.*, in press, 2019.
- [16] C. I.-D. Joeri R. Hermans. Distributed keras: Distributed deep learning with apache spark and keras. <https://github.com/JoeriHermans/dist-keras/>, 2016.
- [17] P. Mulinka and P. Casas. Stream-based machine learning for network security and anomaly detection. In *ACM Big-DAMA'18*.
- [18] E. Viegas, A. Santin, A. Bessani, and N. Neves. Bigflow: Real-time and reliable anomaly-based intrusion detection for high-speed networks. *Elsevier FGCS*, 2019.
- [19] B. Zhou, J. Li, Y. Ji, and M. Guizani. Online internet traffic monitoring and DDoS attack detection using Big Data frameworks. In *IEEE IWCMC'18*.
- [20] V. Persico, A. Pescapé, A. Picariello, and G. Sperli. Benchmarking Big Data architectures for social networks data processing using public cloud platforms. *Elsevier FGCS*, 89, 2018.
- [21] M. A. Alsheikh, D. Niyato, S. Lin, H.-P. Tan, and Z. Han. Mobile Big Data analytics using Deep Learning and Apache Spark. *IEEE Network*, 30(3), 2016.
- [22] V. D'Alessandro, B. Park, L. Romano, C. Fetzer, et al. Scalable network traffic classification using distributed support vector machines. In *IEEE CLOUD'15*.
- [23] L.-V. Le, B.-S. Lin, and S. Do. Applying Big Data, Machine Learning, and SDN/NFV for 5G early-stage traffic classification and network QoS control. *Transactions on Networks and Communications*, 6(2), 2018.
- [24] A. Abeshu and N. Chilamkurti. Deep Learning: the frontier for distributed attack detection in Fog-to-Things computing. *IEEE Commun. Mag.*, 56(2), 2018.