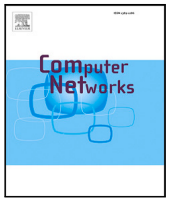
Computer Networks 220 (2023) 109471

Contents lists available at ScienceDirect

Computer Networks

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



Web browsing privacy in the deep learning era: Beyond VPNs and encryption☆

Daniel Perdices [a,*], Jorge E. López de Vergara [a,b], Iván González [a,b], Luis de Pedro [a,b]

[a] Department of Electronic and Communication Technologies, Escuela Politécnica Superior, Universidad Autónoma de Madrid, Spain
[b] Naudit High Performance Computing and Networking, S.L., Spain

ARTICLE INFO

Keywords:
Web browsing analytics
Neural network
Privacy
Deep learning
Transformer

ABSTRACT

Web browsing privacy is a matter of paramount importance for the Internet users. While they try to protect themselves from being monitored by getting advantage of encryption or VPNs, users' privacy is still unaccomplished, even taking into account the tangled web, with several domains visited at the same time in a single web page, or IP addresses of a cloud provider shared by several sites. In this work, we provide a novel approach to identify user web browsing that only takes into account the IP addresses that the user has connected to and without performing any DNS reverse resolutions. We use this sequence of addresses as an input of different state-of-the-art deep learning models, such as multi-layer perceptron and transformers, which are able to accurately identify which was the website actually visited among Alexa's World Top 500 most visited domains. Moreover, we have also studied other factors, such as the dependence on the DNS server used to resolve the visited IP addresses, the accuracy for the top domains (e.g., Google, YouTube, Facebook, etc.), data augmentation by packet sampling simulation to improve our results, the impact on packet sampling and the fine-tuning and possible impact of model parameters or the scalability of our approach. We conclude that, using only a 10% of the packets, we can identify the visited website with an accuracy and F1 score between 94% and 95%.

1. Introduction

In the last years, data have become one of the most valuable assets in the world. It can provide deep customer insights to business makers, which are really interested in knowing the interests of their clients or if they are visiting the competitors web pages. Nowadays, most of the top tech companies exploit such data or even sell it to others, making this a really profitable business, as noticed by public authorities and governments [2–4]. Thus, end users have been paying attention to this matter since they are the ones who produce data, and they have no real knowledge of how their data are being employed.

Consequently, privacy and data usage have become main concerns of the Internet users [5]. Answering questions such as which data are used by the companies, who they share it with, and how valuable it is; are major issues nowadays. Therefore, users try to protect themselves as much as possible, in particular, they limit the amount of data they share. However, this sometimes does not avoid the data being captured and used by many agents. Solutions such as encryption, both at HTTP level [6] and at DNS level [7,8], have become default standards that will cover the majority of the traffic in the next years. Nevertheless, they can only encrypt end-to-end conversations, meaning that IP and TCP or UDP information is still available.

Another popular method used to protect the privacy and avoid data usage is using Virtual Private Networks (VPNs). Although VPNs have become increasingly popular and most of them may encrypt and tunnel IP traffic, in fact, traffic can still be monitored at the termination point of the VPN. Then, it is just a matter of who you are giving your data to. This means that actors between the VPN server network and the website server can see and use the data. The VPN provider can even go beyond that, since it also knows the identity of the client.

Other alternatives, such as Tor [9], Brave browser [10] or using chains of proxies suffer from a similar issue, where your identity might not be clear, but your navigation data can also be used by the last server in the chain. For many purposes, such as marketing and trends studies, aggregated data are still valuable, so it does not matter whether the identity of the user is known. In fact, we checked that popular Tor-capable browsers such as Tor or Brave use the same endpoint within

☆ This paper is an extended version of our work presented in [1].

* Corresponding author.
E-mail addresses: daniel.perdices@uam.es (D. Perdices), jorge.lopez_vergara@uam.es (J.E. López de Vergara), ivan.gonzalez@uam.es (I. González), luis.depedro@uam.es (L. de Pedro).

https://doi.org/10.1016/j.comnet.2022.109471
Received 25 March 2022; Received in revised form 10 November 2022; Accepted 14 November 2022
Available online 21 November 2022

the web browsing activity of a tab, meaning that these Tor exit nodes can still use, give, and monetize your data. This has proven to be more than a possibility, given that a significant percentage of Tor exit nodes have spied on their users' activity [11] and even have used `sslstrip` for HTTP traffic to cryptocurrency exchange websites [12].

In this light, we want to answer the following main research question: Can someone identify where you are navigating through in these cases? As we pointed out, this could be achieved as long as web browsing activity can be inferred from IP traffic. This means that this question is equivalent to: Can it be inferred what website you are visiting using only IP layer information? This is the research question we want to answer, since IP addresses can be easily obtained using NAT logs, Netflow [13] or IPFIX [14] records, or custom monitoring tools.

Given an affirmative answer to the main matter, we are also interested in subsequent questions regarding the feasibility of a model in real-life conditions. These are: does the model depend heavily on external factors such as the location or the DNS resolution? Can the model scale to real situations with thousands of customers and thousands of websites to be identified? Are there ways of preventing the identification of the website?

From an operational point of view, ISP can see our results from two rather different points of view. First, they could use our work to identify customers' traffic. This allows ISP to monetize data useful for many marketing purposes, after proper anonymization and customers' consent. Second, ISP must take care of who has access to network traffic data. Giving it to third parties could lead to potential data breaches on clients' privacy.

Although the main task is supposedly simple, there are many complications that can arise. The main issue is what other authors have defined as the tangled web [15]. When connecting to a website, the web browser has to open a cascade of connections to other websites due to images, ads, banners, JavaScript libraries, social media links, and many more. It is not only content from third-parties that developers include in their websites, but also mechanisms of the web browser such as prefetching, ad blockers, or caching that may cause trouble.

This entails that discerning the web browsing behavior from these connections can be quite problematic and unstable, since some connections to Facebook servers or Google servers might appear in other different websites that have nothing to do with them. It is even possible that some connections are opened to prefetch other websites or, on the other hand, they are not opened due to the cache or an ad blocker. Moreover, the websites deployed on content delivery networks and cloud providers make this matter worse, as a single IP address can be shared by several domains [16], which causes DNS reverse queries to be useless. Furthermore, in our datasets, we observed that, for each pair of web browsing activities, an average of 25% of the IP addresses receiving HTTP/HTTPS connections were present in both traces. Additionally, the mean of the percentage of IP addresses of one domain that overlap with the IP addresses of other domains is around 80%.

In this paper, we present two deep learning models: one simple straight-forward approach, and an adapted version of the transformer [17], that predict the domain visited using just information of the observed IP addresses. For the sake of the evaluation, we have built a dataset of more than 340 GB of network traffic of automatic web browsing activities through the top 500 domains of Alexa [18]. This model achieves an accuracy higher than 94% on this dataset and proves that the answer to the previous question is affirmative. Also, it allows us to see how many actors can really know about your web browsing behavior and, thus, use your data for any purpose.

The rest of the document follows this outline: Section 2 provides a summary of the current state of the art, focusing of the novelty of this work. Next, Section 3 explains the development of the deep learning models. Section 4 benchmarks the performance of the models with several facts in mind, such as the DNS servers, the precision for the most visited domains, the impact of packet sampling, and a discussion of the importance of the parameters of the model. After this, Section 5 comments on the findings and outcomes of this work and Section 6 concludes the document summarizing the main results of this work.

2. State of the art

Traffic identification [19] has become a popular issue in the last few years. The reasons to do so, however, are quite varied. For instance, authors in [20] focused on monetization, in particular, they used DNS data to generate website fingerprints that can be later used for traffic identification or even traffic generation. Similarly, authors in [21] used a method called Bag of Domains, akin to Bag of Words for text processing. Although the objectives of these papers are quite similar, the data inputs are rather different. It is clear that DNS data is more reliable, since names might be the same over time or among different locations, whereas IP addresses usually change.

Related to this work, authors in [22] addressed a similar problem of web browsing identification, but with some differences. First, their set of websites includes some domains that are clearly not accessed by users, such as `fonts.gstatic.com`, whereas we only consider websites or main domains such as `google.com`. Also, they use a simple fingerprint model for this task, while we consider more complex deep learning models that make this approach effortlessly generalizable to other setups or list of domains. In contrast to this previous work, the authors in [23] introduce a variety of deep learning models for a different purpose: predicting the next DNS query given the sequence of previously performed queries. Similar to this approach, authors in [24] used deep learning models to identify web browsing activities, but in this case from DNS queries.

Furthermore, authors in both [16,20] tackled the issue of the tangled web that we have explained before, where it is hard to tell when some connections are caused by a particular website. On the other hand, DNS encryption through DNS-over-HTTPS (DoH) [7] is becoming increasingly popular, meaning that the only one that can monetize the data would be the DNS server that receives the encrypted requests. In this case, relying on the IP addresses solves this issue and makes our proposal more future-proof. Similarly, authors in [25] propose a system to determine the traffic generated by a site, but not with classification purposes in mind.

Flow features have also been used to identify web browsing activities. For instance, the work in [26] proposed to use the density estimation of the flow size and binary Bayes belief networks to distinguish between the top 50 most popular websites using anonymized NetFlow logs. This approach, although it is limited to 50 websites, can only detect correctly a 48% of the cases. Moreover, authors of [27] designed a model to identify domain names using flow-level features such as the size of the second packet, the inter-delay between packets, or the number of packet retransmission. Although this approach may detect servers even if IP addresses have changed, the output of the model is different from our proposal. While we predict the website the users are navigating through, they just predict the domain name of each individual connection.

Machine learning and deep learning have also helped the community to build better proposals. In [28], the authors propose a deep learning approach based on Convolutional Neural Networks (CNN) to identify the users' web connection. Despite the novelty, the approach is quite focused on the Tor network case, where security and privacy are a priority but data usage does not play a significant role. Also, the adversary that wants that information is in between the user and the input node of the Tor network. This is quite focused on security issues such as preventing users to access illegal sites, whereas we are focused on the other side—i.e. being near the exit node to use the produced data.

Coarse-grained traffic identification and classification is also useful to profile the users or clients of the network or to provide appropriate Quality of Service depending on the type of service—e.g. P2P, video streaming. In [29], a non-supervised algorithm is proposed to classify traffic into different categories. However, they use URLs for this sake, which entails limitations with the increasing presence of encrypted connections. Also, in [30], a collection of supervised algorithms is

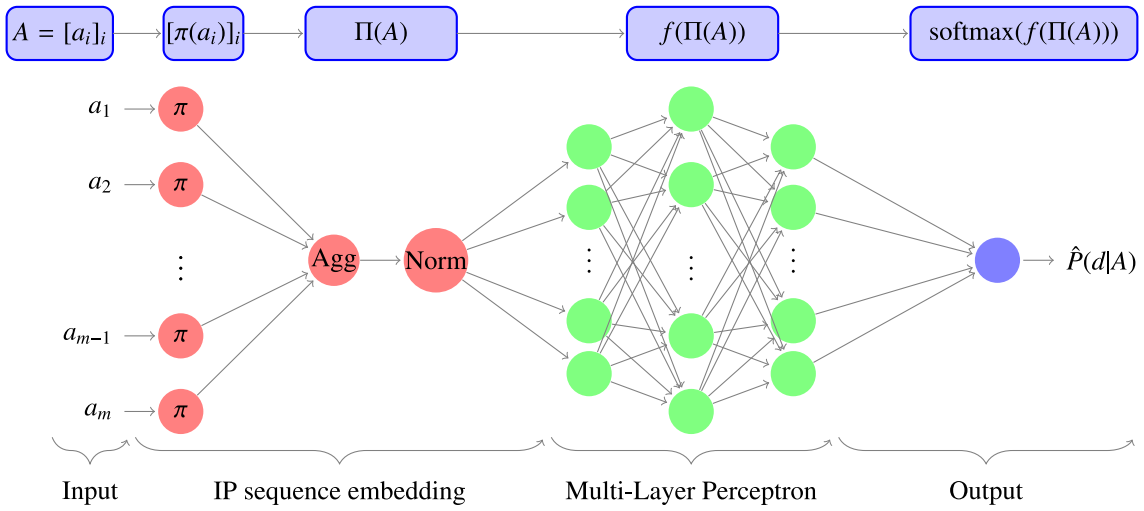


Fig. 1. Visual and mathematical description of the simple model.

applied with a similar purpose, with raw captured traffic as input. In this case, the classes – for instance, www, telnet, or ftp – are very coarse for many purposes, such as web visit identification for monetization and website banning and filtering systems. Authors in [31] provided simple probabilistic signatures, obtained with first n bits of a flow, that can identify network applications, paying special attention to the amount of data they need for that purpose. Furthermore, the limitation of packet traces as input is a huge limit for many situations, where capturing a large amount of traffic can be overwhelming.

A similar topic that is growing in the community is mobile traffic identification, this is, identify if the traffic was produced by a mobile *app* and which *app* produced the traffic. In [32–37], different authors followed a similar approach using neural networks, CNN in particular, to identify the traffic. Authors in [38] provided a detailed overview of this kind of approaches that use deep learning with encrypted data for mobile traffic identification. In contrast to our approach, they mainly focus on encryption and using raw data captures, which limits the final applicability and scalability to scenarios where full capture is feasible. It is worth noting that our proposal can also be applied to this case of mobile data, but it would require to re-train the model with an appropriate dataset. As it was pointed out in [32], most of the traffic they observed is HTTP or HTTPS and many mobile applications are just web views with native API connectors.

Finally, with respect to our previous work [1], we have extended it by better tuning the Multi-Layer Perceptron (MLP) model, adding and adapting the transformer and doing a comparison between them. Additionally, we have included other metrics aside from the accuracy to assess the models. Moreover, we have also studied the impact of the choice of the parameters, such as the hash length when using the hashing trick.

3. Model development

In order to precisely define our approach, we must first specify the inputs and outputs of our models. The input will be a sequence of IP addresses observed by a monitoring system in connections of a particular client, i.e.,

$$A = [a_1, \dots, a_M], \tag{1}$$

where M is the length of the sequence and each a_i is an IP address. It is clear that, in real conditions, M might depend on several factors. However, we will fix M as a global parameter that trims or pads the sequences to the desired length. The output is the vector of probabilities, this is,

$$\mathbf{P} = [P(D = d|A = [a_1, \dots, a_M])]_{d\in\mathbb{D}} \tag{2}$$

where $\mathbb{D}$ is the set of all websites we want to identify, d each website in the set, and D the website to be identified as a random variable. From the point of view of parameters, there are two parameters that may impact the model: $|\mathbb{D}|$, the size of $\mathbb{D}$, and M, the length of the sequence. In both cases, the longer number of parameters, the more accurate the model is; unfortunately, training is also going to take longer.

3.1. IP embeddings

In this first subsection, we cover how to process IP addresses. One of the most important steps to build a neural network model that is fed with IP addresses is to convert the addresses to real-valued numbers or vectors. In fact, either an IPv4 or an IPv6 address are just integers that take values over a huge domain, the upper limits are $U = 2^{32}$ for IPv4 and $U = 2^{128}$ for IPv6. For that sake, we will be using the hashing trick [39] with a linear operator, as we explain below. For both models presented in this paper, the same process applies to convert A into $[\pi(a_i)]_i$.

First, we need a hash function

$$h : \{0, \dots, U\} \to \{0, \dots, v\} \tag{3}$$

that projects the large space $\{0, \dots, U\}$ to a much smaller space $\{0, \dots, v\}$, where U is the aforementioned upper limit and v is usually called the vocabulary size, i.e., the number of different elements given by the hash. Additionally, we expect this function to have the usual properties of a hash, such as distributing the original data in the project space with little collisions. Given that $v \ll U$, collisions are unavoidable, however, this is a price we must pay in order to build a model that fits in memory and that can be trained without major issues. We will evaluate the impact of v on the results in Section 4.4.

The next step is just as simple as mapping each index to a vector, this is, we train w_i, a vector in $\mathbb{R}^\mathfrak{d}$, for each $i \in \{0, \dots, v\}$, where $\mathfrak{d}$ is the dimension of the embedding. The categorical embedding can be defined as

$$\begin{aligned} L &: \{0, \dots, v\} \to \mathbb{R}^\mathfrak{d} \\ &i \to L(i) = w_i \end{aligned} \tag{4}$$

where w_i are the parameters that must be trained in the neural network.

To sum up, both of these functions achieve the objective of projecting an IP address to a continuous domain. In particular,

$$\begin{aligned} \pi &: \{0, \dots, U\} \to \mathbb{R}^\mathfrak{d} \\ &a \to \pi(a) = (L \circ h)(a) = L(h(a)) \end{aligned} \tag{5}$$

is a trainable neural network layer with $v \cdot \mathfrak{d}$ parameters. This means that the number of parameters increases linearly with the vocabulary size, forcing us to find a balance between using a v large enough to achieve the desired accuracy and not excessively large to cope with the constraints in training time and memory usage.

3.2. A straightforward approach

Once we have built a layer to project IP addresses to dense vectors, we have a sequence of real-valued vectors. This sequence should be processed by the model to obtain the prediction. In this matter, several approaches can be taken. First, one can use the simplest possible approach and aggregate all embeddings into a vector by using a global average pooling layer, for instance. This is, we project the sequence A using the previous layer and take the mean. Also, we add a normalization layer to make the training process smoother and avoid any kind of gradient explosion or fading [40].

$$\Pi(A) = \text{Norm}\left[\frac{1}{M}\sum_{i=1}^{M}\pi(a_i)\right]. \tag{6}$$

Eq. (6) will be called simple model, due to its straightforward approach to the matter. Fig. 1 summarizes this model. First, we process the sequence as in the previous subsection to obtain $[\pi(a_i)]_i$. Second, we do an aggregation of all elements of the sequence and a normalization. And finally, we pipe the vector into an MLP to obtain the objective probability as in (2), which is a vector whose components are the probability of visiting each website given the input sequence that has been observed.

3.3. Transformers and the attention mechanism

Other more complex alternatives can also be considered. Since we are processing sequences, Long Short-Term Memory (LSTM) cells, or even generally speaking Recurrent Neural Networks (RNN), might be a useful tool, but they focus too much on the interaction between the elements and the order of the sequence.

In the last few years, the idea of attention has been introduced in the Artificial Intelligence community, and it has been of paramount importance for Natural Language Processing. In particular, self-attention mechanisms might be a good option given that they offer a better performance than LSTM as they are also outperforming other models in many areas [41–43]. Along the different parts, references to Fig. 2 will be made to clarify the main concepts of the attention mechanism and the transformer architecture.

In general, the attention mechanism provides a model that builds and uses an alternative representation of the sequence more akin to the human way of processing sequences and images. In particular, the attention head tries to imitate the way human beings focus on something. When we put our attention on an element, we are able to see not only the element itself with high detail, but also the surroundings with less detail. By incorporating part of the context into each element of the sequence, we get a much better representation of each element of the sequence.

For each attention head, we compute these elements: the query Q, the value V, and the key K, all of them using linear layers with a sequence T as input. In general, each of these elements in an attention network might be computed using different inputs, but in the case of self-attention, all of them are computed using the same input T. Usually, this input T is a token and position embedding, i.e. an embedding that combines the information of the element itself plus its position in the sequence. In our case, we put the positional encoding outside the equation, and we use the IP embeddings we designed as input, i.e. $T = [\pi(a_i)]_i$:

$$Q = \text{Linear(T)} = W_Q \cdot T + b_Q \tag{7}$$

$$V = \text{Linear(T)} = W_V \cdot T + b_V \tag{8}$$

$$K = \text{Linear(T)} = W_K \cdot T + b_K, \tag{9}$$

where W_Q, W_V, and W_K are weight matrices of size (v, d_K), d_K the dimension of the transformer embedding, and b_Q, b_V, and b_K optional biases. Despite all of them being similarly computed, they differ in meaning. Q is a row-oriented matrix where each row corresponds to the query of each element of the sequence, i.e. how each element of the sequence looks for related elements in their context. K is a column-oriented matrix where each column is the key of each element of the sequence, i.e. the value that you use to look-up with the query. On the other hand, V is just a representation of each element of the sequence.

Once these are computed, one can state the self-attention output as

$$\text{Att} = \text{softmax}\left(\frac{Q \cdot K'}{\sqrt{d_K}}\right) \cdot V, \tag{10}$$

this is, we compute the scalar product of all queries and keys (scores in Fig. 2), apply some normalization, and use the value to build the weighted representation of the sequence.

This would make one self-attention head. However, this model usually employs several attention heads, this is, we have different Q_i, V_i, K_i that compute Att_i with $i = 1, \ldots, H$, being H the number of attention heads. All of them are concatenated and aggregated using matrix dot product

$$O = [att_1, \ldots, att_H]W_O + b_0, \tag{11}$$

where W_0 is a weight matrix of size $(H \cdot d_K, v)$ and b_O an optional bias.

Although this is the original version of the formulation of the algorithm given in [17], we think that a more explanatory version of the algorithm is helpful to understand the purpose of the attention mechanism. Algorithm 1 displays a simplified version of the self-attention mechanism with one head. Each $T[i]$, element of the sequence T, has associated three vectors $K[i]$, $V[i]$, and $Q[i]$. $K[i]$ and $V[i]$ represent the element itself and $Q[i]$ represents how other elements may impact the representation of $T[i]$, i.e. how the context is incorporated into the representation. An important detail is that we compute a matrix of attention scores using Q and K. In the real method, the scores are computed as

$$\text{scores} = \text{softmax}\left(\frac{Q \cdot K'}{\sqrt{d_K}}\right), \tag{12}$$

and this matrix models how the context impacts the representation of each element, since (10) can be written as

$$Att = \text{scores} \cdot V. \tag{13}$$

Since the result of the self-attention is still a sequence, we have to process it into a vector, i.e. perform a sequence embedding. For that sake, the following operations are adapted from [17]:

$$O = \text{MultiHeadSelfAttention}(T), \tag{14}$$

$$\text{out}_1 = \text{Norm}(T + O), \tag{15}$$

$$\text{out}_2 = W_{o2} \cdot \max(0, W_{o1} \cdot \text{out}_1 + b_{o1}) + b_{o2}, \tag{16}$$

$$FFN = \text{Norm}(\text{out}_1 + \text{out}_2), \tag{17}$$

and we finally aggregate the sequence into a vector by using an average

$$\Pi(A) = \frac{1}{M}\sum_{i=1}^{M} FFN_i \tag{18}$$

A priori, choosing the simple model or the attention model is just a matter of deciding how complex the model should be, assuming a similar performance. Since packet disorder is quite common and this may result in unreliable results, we decided to ignore order of the elements. In the first case, this means using an aggregation that ignores order, such as the average. In the second case, this would be

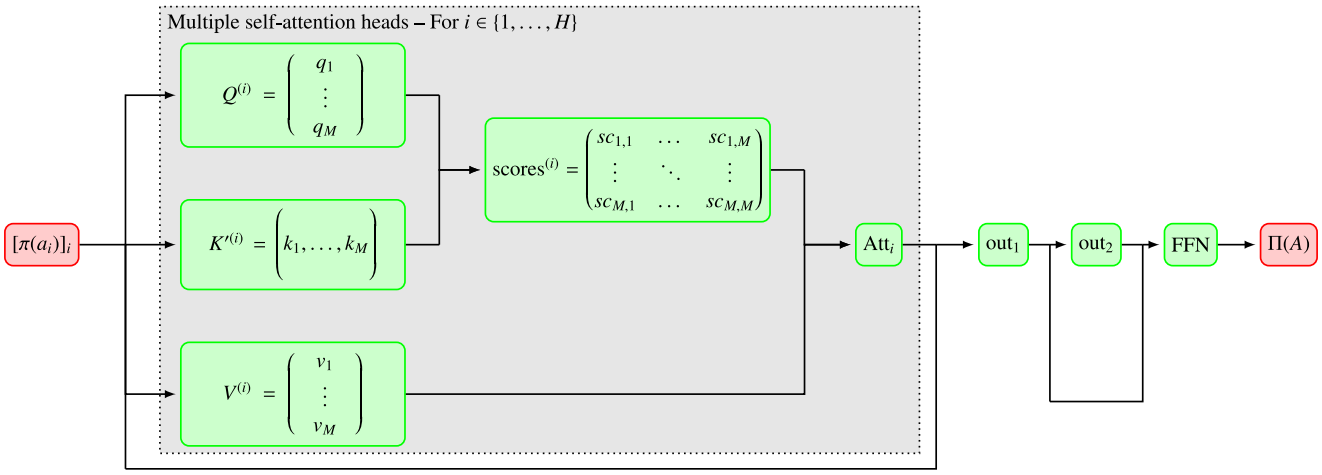


Fig. 2. Visual and mathematical description of the attention model.

Algorithm 1 Simplified self-attention mechanism

procedure ComputeValueKeyVectors(T)
  for $i \in \{0, \ldots, \text{len}(T) - 1\}$ **do**
    $K[i] \leftarrow W_K \cdot T[i]$ ▷ Key vector of $T[i]$
    $V[i] \leftarrow W_V \cdot T[i]$ ▷ Value vector of $T[i]$
  end for
  return K, V
end procedure
procedure OneHeadSimpleSelfAttention(T)
  K, V = ComputeValueKeyVectors(T)
  for $i \in \{0, \ldots, \text{len}(T) - 1\}$ **do**
    $Q[i] \leftarrow W_Q \cdot T[i]$ ▷ Query vector of $T[i]$
    for $j \in \{0, \ldots, \text{len}(T) - 1\}$ **do**
      Attention score of $T[i]$ **w.r.t.** $T[j]$
      scores$[i][j] \leftarrow Q[i] \cdot K[j]$
      $wV[i][j] \leftarrow$ scores$[i][j]V[j]$
    end for
    output$[i] = \sum_j wV[i][j]$
  end for
  return output, scores
end procedure

accomplished by avoiding the position embedding. In both cases, $\Pi(A)$ is a vector of $\mathbb{R}^{\mathfrak{d}}$ that can be used in other models such as a Multi-Layer Perceptron (MLP), denoted hereinafter with f, to predict the output $\mathbf{P}$ as in (2) using a softmax activation function. Once the network has produced a prediction $\hat{P}(d|A)$, the predicted domain, $\hat{d}$, can be computed just by taking the arg max,

$$\hat{d} = \arg\max_d \hat{P}(d|A). \tag{19}$$

Code for these models implemented using TensorFlow 2 [44,45] has been made publicly available for reproducibility of the results.[1]

Once the models have already been defined, we need a particular set of values of the hyperparameters to build instances of the models. Some of these parameters come from the data, such as the number of domains, $|\mathbb{D}|$, the length of sequences, M, or U. However, other parameters depend on the data but cannot be directly estimated. For most of them, we performed a grid search with different reasonable values and Table 1 shows the chosen ones for our dataset, described in the next section. For the rest of the parameters, like the optimizer, loss function or activation functions, we followed standard conventions for text classification problems with neural networks [24].

Table 1
Parameters of the models for our data.

Name	Description	Value
M	Length of the sequence	250
U	Total number of IP addresses	2^{32}
h	Hash type	md5
υ	Hash output dimension	60 000
$\mathfrak{d}$	Embedding dimension	20
H	Number of attention heads	8
d_K	Dimension of the attention key embedding	32
f	Layers of MLP	[75, 150, 250, 350, 400]
	Activation functions	ReLu [46]
Train	Optimizer	Adam [47]
	Loss function	Cat. cross entropy

4. Evaluation

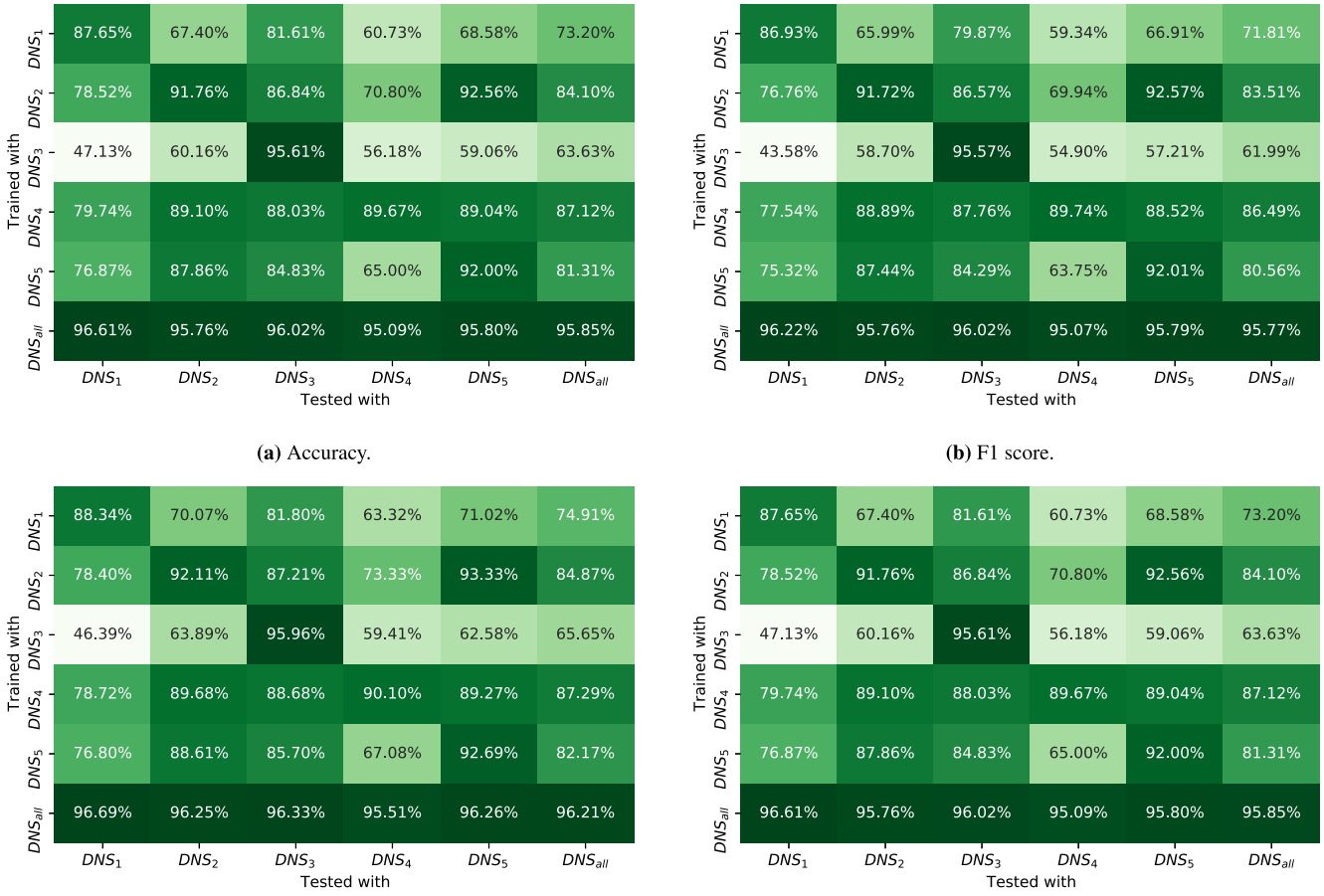
To assess the performance of the model, we have built a dataset consisting of a total of more than 100 000 samples of web browsing activities of sites in Alexa's World Top 500 most visited domains, performed with five different DNS servers. Although DNS server might seem not important, it is quite relevant since observed IP addresses can highly depend on the DNS resolutions of the domains.

These samples were collected using an automatic system previously developed by authors of [20] that was modified to save capture files. Dataset naming conventions and information about DNS servers are shown in Table 2. Also, these datasets are publicly available in aggregated format,[2] for the reproducibility of the results and any follow-up work.

For the dataset, we decided to evaluate it in four different ways: first, we will assess averaged metrics for all the datasets with different DNS servers. Second, we will see the accuracy of our model for the most popular websites, thirdly, we will cover how packet sampling impacts performance of the model, and, lastly, we will analyze the importance of the hash choice and the hash output size in relation to performance.

[1] https://github.com/hpcn-uam/ip-web-analytics/tree/main/code

[2] https://github.com/hpcn-uam/ip-web-analytics/tree/main/dataset



(a) Accuracy.

(b) F1 score.

(c) Precision.

(d) Recall.

Fig. 3. Comparison of the performance obtained training and testing with different datasets for the simple aggregation model.

Table 2
Description of the datasets.

Name	Desc.	DNS server	# samples
DNS_1	Campus DNS	150.244.X.Y	25 000
DNS_2	Google	8.8.8.8	25 000
DNS_3	Cloudfare	1.1.1.1	25 000
DNS_4	Quad9	9.9.9.9	25 000
DNS_5	OpenDNS	208.67.222.222	25 000
DNS_{all}	All datasets	All	125 000

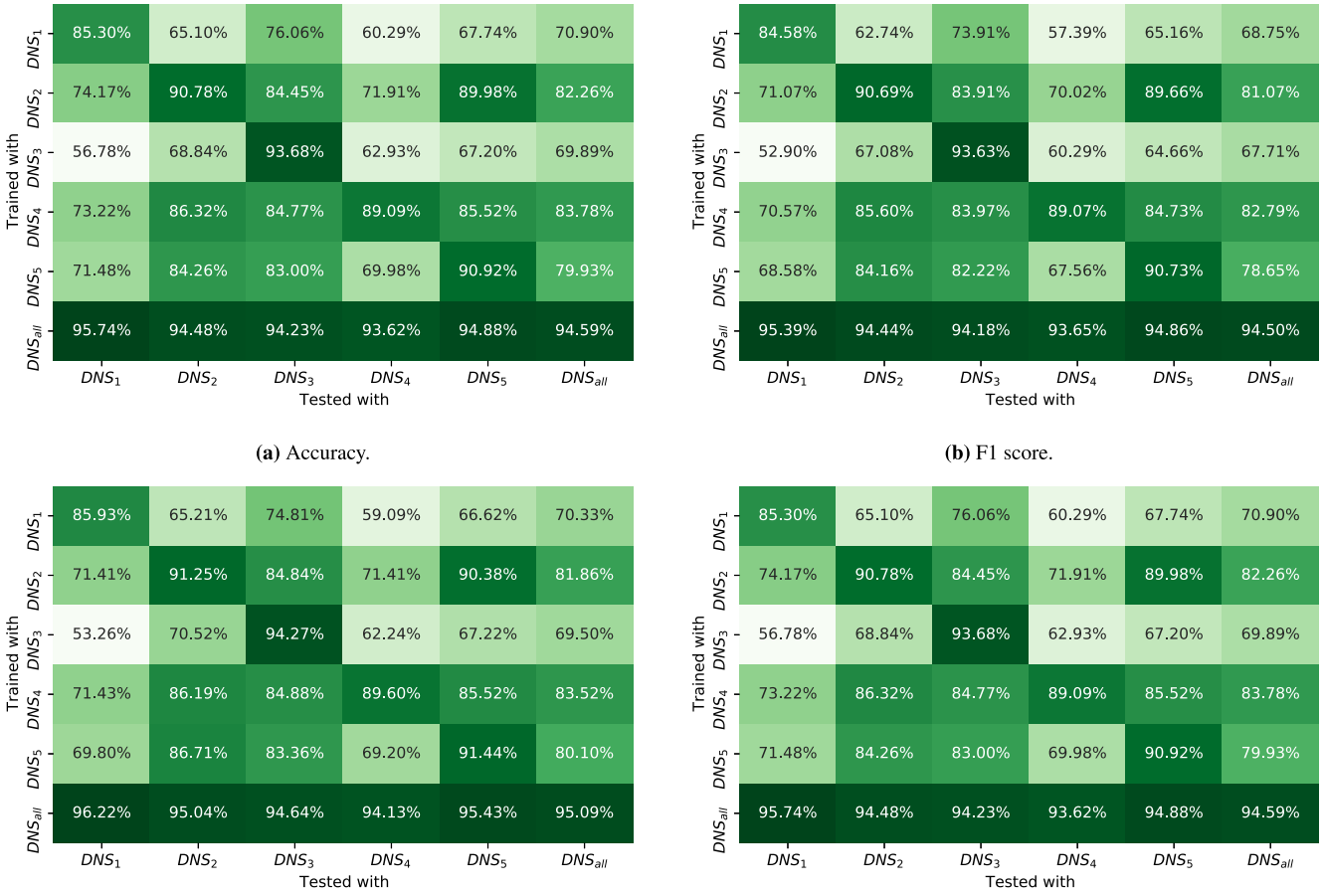
4.1. Effect of name resolution

As we said before, DNS resolution can impact directly the observed IP addresses during the web browsing. Thus, we compare briefly the performance of the model as a function of the data the model has been trained with and the data the model has been tested with.

For this sake, we split each dataset into three pieces: train (65%), validation (15%) and test (20%). We used the train dataset to train all the models, the validation dataset to select the size of the model and control early stopping policies to avoid overfitting, and the test dataset to compute the next results. Figs. 3 and 4 show this comparison for four classical classification metrics:

1. Accuracy: the percentage of corrected predicted domains,
$$\text{accuracy} = \frac{|\{(A,d) \in \text{DNS} : \hat{d}(A) = d\}|}{|\text{dataset}|}, \tag{20}$$
where DNS stands for the whole dataset.
2. Precision: the weighted average of the number of times we predict correctly a domain (TP) over the number of times we predict that domain (TP+FP),
$$\text{precision} = \frac{1}{|D|} \sum_{d \in D} \frac{|\{(A,d) \in \text{DNS}_d : \hat{d}(A) = d\}|}{|\{(A,\_) \in \text{DNS} : \hat{d}(A) = d\}|} \tag{21}$$
where DNS_d stands for the records of the dataset that visits to domain d.
3. Recall: the weighted average of the number of times we predict correctly a domain (TP) the number of samples of that domain (TP+FN),
$$\text{recall} = \frac{1}{|D|} \sum_{d \in D} \frac{|\{(A,d) \in \text{DNS}_d : \hat{d}(A) = d\}|}{|\{(A,d) \in \text{DNS}_d\}|}, \tag{22}$$
4. F1 score: the harmonic mean of the precision and recall,
$$\text{F1 score} = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} \tag{23}$$

Bear in mind that some of these metrics are not properly defined for a non-binary classification problem and, consequently, we have extended them by taking the per-class average of the binary metric.



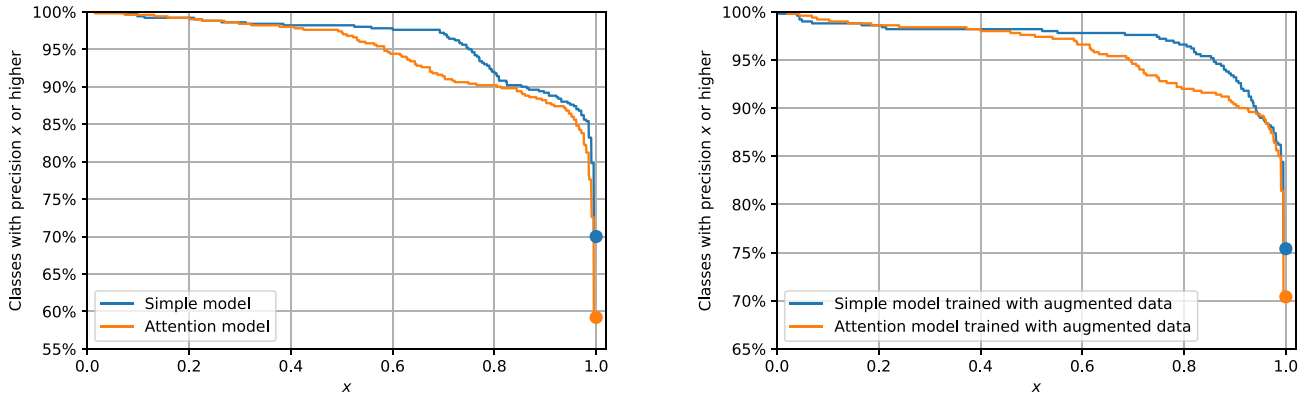
(a) Accuracy.

(b) F1 score.

(c) Precision.

(d) Recall.

Fig. 4. Comparison of the performance obtained when training and testing with different datasets for the attention model.



(a) Without data augmentation.

(b) With data augmentation.

Fig. 5. Performance per website for the simple model and the attention model.

Note also that accuracy and recall can provide similar results when classes are well-balanced.

Although all metrics aim at different objectives, all can be assessed in the same way: the closer to 1, the better. With this in mind, we observed that clearly datasets from DNS_1 to DNS_5 have a similar behavior: they score really well when they are tested with samples from the same dataset, but performance drops significantly with other datasets. It is interesting the case of (DNS_3, DNS_1), where performance drops to worrying levels, especially in the simple model.

On the other hand, the DNS_{all} dataset is clearly more consistent, meaning that we must have a wide variety of data to achieve our objective. In order to be compared in fair conditions, all models were trained with the same number of samples, no matter DNS_{all} is five times larger than the rest of the datasets. This means that DNS_{all} was down sampled to have 25 000 samples just for this experiment.

Comparing the simple model and the attention model, we see that performance is quite similar with small variations. It is true that, in general, the simple model is better than the self-attention one by a 2%,

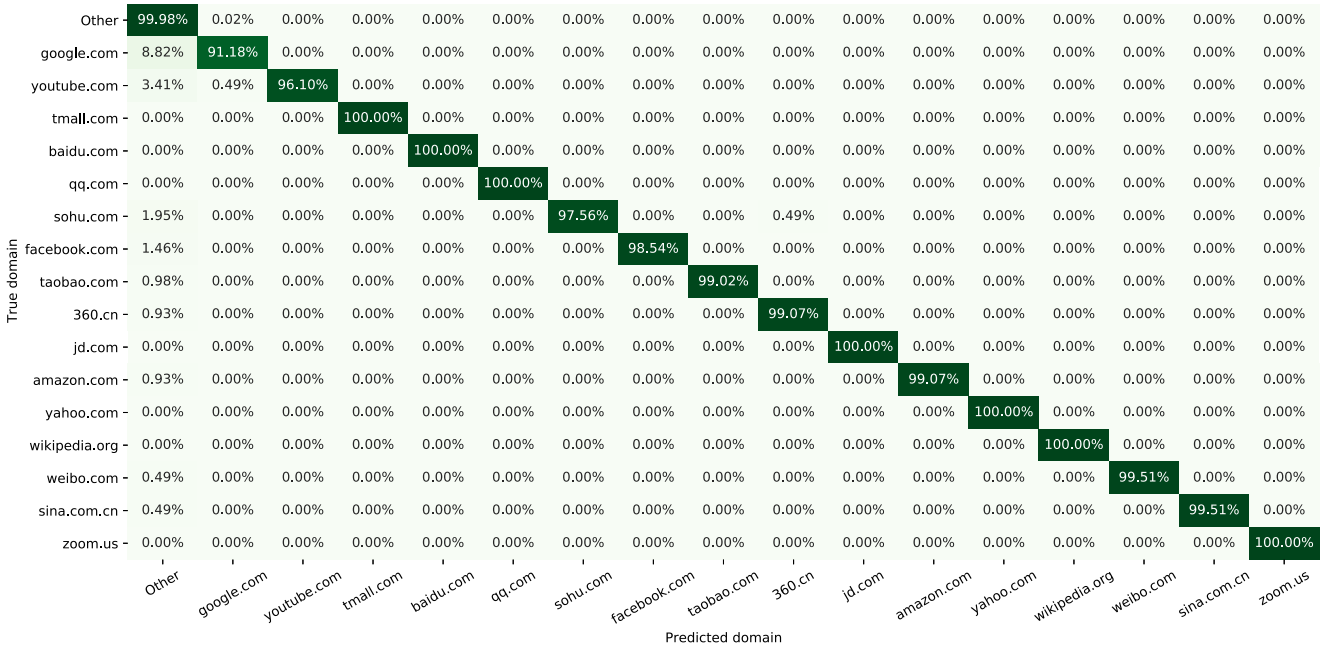
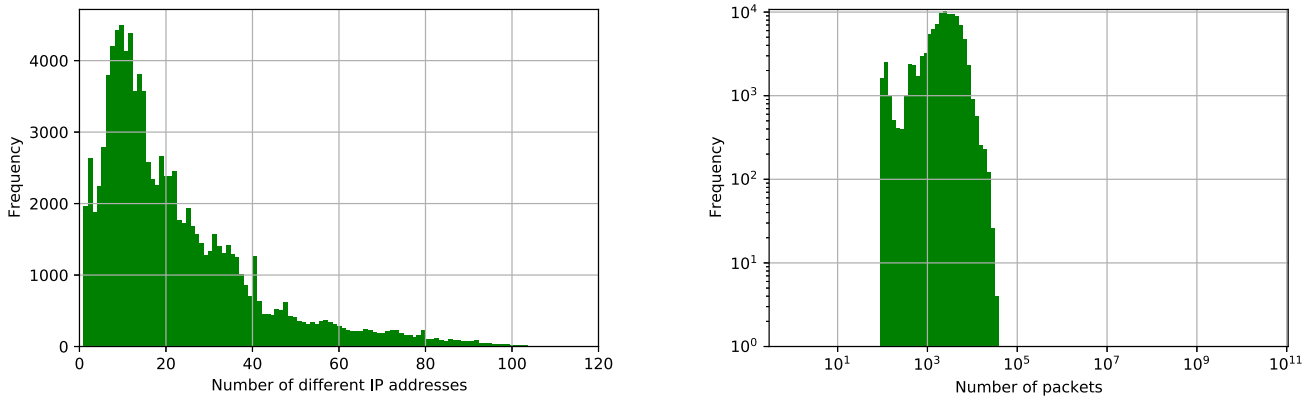


Fig. 6. Confusion matrix for most visited domains in Alexa's World Top 500 for the attention model.



(a) IP addresses. **(b)** Packets.

Fig. 7. Histograms of the number of IP addresses and packets observed per web browsing sample.

but such a small margin is not yet conclusive to discard one model over the other. Further analysis is required.

All the provided metrics give approximately the same results, which means that results are consistent and definitely not ill-conditioned or degenerated.

Apart from DNS resolution, other factors such as the web browser – e.g. Google Chrome or Mozilla Firefox – or the device – e.g. mobile phone, tablet or computer – play a significant role. Authors of [20] had already discussed some of these matters with DNS queries in the past, which entails that the same applies to IP addresses.

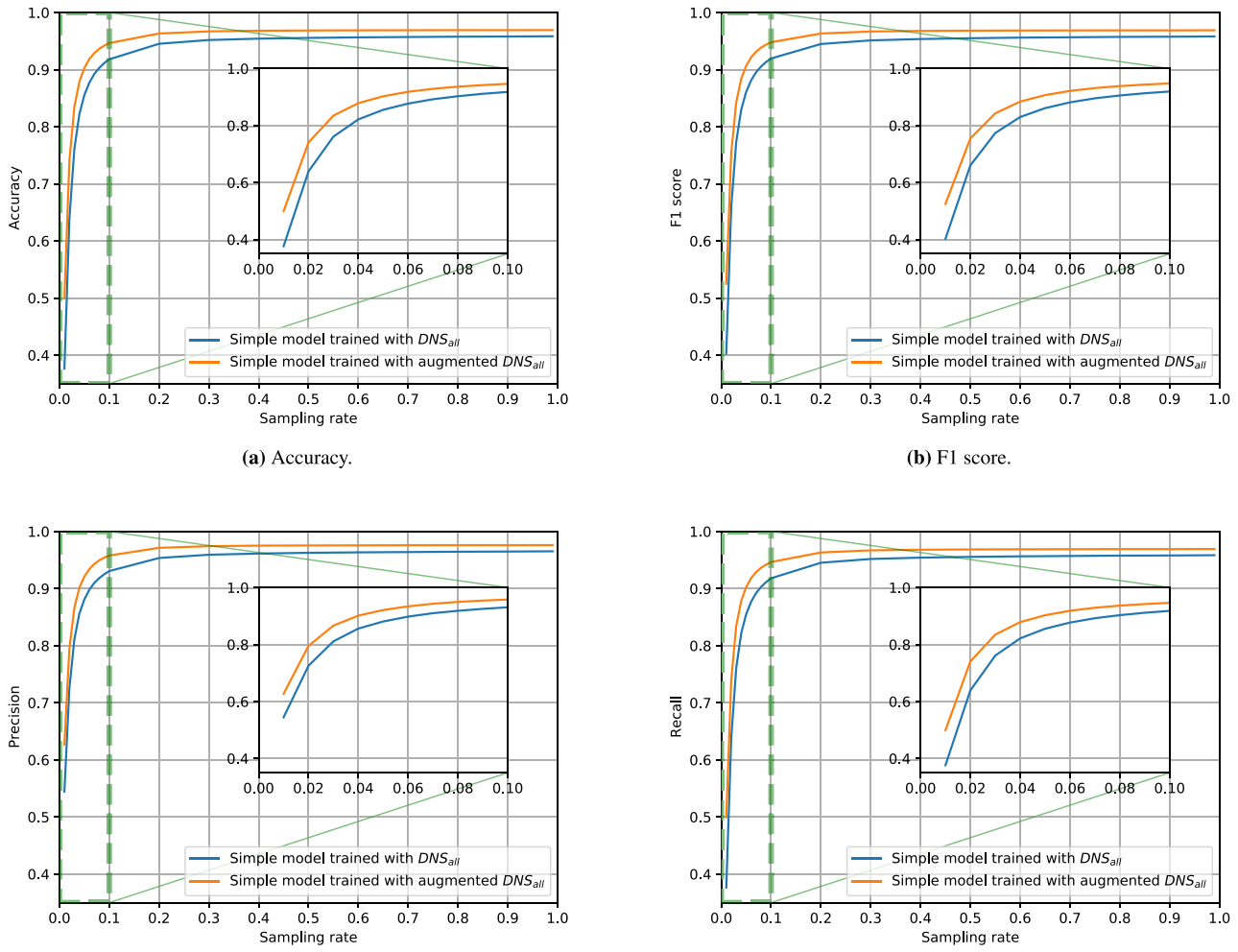
Given the results, hereinafter we will consider just the model trained with DNS_{all} and we will use 65% of it for training, 15% for validation and model selection and 20% for test.

4.2. Detailed view of the most important domains

As it is frequent in classification problems, performance differs from class to class, this is, we have classes whose precision is 100% and classes with a much lower value. It is interesting to see the per-class performance of the model to check that the averaged metrics may vary.

Fig. 5(a) shows the per-class precision and the percentage of websites that achieved at least that precision. In particular, we see that 70% and 60% of the websites are predicted with perfect precision, for the simple and the attention model, respectively. Also, in both models, 90% of the websites are predicted with at least 0.8 of precision. Between 0.4 and 0.8, model performance is around 5% worse for the attention model. This is coherent with previous results, since the attention modelwas performing around 2% less in all metrics, which accounts for this gap of 7% of the websites that have a decrease of accuracy around 0.4.

However, previous analysis regarding Fig. 5 did not take into consideration a possible prior distribution of the classes of our problem, this is, the popularity of the website is assumed to be equal for all the websites. As an example, frequent domains such as Google or Facebook will have the same importance in the final metrics as less frequent domains. In real-world datasets, it is very frequent that the most frequent domains are responsible for 90% of the network traffic [48], the so-called Pareto rule [49] or the Zipf law in the discrete case [50], whereas the 10% remaining is the rest of the web browsing activities. This means that, for many purposes, as long as we are accurate with the top domains, our model will identify successfully a huge amount of the network traffic volume.



(a) Accuracy.

(b) F1 score.

(c) Precision.

(d) Recall.

Fig. 8. Performance of the simple model with and without data augmentation against packet sampling rate per user. Zoomed-in versions between 0 and 0.1 are included inside each plot.

For this sake, Table 3 shows the precision by class, i.e. for each website, for the top domains according to Alexa for the simple model. The category "Others" refer to domains not belonging to the top domains shown in the table. For completeness, the confusion matrix of the simple model can be found in Appendix. Similar results can be obtained for the attention model, as shown in the confusion matrix in Fig. 6.

Overall, the performance with the simple model is outstanding and even perfect for some of the most important domains such as Google, YouTube, or Baidu. It is worth noting that Google and YouTube share some servers, which complicates our task since similar if not the same IP addresses may appear on sequences of both domains.

On the other hand, some domains predictions are less accurate. For instance, Facebook only scores a precision of about 90% in the simple model, meaning that there is a 10% of the times that a user is navigating Facebook, but the model predicts otherwise. Similarly, Google and YouTube are not always correctly detected in the attention model. This could be due to several reasons: similar advertisement presence in other pages, embedded Facebook pages in other domains, or sign-in with Facebook or Google plugins.

Other small errors in the top sites are located in Chinese websites, such as sohu.com, qq.com, 360.cn, or tmall.com. Similar reasons may apply to these cases, where content or advertising could be shared among different websites. Anyway, average precision is over 95%, making the model both accurate and precise.

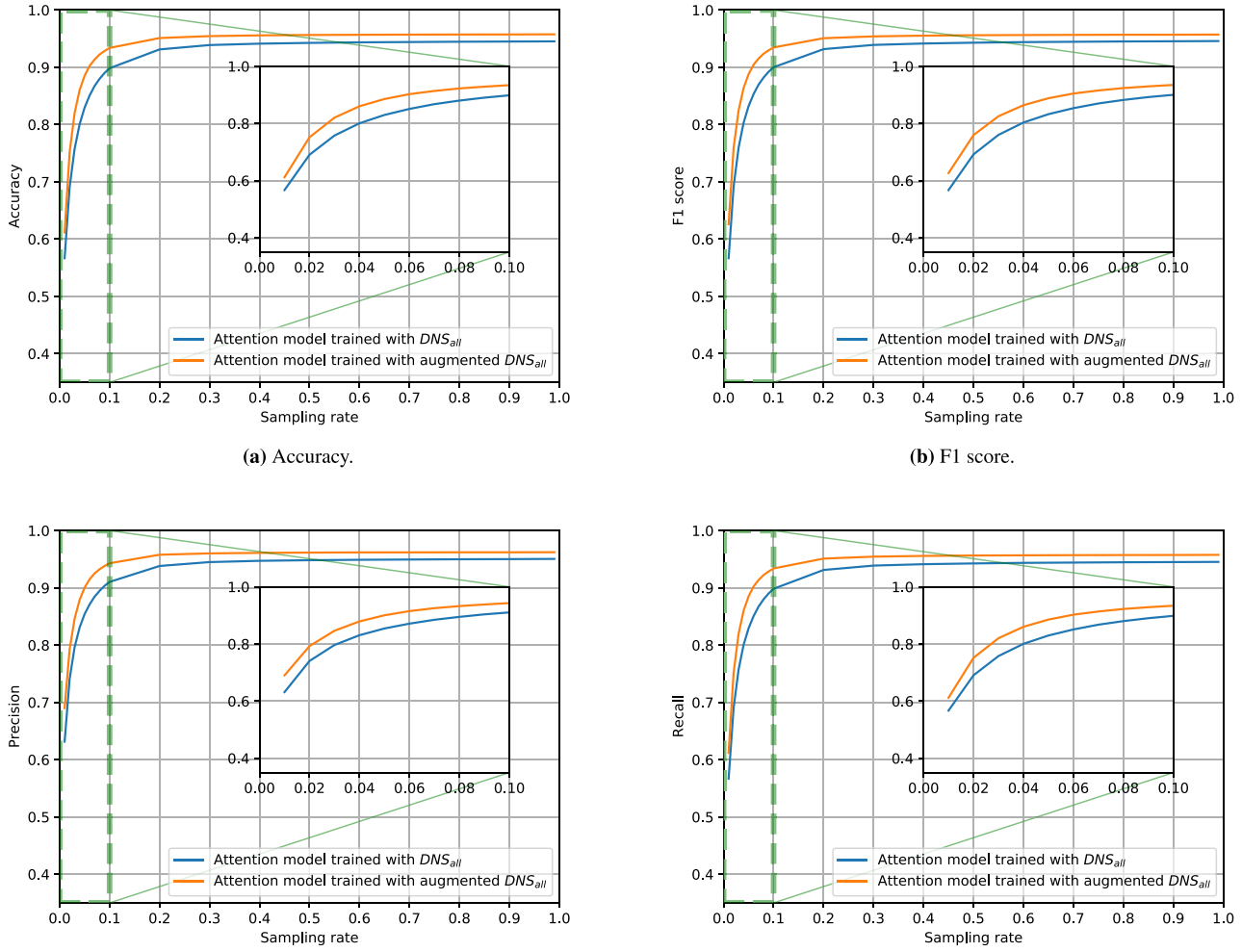
4.3. Impact of packet sampling

The third part of the section covers the impact of packet sampling. First, we want to assess the distributions in our dataset. This provides valuable insights of the dataset that can help us to assess the limits of packet sampling per user, i.e., the minimum number of packets that are required to predict accurately the domain the user is visiting.

Fig. 7 displays both the histogram of the number of observed IP addresses and the total number of packets. The left-hand side of the figure depicts the distribution of the number of IP addresses in each web browsing session, i.e., the distribution of the length of the sequence A. The mode of the distribution is 15, so the most common length is 15, but around 10% percentage of the samples are composed of more than 50 different IP addresses.

On the other hand, the right-hand side shows the histogram of number of packets per web browsing session in logarithmic scale for both axes. As we see, the mode of the distribution is around 10^3, i.e., frequently, the web visits are composed of hundreds to thousands of packets, while only less than 2% of them are composed of more than tens of thousands of packets.

Since we want to know how the model behaves when we start losing information, we simulated packet sampling and look for solutions to mitigate it. In particular, data augmentation can be applied. It consists of extending the training dataset with additional samples which are just



(a) Accuracy.

(b) F1 score.

(c) Precision.

(d) Recall.

Fig. 9. Performance of the attention model with and without data augmentation against packet sampling rate per user. Zoomed-in versions between 0 and 0.1 are included inside each plot.

Table 3
Precision by class for the most visited websites in Alexa's World Top 500 for the simple model.

Website	google.com	youtube.com	tmall.com	baidu.com	qq.com	sohu.com	facebook.com	taobao.com	360.cn	jd.com	amazon.com	yahoo.com	wikipedia.org	weibo.com	sina.com.cn	zoom.us	Others
Prec.	100.00%	100.00%	97.96%	100.00%	95.92%	97.96%	89.80%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	99.96%

the original ones with small modifications or extra noise. In computer vision, it is quite useful to train a model to ignore scale or orientation.

Let $\mathcal{F} = [F_1, \dots, F_M]$ be the vector of frequencies associated to each IP address, this is, a_1 has been observed in F_1 packets. It is simple to convert $\mathcal{F}$ into a normalized vector $\mathbb{F} = \frac{\mathcal{F}}{\sum_{i=1}^{M} F_i}$, which will be used for the packet sampling and whose components are the relative frequencies of each IP address. Then, we can just sample from A using this distribution. If we draw a random sample following the probability distribution $\mathbb{F}$ and we call the empirical frequencies $\mathcal{F}' = [F'_1, \dots, F'_M]$, it is direct how to compute the augmented sample by just taking the elements of A whose element in F' is greater than zero—i.e., the elements that appeared in the sample:

$$A' = [a_i \in A \text{ such that } F'_i > 0]. \tag{24}$$

Once we have A', we can add it to our extended dataset and train the model. Fig. 8 (simple model) and Fig. 9 (attention model) show the results of the four aforementioned classification metrics for both the DNS_{all} dataset and the augmented version of DNS_{all} against the sampling rate of the test subset. For each sample of the dataset, we have added five extra samples for training. Each sample contains only 20% of the original information, i.e., we have used a sampling rate $r = 0.2$.

For all the metrics, we observed a similar behavior: a slight decay of the non-augmented model with sampling rates from 0.2 to 1, followed by a more noticeable reduction between 0.2 and 0.1, ending in a complete drop at around 0.05. We see that the augmented model always improves the performance, reaching a maximum at 100% of sampling rate – i.e. no sampling for testing – of 97% and 96% with the simple and attention model respectively.

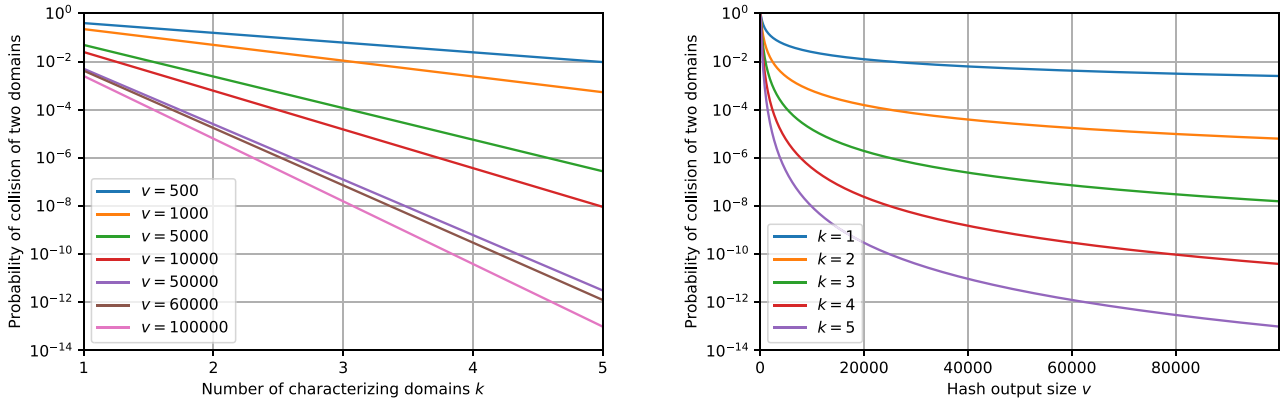


Fig. 10. Estimation of the probability of collision in terms of the number of characteristic domains in the sequence and the hash output size.

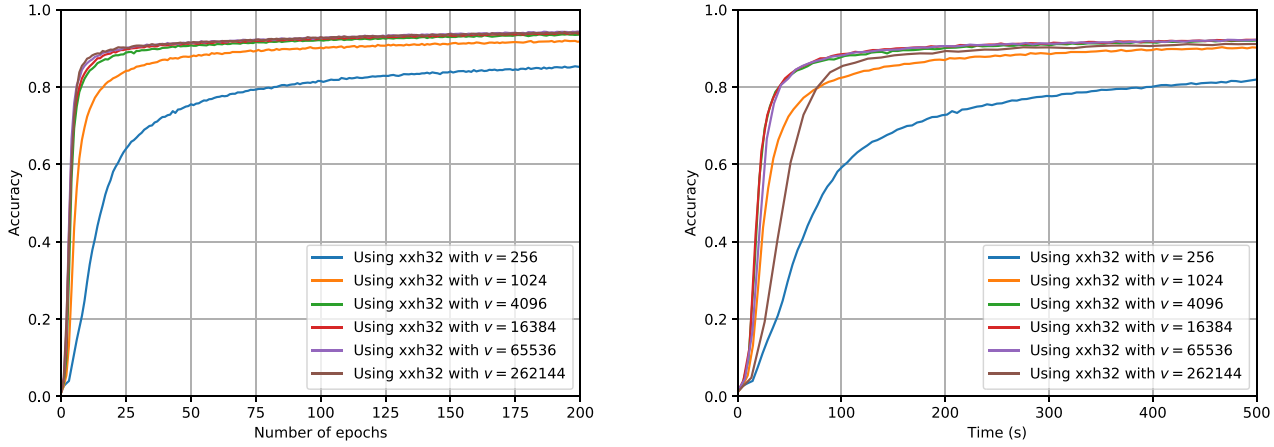


Fig. 11. Training curves with different values of hash output size v for the simple model.

Besides, zoomed-in subfigures show the metrics decay between 0 and 0.1. It becomes apparent that for sampling rates up to 5%, augmented model performance is around 90%. Since we have observed that the mode of the number of packets per sample is around 3 000 packets, it means that we need around 150 packets to have representatives samples that our model predicts with an accuracy of 80%. Besides, if we had 300 packets, accuracy would improve to more than 93%. Comparing both models, we observe a difference when sampling rates are close to zero, when the attention model seems to generalize better than the simple model.

Additionally, the right-hand side of Fig. 5 shows the per-class precision of the models trained with augmented data. In general, we see a 5% increase in both models, being the attention model the one that benefits the most of this. The shape of the curves slightly changes, making them more uniform. However, the difference between models became more apparent in the range of accuracies between 0.4 and 0.9.

4.4. Impact of hash choice and hash collisions

Up to this point, there is no clear evidence that the hash collisions might be playing an important role here. However, model size hugely depends on the size of the input, this is, the hash output dimension v. Therefore, optimizing the model implies reducing v as much as possible.

First, a theoretical analysis can be helpful in order to determine potential issues and have a preliminary estimation. Let $h(A)$ be a sequence of hashed IP addresses, $\{b_1 = h(a_1), \dots, b_m = h(a_m)\}$. Some assumptions must be made for this purpose:

- *There are at most k characterizing IP addresses in the sequence*. This means that the prediction of the model is fundamentally based on the presence of a particular subset of the sequence, hereinafter called characterizing IP addresses or domains. Without loss of generality, these characterizing elements will be the first k of the sequence, since both models ignore the order of the elements. This entails that $\forall d \in \mathbb{D}$

$$P(d|h(A) = \{b_1, \dots, b_m\}) = P(d|h(A) = \{b_1, \dots, b_k\}). \tag{25}$$

- *The hash output is uniform*. This means that the distribution of the outputs of the hash is uniform, given that the input is also uniformly distributed. We have tested this hypothesis on the data using a χ^2 test for different hashes and values of v, and, with a p-value of 1, we have for all of them that it holds, i.e.

$$X \sim \text{Uniform}([0, U]) \implies h(X) \sim \text{Uniform}([0, v]). \tag{26}$$

Let B' be another sequence of hashed IP addresses, $\{b'_1, \dots, b'_m\}$ belonging to a different website. We want to compute what is the probability of B' getting confused with $h(A)$. As reasoned before, the characterizing

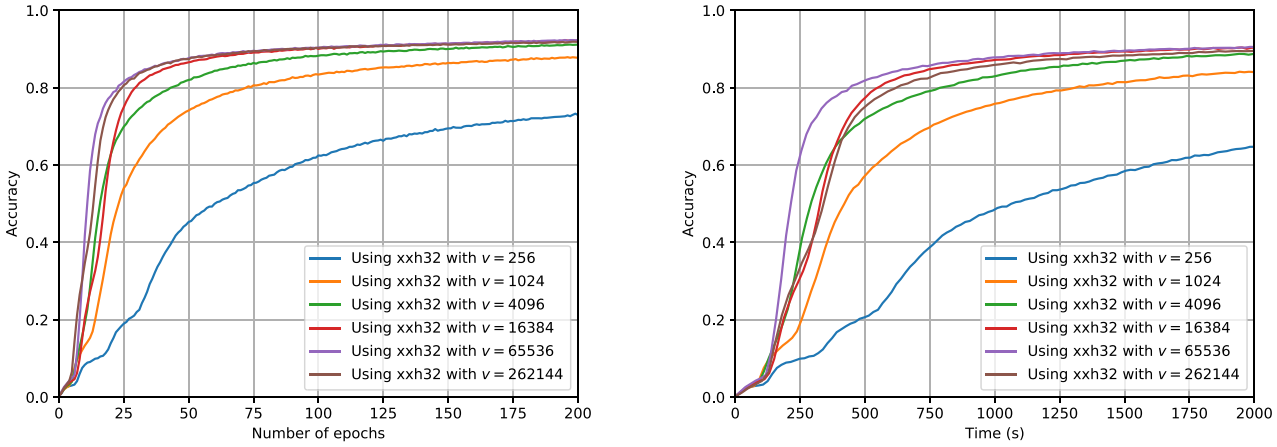
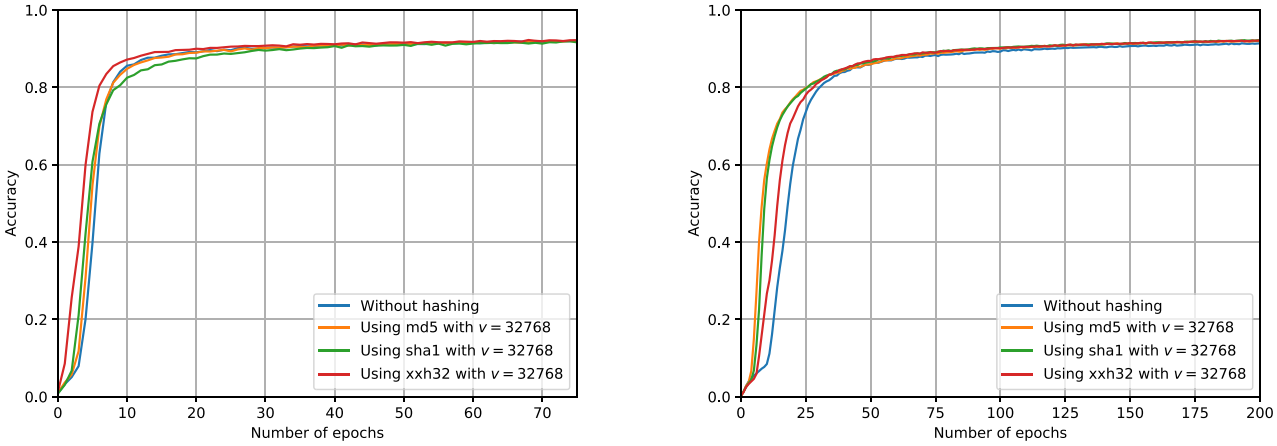


Fig. 12. Training curves with different values of hash output size v for the attention model.



(a) simple model. **(b)** attention model.

Fig. 13. Training curves with different types of hashes for both simple model and attention model.

IP addresses are the ones responsible for the prediction, so if we have all characterizing domains in B', then we will have that the model predicts incorrectly due to hash collisions. Assuming $k = 1$, it is simple to compute the probability of incorrect prediction

$$\begin{aligned} I_1 = P(d_B = d_{h(A)}) &= P(\exists i \in 1 \dots, m : b'_i = b_1) \\ &= 1 - P(\forall i \in 1 \dots, m : b'_i \neq b_1) \\ &= 1 - \prod_{i=1}^{m} P(b'_i \neq b_1) \\ &= 1 - \left(\frac{v-1}{v}\right)^m \\ &= \frac{v^m - (v-1)^m}{v^m}. \end{aligned} \tag{27}$$

Once this is done, it can also be computed for $k = 2$ following a similar procedure

$$\begin{aligned} I_2 &= P(\exists i, j \in 1 \dots, m : b'_i = b_1, b'_j = b_2) \\ &= P(\exists j : b'_j = b_2 | \exists i : b'_i = b_1) P(\exists i : b'_i = b_1) \\ &= \left(\frac{v^m - (v-1)^m}{v^m}\right)\left(\frac{v^{m-1} - (v-1)^{m-1}}{v^{m-1}}\right). \end{aligned} \tag{28}$$

A general formula for the probability with k characterizing domains can then be derived

$$I_k = \prod_{j=m-k+1}^{m} \left(\frac{v^j - (v-1)^j}{v^j}\right), \tag{29}$$

and we can provide an upper bound $\mathcal{U}(k, v)$

$$I_k \leq \mathcal{U}(k, v) = \left(\frac{v^m - (v-1)^m}{v^m}\right)^k. \tag{30}$$

The previous estimation implies that I_k behaves asymptotically like $\frac{1}{v^k}$. Fig. 10 shows both I_k in terms of k for different hash output sizes v and the same quantity in terms of v for different values of the number of characterizing IP addresses k. In a worst-case scenario, this means that for $v = 60\,000$, the probability of collision would be less than 1% if $k = 1$. However, we found that for small values of v such as 200, the model could reach 86% of accuracy, which entails that the number of collisions is really far from a really catastrophic $\mathcal{U}(k = 1, v = 200) \approx 71\%$. It should be around $\mathcal{U}(k = 3, v = 200) \approx 36\%$ and $\mathcal{U}(k = 4, v = 200) \approx 26\%$ at most, therefore, it is more likely an average value of k around 3. This would mean that the probability of collision between two domains must be around 10^{-7}, i.e. it is negligible even for huge datasets.

Nevertheless, it is a parameter that impacts model size and model training time. In general, it is desirable that v is large enough to fit the data without many collisions but small enough to prevent the model to become too large to be handled. Fig. 11 shows the training performance in terms of number of epochs and time for the simple model.

In the former case, we can see that training curves are similar for $v \geq 4096$, meaning that there is little to no difference between these

values. For $\upsilon \leq 1024$, training curves are different, but they reach 91% and 86% of accuracy for $\upsilon = 1024$ and $\upsilon = 256$, respectively. In the latter case, when we change the axis to time instead of epochs performance differences are negligible when $\upsilon \geq 4096$, this is, only for small values of υ, such as $\upsilon = 256$, we can see the impact of hash collision. This exhibits the trade-off between complexity and accuracy.

Fig. 12 shows the same curves, but for the attention model. Again, for $\upsilon \geq 16\,385$, performance is really similar both in terms of number of epochs and time. This is expected, since training transformers will usually take much more time and effort than most of the simple model. However, we saw before that the attention model is better at generalizing and learning context, as it was depicted in previous section.

Furthermore, hash function choice is not playing an important role either. One may think that hash function choice may impact model performance, so the best hash must be chosen. Fortunately, this is not the case, as shown in Fig. 13. In the left-hand side, training curves for the simple model are displayed, including also in this case the training without using hashes, this is, assigning a number to each IP address in the dataset (this has been possible by using a GPU larger than the one available at the beginning of this work). They show a slight difference in the very first epochs, but it is negligible when 20 epochs are surpassed. Similarly, the right-hand side figure shows the same curves for the attention model, where behavior is almost identical, proving our point. The figures also show that hashes provide similar results than a non-hashed solution. Therefore, we can choose the faster hash function to reduce the computing times, and work with the hashing trick when the number of IP addresses to be analyzed is unknown or very large, making it a scalable solution.

4.5. Comparison with other tools

In Section 2, we covered several alternatives to this problem from a qualitative point of view. To give the full picture, it is necessary to provide some numerical comparison alongside the main differences to ensure our solution performance is similar to the state of the art, or better. In [20], they provide a system that relies on DNS data for the same purpose. They managed to obtain an F1-score of 94%, comparable to both simple model and attention model. However, the emulation of the DNS cache that they are doing scales poorly, requiring the use of big data frameworks to deal within an ISP network. In contrast, our work allows deploying the model in several points of the network, and since it does not require keeping any per-client state, it can easily scale up to any number of users.

In next work [24], they rely on NLP techniques to avoid the emulation of the cache, offloading that learning process to NLP techniques. They achieve between 90%–95% of accuracy, again comparable to this work. In both cases, the same level of performance was obtained, but with some qualitative differences. The most obvious one is the type of inputs, in both works they use DNS requests with domain names, whereas we used IP addresses, a more challenging problem. On the other hand, the training data of these alternatives might be valid more time than in our case. This is because domain names in websites do not usually change, in contrast to IP addresses, which may even depend on DNS resolvers or on customers' geo-location.

Authors in [22] proposed a method that uses similar data input, sequences of IP address. They rely on a custom algorithm doing fingerprinting. Their accuracy is around 84%, obtaining 78% for websites they considered sensitive since they may leak your political views, sexual orientation or religion. In contrast to our work, the number of websites included is higher, around 200 000, but with a number of samples per website much lower, 24 per website for their work against 200 per website of our work. We were also concerned about the data diversity, i.e. obtaining many samples over time that are not always equal by changing DNS resolvers. This guarantees significance of the results as well as it makes our evaluation closer to real-world situations.

5. Discussion

During this work, we have observed several contributions and remarkable ideas:

1. *Web browsing can be inferred from IP addresses*: this was the main objective of this work and the model achieved it with accuracy higher than 90%. It is worth mentioning that data plays a key role, so updated and varied data are required; as we observed with DNS servers. As long as the training data is representative of the users' behavior, web browsing activities can be easily inferred. This proves that, as long as an actor has data about where you are connecting to, web traffic identification can be easily achieved.
2. *Pay attention to the most popular websites*: because dataset imbalance can be problematic for training, it is useful to use a balanced dataset. However, top domains must be taken into consideration separately, since they are responsible for the majority of the traffic. It does not matter if the model is 99% accurate if it does not work for these domains that represent, in most cases, 90% of the traffic. Providing a detailed analysis is advisable to know better how the model works and which popular domains can be confused.
3. *Data augmentation and sampling improve the model*: as we mentioned before, data augmentation is a powerful way of extending training datasets and teaching the model not to pay attention to some facts. In this case, we simulate traffic sampling so that we generate new samples to be incorporated into the training set. It makes the model more resilient to sampling and accurate in all cases, since it allows the learning to be focused on features that are more likely to happen, even under packet sampling. We also did study how many packets the model would need to provide an accurate prediction, which specifies when the model result is trustworthy. This answers our concerns with respect scalability: the model can scale easily for a huge number of websites.
4. *Hashing trick makes the model scalable*: we have shown that hash output size (υ) is an important parameter regarding collisions and thus performance. The larger υ, the fewer collisions and better performance. However, this benefit from increasing υ further and further comes with a cost in computational resources for training and prediction. It has been shown that, even for relatively small values of υ, the model managed to achieve 90% accuracy, meaning that one can keep this parameter low, allowing the model to fit in many commodity devices, even for large datasets with thousands of websites and millions of different IP addresses.
5. *There is no perfect model*: in general, both models achieve a similar performance, higher than 90% of accuracy. The simple model is more fine-tuned and its cost-performance trade-off is magnificent, making it a great alternative in really demanding scenarios or where computation resources such as GPUs are scarce. On the other hand, self-attention mechanism takes more time to achieve a similar level, but we have observed that this model is more resilient to gaps in the information (compared to itself), like when packet sampling is applied. In the end, it is all about choosing the right solution for each scenario while analyzing the aforementioned factors.

However, there are few drawbacks about this approach:

1. *Periodical re-training is needed*: given the dynamic nature of many IP addresses, models are expected to drop performance as time passes since IP addresses change. A better solution would be to use domain names when they are available. Authors in [20, 24] already covered these solutions, but they are prompt to disappear with the advent of encrypted DNS (e.g. DoH) and eSNI. Consequently, the only possibility is to assume the costs of

re-training the model periodically. How performance decreases as the data becomes older and older has already been studied in [20,22], stating that data normally starts to affect performance after one week, but most of the data still was valid even after two months.

2. *Data of quality is key*: we observed that the results vary significantly if the DNS server changes. There are other variables such as the location, the user agent or the type of device that may affect the results. Therefore, obtaining high quality datasets is required to get a system that works on general setups. Most of the effort should be put on building a highly curated dataset instead of the fine-tuning of the models.
3. *NAT adds noise*: in our approach, we assumed that users may access different websites, but never concurrently. This might be true for most of the households, but when a home connection might be providing internet connection to tens of devices, the probability of two devices accessing several websites at the same time increases. Also, some ISPs employ Carrier-Grade NAT (CG-NAT), hiding many customers behind a single IP address, making this probability really close to 1. This could be solved either by assuming your model outputs a percentage of usage instead of a probability of visiting, which requires a dataset with tuples of websites being visited at the same time; or by separating users traffic. Being the former more reasonable in some contexts, the latter has also some state of the art in other fields such as signal processing, where distinguishing several speakers in an audio signal is called speaker diarization.

These obstacles open a brand-new research line in how to build these datasets. It implies realistic user behavior simulation and user device emulation, as well as external variables to be considered. Besides, they should be built cost-effectively, so instead of doing thousands of website visits, generative models, such as Generative Adversarial Networks [51] or diffusion models [52], can be the solution to incorporate many factors and increase the diversity of the data while keeping the cost of obtaining new data under control.

6. Conclusion

Along this work, we have presented a model that effectively identifies web browsing activities just using IP addresses. As we motivated, this means that there are potentially many actors that are able to use your data without your consent. This should not only concern users, but also ISPs that may be already providing this data to third-parties. In fact, even some of the best alternatives presumably, such as VPNs, to protect your data and your privacy cannot totally bypass this, which means that we must be aware of this possibility.

The obtained results also proved that the model is scalable, as the number of domains increase, and it can be deployed in a real environments thanks to the good performance for low values of the hash output size υ. Furthermore, the model can be easily scalable as the number of customers increases, given that the model can be deployed in several points of the monitoring infrastructure, each of them focused on a portion of all the customers. Some drawbacks have also been covered, but bear in mind that most of them can actually be solved with just a highly-curated training dataset.

In order to overcome this problem and avoid traffic identification, we foresee the following countermeasures depending on which side of the connection you are. From the user perspective, if the user navigates through Tor or a proxy chain, the browser should use a different exit node for each connection. In this way, it becomes more difficult to guess which website the user is visiting. Also, the user can employ a noise generator – i.e. a program or browser plugin [53] that generates random web browsing activities – to make your data harder to analyze and therefore less valuable.

Some service providers may want to provide extra protection to their users. However, on this side, actions are pretty limited. Servers should avoid any static IP address and change IP addresses within a large set every short period of time so that it becomes nearly impossible to train any model before the IP address changes again. This increases management costs, and routing issues may arise, but they are left with little alternative.

Yet, there are still some open related research lines. First, it is interesting to study the lifespan – validity over time – of the datasets. One of the aforementioned issues that potentially may face these approaches is to determine when the training data is very old to be representative of the real-world data. Selectively choosing the websites that change a lot over time in contrast to the ones that remain mostly static could also be key to provide a cost-effective update of the training dataset. As mentioned before, obtaining cost-effectively a good dataset is key to achieve traffic identification.

Besides, other data sources could be considered, especially, how SDN can enrich this model. In particular, it is interesting to see how network equipment can be programmed to send to the monitoring platform the first packet of each connection. Therefore, data reduction is optimal, and it is horizontally scalable. Finally, this methodology can also be applied to mobile apps. Applications can be identified using their interactions at the IP level.

CRediT authorship contribution statement

Daniel Perdices: Conceptualization, Methodology, Software, Formal analysis, Investigation, Data curation, Writing – original draft, Visualization. **Jorge E. López de Vergara:** Conceptualization, Methodology, Writing – original draft, Supervision, Funding acquisition. **Iván González:** Software, Data curation, Resources, Writing – review & editing, Project administration. **Luis de Pedro:** Writing – review & editing.

Data availability

We have published both data and code in GitHub[3].

Acknowledgments

This research has been partially funded by the Spanish State Research Agency under the project AgileMon (AEI PID2019-104451RB-C21) and by the Spanish Ministry of Science, Innovation and Universities under the program for the training of university lecturers (Grant number: FPU19/05678).

Declaration of competing interest

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

Appendix. Confusion matrix for the simple model

For the sake of completeness, Fig. 14 depicts the confusion matrix for the top domains according to Alexa.

[3] https://github.com/hpcn-uam/ip-web-analytics/

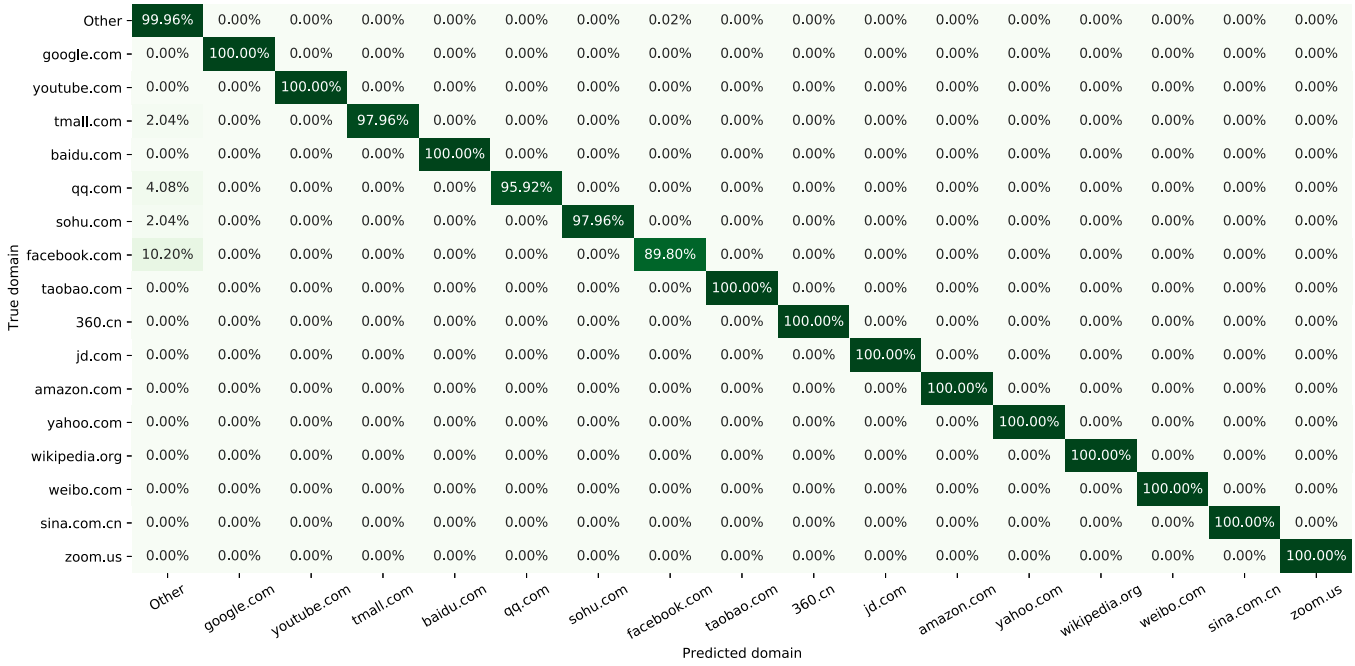


Fig. 14. Confusion matrix for most visited domains in Alexa's World Top 500 for the simple model.

References

[1] D. Perdices, J.E. López de Vergara, I. González, Assessing the limits of privacy and data usage for web browsing analytics, in: 2021 17th International Conference on Network and Service Management, CNSM, 2021, pp. 173–179, http://dx.doi.org/10.23919/CNSM52442.2021.9615560.

[2] European Commission, The European data strategy, 2020, URL https://ec.europa.eu/commission/presscorner/detail/en/fs_20_283.

[3] US Government, Federal Data Strategy: Leveraging Data as Strategic Asset, 2021, URL https://strategy.data.gov/overview/.

[4] L. Liu, The rise of data politics: Digital China and the world, Stud. Comparat. Int. Dev. 56 (1) (2021) 45–67, http://dx.doi.org/10.1007/S12116-021-09319-8/TABLES/1.

[5] European Commission, How do EU citizens manage their personal data online?, 2020, URL https://ec.europa.eu/eurostat/web/products-eurostat-news/-/EDN-20210128-1.

[6] D. Benjamin, Using TLS 1.3 with HTTP/2, Request for Comments, 8740, RFC Editor, 2020, http://dx.doi.org/10.17487/RFC8740.

[7] P.E. Hoffman, P. McManus, DNS Queries over HTTPS (DoH), Request for Comments, 8484, RFC Editor, 2018, http://dx.doi.org/10.17487/RFC8484.

[8] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, P.E. Hoffman, Specification for DNS over Transport Layer Security (TLS), Request for Comments, 7858, RFC Editor, 2016, http://dx.doi.org/10.17487/RFC7858.

[9] R. Dingledine, N. Mathewson, P. Syverson, TOR: The Second-Generation onion router, in: 13th USENIX Security Symposium, USENIX Security 04, USENIX Association, San Diego, CA, 2004, pp. 1–17.

[10] Brave Software Inc., Brave browser: Secure, fast and private web browser, 2022, https://brave.com/. (Accessed 07 March 2022).

[11] The Hacker News, Over 25% of tor exit relays spied on users' dark web activities, 2021, https://thehackernews.com/2021/05/over-25-of-tor-exit-relays-are-spying.html. (Accessed 19 May 2021).

[12] The Tor Project, Tor security advisory: exit relays running SSLstrip in May and June 2020, 2021, https://blog.torproject.org/bad-exit-relays-may-june-2020. (Accessed 19 May 2021).

[13] B. Claise, Cisco Systems NetFlow Services Export Version 9, Request for Comments 3954, RFC Editor, 2004, http://dx.doi.org/10.17487/RFC3954.

[14] P. Aitken, B. t Claise, B. Trammell, Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information, Request for Comments 7011, 2013, http://dx.doi.org/10.17487/RFC7011.

[15] X. Hu, N. Sastry, What a tangled web we weave: Understanding the interconnectedness of the third party cookie ecosystem, in: WebSci 2020 - Proceedings of the 12th ACM Conference on Web Science, Association for Computing Machinery, Inc, 2020, pp. 76–85, http://dx.doi.org/10.1145/3394231.3397897.

[16] I.N. Bermúdez, M. Mellia, M.M. Munafo, R. Keralapura, A. Nucci, DNS to the rescue: Discerning content and services in a Tangled Web, in: Proceedings of the 2012 Internet Measurement Conference, IMC '12, Association for Computing Machinery, New York, NY, USA, 2012, pp. 413—426, http://dx.doi.org/10.1145/2398776.2398819.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS '17, Curran Associates Inc., Red Hook, NY, USA, 2017, pp. 6000—6010.

[18] Amazon Web Services, Alexa - Top sites, 2020.

[19] A. Callado, C. Kamienski, G. Szabó, B.P. Gero, J. Kelner, S. Fernandes, D. Sadok, A survey on internet traffic identification, IEEE Commun. Surv. Tutor. 11 (3) (2009) 37–52, http://dx.doi.org/10.1109/SURV.2009.090304.

[20] J.L. García-Dorado, J. Ramos, M. Rodríguez, J. Aracil, DNS weighted footprints for web browsing analytics, J. Netw. Comput. Appl. 111 (2018) 35–48, http://dx.doi.org/10.1016/j.jnca.2018.03.008.

[21] M. Trevisan, I. Drago, M. Mellia, M.M. Munafo, Towards web service classification using addresses and DNS, in: 2016 International Wireless Communications and Mobile Computing Conference, IWCMC 2016, IEEE, 2016, pp. 38–43.

[22] N.P. Hoang, A.A. Niaki, P. Gill, M. Polychronakis, Domain name encryption is not enough: Privacy leakage via IP-based website fingerprinting, Proceedings on Privacy Enhancing Technologies 2021 (4) (2021) 420–440, http://dx.doi.org/10.2478/popets-2021-0078.

[23] J. Merlino, P. Rodríguez-Bocca, Short-time prediction of DNS queries using deep learning and pre-trained word embedding, in: 2021 XLVII Latin American Computing Conference, CLEI, 2021, pp. 1–10, http://dx.doi.org/10.1109/CLEI53233.2021.9640221.

[24] D. Perdices, J. Ramos, J.L. García-Dorado, I. González, J.E. López de Vergara, Natural language processing for web browsing analytics: Challenges, lessons learned, and opportunities, Comput. Netw. 198 (2021) 108357, http://dx.doi.org/10.1016/j.comnet.2021.108357.

[25] M. Trevisan, I. Drago, M. Mellia, H.H. Song, M. Baldi, WHAT: A big data approach for accounting of modern web services, in: Proceedings of the 2016 IEEE International Conference on Big Data, IEEE Big Data 2016, IEEE, 2016, pp. 2740–2745, http://dx.doi.org/10.1109/BigData.2016.7840921.

[26] S. Coull, M. Collins, C. Wright, F. Monrose, M. Reiter, On web browsing privacy in anonymized NetFlows, in: 16th USENIX Security Symposium, USENIX Security 07, USENIX Association, Boston, MA, 2007, pp. 339–352.

[27] M. Trevisan, F. Soro, M. Mellia, I. Drago, R. Morla, Does domain name encryption increase users' privacy? SIGCOMM Comput. Commun. Rev. 50 (3) (2020) 16–22, http://dx.doi.org/10.1145/3411740.3411743.

[28] S. Bhat, D. Lu, A. Kwon, S. Devadas, Var-CNN: A Data-Efficient Website Fingerprinting Attack Based on Deep Learning, Proc. Priv. Enhanc. Technol. 2019 (4) (2019) 292–310.

[29] A. Morichetta, M. Mellia, LENTA: Longitudinal exploration for network traffic analysis from passive data, IEEE Trans. Netw. Serv. Manag. 16 (3) (2019) 814–827, http://dx.doi.org/10.1109/TNSM.2019.2927409.

[30] M. Shafiq, X. Yu, A.A. Laghari, L. Yao, N.K. Karn, F. Abdessamia, Network traffic classification techniques and comparative analysis using machine learning algorithms, in: Proceedings of the 2016 2nd IEEE International Conference on Computer and Communications, ICCC 2016 - Proceedings, IEEE, 2017, pp. 2451–2455, http://dx.doi.org/10.1109/CompComm.2016.7925139.

[31] N. Hubballi, M. Swarnkar, M. Conti, BitProb: Probabilistic Bit Signatures for Accurate Application Identification, IEEE Trans. Netw. Serv. Manag. 17 (3) (2020) 1730–1741, http://dx.doi.org/10.1109/TNSM.2020.2999856.
[32] S. Rezaei, B. Kroencke, X. Liu, Large-scale mobile app identification using deep learning, IEEE Access 8 (2020) 348–362, http://dx.doi.org/10.1109/ACCESS.2019.2962018.
[33] X. Wang, S. Chen, J. Su, Automatic mobile app identification from encrypted traffic with hybrid neural networks, IEEE Access 8 (2020) 182065–182077, http://dx.doi.org/10.1109/ACCESS.2020.3029190.
[34] T. Shapira, Y. Shavitt, FlowPic: Encrypted internet traffic classification is as easy as image recognition, in: Proceedings of the 2019 IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS 2019, IEEE, 2019, pp. 680–687, http://dx.doi.org/10.1109/INFCOMW.2019.8845315.
[35] M. Wang, K. Zheng, D. Luo, Y. Yang, X. Wang, An encrypted traffic classification framework based on convolutional neural networks and stacked autoencoders, in: Proceedings of the 2020 IEEE 6th International Conference on Computer and Communications, ICCC 2020, IEEE, 2020, pp. 634–641, http://dx.doi.org/10.1109/ICCC51575.2020.9344978.
[36] B. Sun, W. Yang, M. Yan, D. Wu, Y. Zhu, Z. Bai, An encrypted traffic classification method combining graph convolutional network and autoencoder, in: Proceedings of the 2020 IEEE 39th International Performance Computing and Communications Conference, IPCCC 2020, IEEE, 2020, pp. 1–8, http://dx.doi.org/10.1109/IPCCC50635.2020.9391542.
[37] R. Moreira, L.F. Rodrigues, P.F. Rosa, R.L. Aguiar, F.D.O. Silva, Packet Vision: A convolutional neural network approach for network traffic classification, in: Proceedings of the 2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images, SIBGRAPI 2020, IEEE, 2020, pp. 256–263, http://dx.doi.org/10.1109/SIBGRAPI51738.2020.00042.
[38] G. Aceto, D. Ciuonzo, A. Montieri, A. Pescapé, Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges, IEEE Trans. Netw. Serv. Manag. 16 (2) (2019) 445–458, http://dx.doi.org/10.1109/TNSM.2019.2899085.
[39] C. Freksen, L. Kamma, K.G. Larsen, Fully understanding the hashing trick, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS '18, Curran Associates Inc., Red Hook, NY, USA, 2018, pp. 5394—5404.
[40] T.Q. Nguyen, J. Salazar, Transformers without tears: Improving the normalization of self-attention, in: 16th International Workshop on Spoken Language Translation 2019, Zenodo, 2019, http://dx.doi.org/10.5281/zenodo.3525484.
[41] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, X. Tang, Residual attention network for image classification, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2017, pp. 6450–6458, http://dx.doi.org/10.1109/CVPR.2017.683.
[42] J. Fu, J. Liu, H. Tian, Y. Li, Y. Bao, Z. Fang, H. Lu, Dual attention network for scene segmentation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2019, pp. 3141–3149, http://dx.doi.org/10.1109/CVPR.2019.00326.
[43] S. Zeng, F. Graf, C. Hofer, R. Kwitt, Topological attention for time series forecasting, in: A. Beygelzimer, Y. Dauphin, P. Liang, J.W. Vaughan (Eds.), Advances in Neural Information Processing Systems, 2021, pp. 1–12.
[44] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, URL https://www.tensorflow.org/, Software available from tensorflow.org.
[45] F. Chollet, Keras, 2015, URL https://keras.io/.
[46] V. Nair, G.E. Hinton, Rectified linear units improve restricted Boltzmann machines, in: Proceedings of the 27th International Conference on Machine Learning, ICML-10, 2010, pp. 807–814.
[47] D.P. Kingma, J. Ba, Adam: A Method for Stochastic Optimization, in: Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track, 2015, pp. 1–13.
[48] K. Ruth, A. Fass, J. Azose, M. Pearson, E. Thomas, C. Sadowski, Z. Durumeric, A world wide view of browsing the World Wide Web, in: Proceedings of the 22nd ACM Internet Measurement Conference, IMC '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 317–336, http://dx.doi.org/10.1145/3517745.3561418.
[49] J.G. Webster, S.-F. Lin, The internet audience: Web use as mass behavior, J. Broadcast. Electron. Media 46 (1) (2002) 1–12, http://dx.doi.org/10.1207/s15506878jobem4601_1.
[50] J.L. García-Dorado, J.A. Hernández, J. Aracil, J.E. López de Vergara, F.J. Monserrat, E. Robles, T.P. de Miguel, On the duration and spatial characteristics of internet traffic measurement experiments, IEEE Commun. Mag. 46 (11) (2008) 148–155, http://dx.doi.org/10.1109/MCOM.2008.4689258.
[51] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, in: Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, Vol. 27, Curran Associates, Inc., 2014, pp. 1–9.
[52] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, B. Ommer, High-resolution image synthesis with latent diffusion models, 2021, arXiv:2112.10752.
[53] A. Grammatas, Noiszy: A browser plugin which generates meaningless web-traffic to disguise your real browsing data, 2022, https://noiszy.com. (Accessed 07 March 2022).

Daniel PERDICES is researcher and teaching assistant at Universidad Autónoma de Madrid (Spain). He holds an FPU research grant by the Spanish Ministry of Science, Innovation and Universities. Previously, he was an R&D engineer at Naudit HPCN. He received the B.Sc. (Hons) degrees in Mathematics and in Computer Science (2018), the M.Sc. in Mathematics (2019) and the M.Sc. in Information and Communications Technologies (2020) and is currently a Ph.D. candidate, all at Universidad Autónoma de Madrid (Spain). He was a visiting scholar for three months in 2022 at SmartData@PoliTO, Politecnico di Torino (Italy). He researches on statistics, mathematical modeling, machine learning, network traffic analysis and SDN.

Jorge E. LÓPEZ DE VERGARA is associate professor at Universidad Autónoma de Madrid (Spain) since 2007 and is a partner of Naudit HPCN, which is a spin-off company that was founded in 2009 and is devoted to high-performance traffic monitoring and analysis. He received his M.Sc. and Ph.D. degrees in Telecommunication Engineering from Universidad Politécnica de Madrid (Spain) in 1998 and 2003, respectively, where he also held an FPU-MEC research grant. During his Ph.D., he stayed for 6 months in 2000 at HP Labs in Bristol. He studies network and service management and monitoring, and has coauthored more than 100 scientific papers on topics related to this field.

Iván GONZÁLEZ received his M.Sc. degree in Computer Engineering in 2000 and his Ph.D. in Computer Engineering in 2006, both from UAM, Spain. From October 2002 to October 2006, he was a teaching assistant at the Computer Engineering Department of UAM. From November 2006 to January 2008, he was a postdoctoral research scientist at the High-Performance Computing Laboratory (HPCL), Electrical & Computer Engineering Department, George Washington University (Washington, DC). He was a faculty member of the NSF Center of High-Performance Reconfigurable Computing (CHREC) at George Washington University. He is currently associate professor at UAM, where he is teaching computer-architecture-related courses. He is also partner of Naudit HPCN. His main research interests are heterogeneous computing (with GPUs, FPGAs, etc.), parallel algorithms, and performance tuning. Other interests include big data, machine learning and data analytics.

Luis DE PEDRO is associate professor at Universidad Autónoma de Madrid (Spain), and president of Naudit HPCN, a company devoted to high performance traffic monitoring and analysis. He received his M.Sc. and Ph.D. degrees in Telecommunication Engineering from Universidad Politécnica de Madrid (Spain) in 1987 and 1992, respectively. He currently researches on statistical models for network traffic.