Understanding Web Fingerprinting with a Protocol-Centric Approach

Bogdan Cebere
bogdan.cebere@cispa.de
CISPA Helmholtz Center for Information Security
Germany

Christian Rossow
rossow@cispa.de
CISPA Helmholtz Center for Information Security
Germany

ABSTRACT

Recent breakthroughs in machine learning (ML) have unleashed several approaches to fingerprinting web traffic based on traffic analysis. In particular, researchers report impressive classification performances by modeling HTTPS traces using packet metadata. Recent works focus mainly on the packet burst metadata (packet lengths, counts, and directions). The fact that burst metadata characterizes web traces is not surprising per se. Then again, most works stop at providing evaluation results and do not question the *reasons* for the success in qualitative analyses or ablation studies.

In this paper, we try to better understand why and when burst-based web fingerprinting works. To this end, we follow *a protocol-centric approach* —instead of promoting yet another classification approach—that seeks to investigate the impact of the underlying protocols on web fingerprinting. We study several research questions based on typical domain and page classification datasets. Most importantly, we show where the classification gain comes from, i.e., which messages or flows are particularly valuable. In contrast to recent works, we show that the beginning of communication does not always leak valuable fingerprinting information. This knowledge allows the design of targeted and, thus, more efficient fingerprinting attacks and defenses. In addition, we study how data availability (number of labels) and HTTP protocol features (e.g., caching, user agents) might skew the classification results. We hope that future research can profit from this analysis, which complements existing fingerprinting approaches, by better understanding fingerprinting methods and respective countermeasures.

CCS CONCEPTS

• **Computing methodologies → Machine learning**; • **Security and privacy → Pseudonymity, anonymity and untraceability**; **Web protocol security**.

KEYWORDS

Machine Learning, Web Fingerprinting, Encrypted Traffic Analysis, Protocol-Centric Fingerprinting

ACM Reference Format:
Bogdan Cebere and Christian Rossow. 2024. Understanding Web Fingerprinting with a Protocol-Centric Approach. In *The 27th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2024), September 30–October 02, 2024, Padua, Italy*. ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3678890.3678910

1 INTRODUCTION

Recent studies (e.g., [6, 26, 70, 84, 85, 95, 100] and several others) show that middleboxes can robustly fingerprint HTTPS-encrypted communication based on payload metadata, without decrypting the traffic. Such web fingerprinting techniques allow for various attacks, such as domain classification (e.g., revealing which domain a client talked to), page classification (e.g., identifying which subpage of a given domain a client visited), and other types of classification (e.g., malware traffic detection).

All approaches share the common feature of feeding payload metadata (length, timing, direction, etc.) sequences of TLS-encrypted flows to a classification model. Recent works focused mainly on *packet burst information*, i.e., the direction of communication, the number of packets per direction, and packet lengths. Another source of entropy is the packet timing information, but the major downside is the bias towards the training network ([100]). The approaches then mainly vary in implementation details, such as their basis for length sequences (e.g., sizes of raw TCP segment sizes vs. reassembled TLS payloads) or their chosen classifier (e.g., linear classifiers vs. deep learning approaches). However, regardless of the design choices, they perform fingerprinting based on sequences of the packet burst features from TLS-encrypted communication.

Previous empirical studies consistently report high HTTPS fingerprinting accuracy in various tasks, thus defying the encryption layer's privacy guarantees. Such a side channel is a faster alternative to man-in-the-middle attacks, as it works on encrypted data directly. However, these fingerprinting models rarely generalize outside the laboratory conditions. The fingerprinting metadata has a significant risk of collisions between web pages, and any minor update in the downloaded content can lead to misclassification.

Due to these limitations, a challenging question is understanding *why* fingerprinting is effective in controlled setups and *how* to use the insights in real-life attacks and defenses. To that end, various ML interpretability methods have emerged ([65]), spotlighting the importance of features for the prediction task. Yet, for traffic analysis tasks, a more intriguing insight comes from correlating the ML benchmarks with the network protocol's specifications. Concretely, by blindly trusting the given approaches to identify characteristic patterns, we do not know *which part of the TLS communication is significant* to the classifier.

Is the TLS handshake or the encrypted payloads' metadata leaking more fingerprinting information? This unawareness implies several important aspects of high practical relevance. First, we lack an understanding of how we could effectively improve the privacy of TLS-protected communication. Second, most related research is conducted on whole-stream data, potentially relying on characteristic patterns at the end of a TLS flow. This bloats the state of classifiers and causes them to model long flows' whole message length sequences. It also *delays* detection to a point where the communication already happened, which is "too late" for many application scenarios such as intrusion prevention systems or domain censoring. Third, not knowing the reasons for classifiers' success increases the risk that we overestimate the power of fingerprinting. In contrast, if we were able to reveal which phases of HTTPS communication are key to classification, we could judge how robust a classifier is with respect to changes, such as client - or server-side defenses.

In this paper, we take a step back and shed light on which phases of an HTTPS stream contribute to the success of fingerprinting. To this end, we study a *protocol-centric framework* using nine research questions (**RQ**), aiming to explain the fingerprinting results from the TLS and HTTP perspectives. This approach is consistent with the recent trends in ML research, where *data-centric* methods have been intensively studied [74, 82, 99]. We first study the setting of domain fingerprinting in **RQ1** and examine the importance of individual messages within HTTPS flows. We show that the domain classification power heavily relies on only three parts of the HTTPS flow: (1) the server-side TLS handshake, including the certificate, (2) the *first* client request, and (3) the *first* server response. While subsequent messages *slightly* improve the domain fingerprinting, the classifier lacks sufficient entropy without these three initial messages. This provides important insights into which parts of the communication we must focus on when designing fingerprinting defenses, which we investigate in **RQ2**. We further highlight some potential pitfalls in the data collection and the benchmarking process by investigating the impact of HTTP caching on fingerprinting (**RQ3**) and the impact of the dataset diversity on the benchmarking results (**RQ4**). We then extend our analyses to an even more challenging setting in Section 4, namely to page classification. Here, attackers aim to learn which domain subpage a client visits (**RQ5**). As expected, in this setting, the server-side TLS handshake is insignificant, and the HTTP layer carries the most entropy. We find that the classification accuracy benefits significantly from (1) focusing on "characteristic" flows instead of just the first flow (**RQ6**) and (2) combining multiple flows for the classification task (**RQ7**). In particular, we find that static resources fetched in particular flows often boost classification performance, while other flows do not add much entropy. This highlights that defenses do not necessarily have to focus on *all* streams but can be reduced to HTTPS flows carrying most of the classification entropy to minimize overhead (**RQ8**). Finally, we investigate the impact of the HTTP user-agent header on page classification (**RQ9**).

In summary, we provide the following contributions:

- We highlight a set of fingerprinting interpretability techniques from a protocol-centric perspective - focusing on the TLS and HTTP mechanisms- rather than hiding them under a simple numerical dataset.
- We analyze in depth which parts of the HTTPS communication enable domain fingerprinting and page fingerprinting. We show the key differences between domain and page fingerprinting and how the sources of information vary from one setting to the other. In contrast to recent work, we show that the beginning of communication is not always the most significant source of fingerprinting information.
- We show the side effects of relaxing the empirical setup. In particular, we show that results from experiments with just a few hundred labels do not necessarily generalize to realistic setups.
- We measure how some HTTP features - like HTTP caching and user-agent changes - can affect the fingerprinting process, and we discuss how to circumvent these limitations.

These insights help to better understand the underlying key drivers for fingerprinting. We see our work as an attempt to complement the current technological advances in fingerprinting, which apply more complex models on ever-increasing datasets for this task. We argue that despite all the advantages of abstracting away from the question of "*Why does it work?*", it is of utmost time to reflect on what essentially enables website fingerprinting.

2 SETTING, BACKGROUND AND RELATED WORK

We focus on fingerprinting of HTTPS communication to infer the domain or page a user visits. Web fingerprinting can pose a significant privacy concern. It allows user tracking and mass surveillance and can ultimately be used to block or censor communication. We describe our threat model in Section 2.1, provide background information in Section 2.2, and survey web fingerprinting approaches in Section 2.3.

2.1 Threat model

We will study a setting where an active or passive attacker (e.g., middlebox) can intercept encrypted HTTPS traffic without the ability to decrypt it. The attacker aims to use traffic analysis to infer which domain, or subpage of a given domain, a client visits over HTTPS. We refer to these goals as *domain fingerprinting* and *page fingerprinting*, respectively.

Due to the security guarantees of HTTPS/TLS, the only information available to the attacker is network metadata. The attacker has no local information (like CPU counters and memory usage) about the client or server. We assume the analyzed traffic comes from standard browsers without custom defenses in the session encryption. Some browsers add padding to the client handshake packets (e.g., Chrome). We focus on the most recent TLS version, v1.3 [78, 79]. We restrict the attacker's access to HTTPS flows only and ignore information they could infer from encrypted DNS [10, 87].

2.2 Background

Type of features in HTTPS and feature modeling: TLS offers solid guarantees for the security of the HTTP application layer. Assuming that the parameters of the TLS communication (Server Name Indication, TLS extensions, server certificates) are encrypted

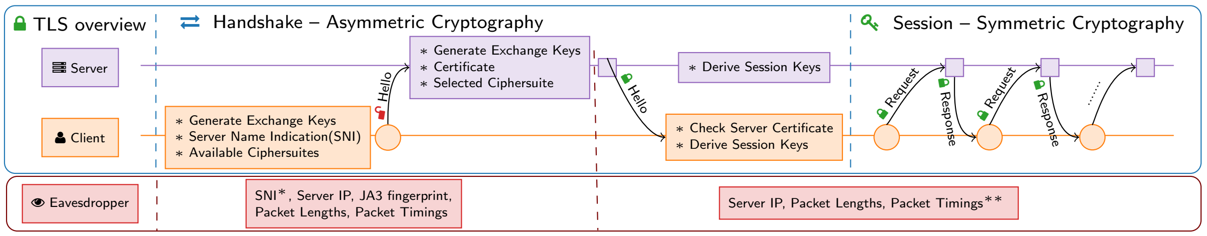
TLS overview
Handshake – Asymmetric Cryptography
Session – Symmetric Cryptography
Server
Client
Eavesdropper
∗ Generate Exchange Keys
∗ Certificate
∗ Selected Ciphersuite
∗ Generate Exchange Keys
∗ Server Name Indication(SNI)
∗ Available Ciphersuites
Hello
Hello
∗ Derive Session Keys
∗ Check Server Certificate
∗ Derive Session Keys
Request
Response
Request
Response
SNI*, Server IP, JA3 fingerprint, Packet Lengths, Packet Timings
Server IP, Packet Lengths, Packet Timings**

Figure 1: *High-level overview of a TLS 1.3 session.* The Client (orange) and Server (violet) agree on the session keys to secure subsequent HTTPS requests/responses. The Eavesdropper (red) indicates which information can be leaked from each phase. *SNI is assumed to be encrypted (ECH). **The Certificate is encrypted since TLS 1.3.

- hence invisible to eavesdroppers - the only available sources of information are the underlying TCP/IP layers and application-layer meta information - e.g., packet lengths, timings, or bursts.

Web fingerprinting, thus, usually relies on two types of information: (1) network-specific information, which is highly correlated to the quality of the local network, such as packet timings and fragmentation, and (2) network-agnostic information, which is not affected by the congestion of the local network (e.g., message lengths, directions, or bursts). Network-specific information can be effective against defenses like packet length padding. However, it is tightly bound to the local system and network conditions, and a generalization across multiple networks is unknown. Due to these limitations, recent works [26, 100] ignore the timing information and focus on network-agnostic information – TCP flow identifiers (source and destination IP and port), reassembled packet length, and TCP flow sequences. This information can be characteristic since the application data layer dictates the ciphertext lengths. Server IP information can help crack the domain classification problem when there is a direct mapping to the contacted domain. However, in practice, the IP information can be affected by (1) geolocation - the same domain can have different IPs in different regions, (2) cloud load-balancing - the cloud vendors can optimize the resource usage by moving or deleting containers allocated to a domain, or (3) same IP multi-domain tenancy, when multiple domains with different certificates share the same public IP. Given these limitations, we only use the server IP information to group the TCP flows and focus on the payload length.

Recent works focus on two fingerprinting learning strategies. On the one hand, attackers can collect statistical information from the bursts - e.g., packet length mean, variance [16, 20, 38, 54, 64, 84]. Such statistics may provide a good starting point for prediction tasks [2] but might miss subtle burst particularities. On the other hand, attackers can organize the burst metadata as time series [14, 22, 23, 52, 53, 56, 83, 85, 93, 100, 101]. Concretely, the dataset is a $3D$ tensor of shape $(N_{Flows}, N_{Packets}, N_{Features})$, where N_{Flows} denotes the number of flows to be fingerprinted, $N_{Packets}$ is the number of packets selected per flow (or observation points), and $N_{Features}$ is the number of packet metadata selected from each packet. In both formats, the attacker has to accumulate N_{Packet} packets for each flow before being able to run the fingerprinting model. All these models try to learn a predictive task based on the feature sequences and output the probability for a label (domain or page).

Semantics of Message Positions: We focus on the time series approach, which allows us to interpret the classification results from a protocol-centric perspective. For our analyses, we leverage the idea that individual packet positions within a TCP stream have an associated semantic meaning. Without losing generality, assume we observe an HTTPS flow under TLS 1.3 using 1-Round-Trip-Time (1-RTT). Such a flow always follows the same semantics. In particular, the first consecutive TCP payload (P_0) contains the client-side TLS handshake, the second (P_1) the server-side TLS handshake (including certificate), and P_n P_{n+1} with $n \in \{2, 4, 6, \dots\}$ are request/response pairs.

Figure 1 illustrates the stages of a TLS session, the exchanged information, and the series of application-layer requests in more detail. The handshake consists of two messages between the client and the server to verify their identities and derive the session keys. In the Client Hello step, the client announces a set of cryptographic capabilities (cipher suites) and demands communication with a specific server (in the server name indication (SNI) extension). Traffic analysis techniques may target this section since it contains plaintext information about the server (the SNI) and the client (the TLS extensions or the TLS cipher suites used for creating client fingerprints [18]). However, there are considerable efforts to close these windows of attacks: The clients might randomize their TLS extensions to prevent JA3 fingerprinting attacks [3], and Encrypted Client Hello (ECH) reestablishes confidentiality of the SNI field [79]. The server replies with its cryptographic capabilities and a certificate, which includes the hostname and a signed public key. We focus on TLS 1.3, where the certificate is encrypted.

After the handshake, the client and server speak over encrypted HTTP, powered by an AEAD (Authenticated Encryption with associated data) symmetric cipher, such as AES-GCM (Advanced Encryption Standard Galois/Counter Mode) [81] or ChaCha20-Poly1305 [69]. AES-GCM is an efficient block cipher with dedicated hardware support, especially in x86 architectures. ChaCha20-Poly1305 is a stream-cipher-based encryption mode that provides similar security properties and is often preferred on devices without AES hardware support. Both methods return a ciphertext of the same length as the plaintext. While this could be fixed with padding (which we will also study in this work), by default, it leaves a window of attack for fingerprinting.

2.3 Related work

Table 1 highlights some key aspects of the current work compared to prior findings. The main insight is that the fingerprinting analysis

Technique	Examples	Clf. Type	# Labels	Flow Length	Multi-Flow	Interpretability
DNS-based	[10]	Domain	>1000	*unk*[*]	Yes	No
Raw IP Frames	[100]	Domain	<200	100	No	No
Raw TCP Payloads	[53]	Domain	<200	*unk*	No	No
Packet Timing-based	[8]	User ID	<1000	*unk*	No	No
Packet length-based	[64, 84]	Page	<200	100	No	No
CDN Resource-aware	[26, 27, 95]	Page	>1000	# bursts	Yes	No
ML-Interpretable methods	[23]	Domain	<200	# bursts	No	Partial[**]
Protocol-centric methods	**(Ours)**	**Domain & Pages**	>1000	# bursts	**Yes**	**Yes**[***]

Table 1: *Overview of existing web fingerprinting works, grouped by their techniques and empirical parameters.* One critical aspect of the current work is to explain the fingerprinting results using a protocol-centric framework. [*] Number of collected payloads per flow is *unknown / not reported.* [**] Method specific to Convolutional networks. [***] Protocol-centric interpretability method.

should be conducted by treating the underlying protocols (TLS and HTTP here) as first-class citizens and not hiding them behind a numerical problem. This framing allows us to correlate the finding with aspects of the protocols' rules – e.g., how much information does the server certificate metadata leak, and for which tasks? – and provide practical defenses. Another highlight of this work is that the empirical setup, especially the total number of labels (pages, domains) and the data collection window, is critical for an insightful conclusion. Any relaxation in the setup can have severe consequences for the quality of the results, as we will show in the experimental section.

Encrypted traffic analysis. A plethora of encrypted traffic analysis techniques have been developed for various purposes: malware detection [6, 7, 32, 33, 52, 53, 101], client identification [19, 20, 22, 42], and website detection [14, 16, 20, 22, 23, 52–54, 56, 64, 83–85, 93, 100, 101]. The problem is also studied for other encrypted protocols, such as Tor networks [46, 68, 70, 89, 90], QUIC [92, 103], DNS-over-TLS [10, 41, 87], or VPN [29, 44, 57]. Several strategies for leaking information out of the encrypted traffic are available. One of the earliest sources of information is the *TLS handshake*, based on the server name indication (SNI), certificate, the list of TLS extensions, cipher suites, or JA3 fingerprints [18, 22, 42]. While deterministic, this attack window is getting narrower with each update on the TLS protocol [78, 79]. Another thread of research is based on the inter-arrival times of the packets [8, 10, 29, 68, 87]. A popular source of information is based on the packet lengths [6, 14, 55, 56, 64, 70, 84, 85, 102]. Modern neural network architectures even allow for raw traffic processing [23, 33, 52, 53, 57, 68, 83, 89, 90, 92]. On the modelling side, popular models are Gaussian Distributions [16, 64], linear models [20], random forests [54], clustering methods [38, 84], and neural networks [14, 22, 23, 52, 53, 56, 83, 85, 93, 100, 101]. While most of the research is done on single TCP flows, specific to website fingerprinting, some papers [26, 27, 95] also analyze the impact of fingerprinting using page content like images.

Defense techniques. Various defensive techniques have also been investigated in response to fingerprinting techniques. Popular defensive techniques against fingerprinting attacks include padding techniques ([4, 5, 36, 45, 47, 63, 75, 91]), fixed-rate traffic ([11–13, 58]), traffic morphing ([5, 17, 77, 91, 96, 97]), chaff traffic ([4, 5, 17, 36, 45, 61, 75, 91]), adversarial perturbations ([67, 77]) and multihoming/traffic splitting ([24, 40]). Other works focus on the defense at the application layer by employing half-duplex HTTP ([97]), HTTP pipelining ([61]), Web objects morphing ([17]), or the HTTP Range header ([60]). These defenses are generic, and most have a known attack to bypass them. In our work, we do not aim for a novel defense technique but rather intend to provide a source of transparency for which sections of traffic leak information. We apply a basic padding defense to test the critical windows of the prediction support hypothesis, and we do not attempt to cover all other channels of attacks (such as timing-based attacks).

Some papers formalize the problem of web fingerprinting defenses from an information-theoretic point of view ([51, 62]). Namely, they measure the bits of information leakage on Tor traffic. One of the highlights is that the burst information (including packet statistics) leaks the most information about the visited pages, regardless of whether any defense is employed. While the timing information can be used for fingerprinting in closed-world setups, the feature is biased towards the training network and does not generalize well to other setups ([100]).

Some related work highlights that the beginning of the communication usually carries most of the fingerprinting entropy ([23, 36]). While this conclusion is true for domain fingerprinting tasks, our protocol-centric analysis identifies other critical windows depending on the task at hand - domain or page classification.

Empirical Setup. When analyzing network data, there is a compromise between practical usability and empirical added value. Unfortunately, the discussion around the experimental parameters is often omitted. Various works are inconsistent in the number of labels in experiments, making it difficult to compare related work. Some methods are evaluated on datasets with fewer than 200 labels ([23, 53, 56, 64, 83, 84, 100]), 200 – 1000 labels ([102]), or more than 1000 labels ([10, 26, 27]). In this work, we study the impact of such different empirical strategies to better understand how to compare.

Machine Learning explainability. Interpretability methods for machine learning focus on explaining the importance of each feature in the models' output. Standard techniques include learning the weights of a linear model, example-based interpretability methods [48], global model-agnostic methods [98, 104], instance model-agnostic methods [59, 80], and neural-network-specific methods [49, 88]. We follow the model-agnostic line of thought and rely on the best-performing downstream model to correlate with TLS stages. This approach is popular in other machine learning subdomains, like missing data imputation [43, 66], where the imputation quality is reported using the performance of a downstream model.

In contrast to standard ML work, we try to interpret the findings from a protocol-centric perspective, namely the TLS and HTTP protocols.

3 HTTPS DOMAIN FINGERPRINTING

Seeing the sheer volume of prior works that used ML models for web fingerprinting, we now take an orthogonal approach and try to understand when and why fingerprinting is successful. The overarching question of our protocol-centric approach is spotting where the entropy for classification comes from. To this end, we first model HTTPS traffic similar to related work (Section 3.1) and employ datasets also used by prior work (Section 3.2), thereby intentionally approximating typical settings in which fingerprinting approaches have been evaluated. Recall that our focus is not on providing new methodologies; instead, we would like to *explore the reasons* for fingerprinting success.

In this section, we explore several research questions on the effectiveness of domain fingerprinting (we cover *page* fingerprinting in Section 4). We first study if the position of HTTPS payloads influences the domain classification model's entropy (Section 3.3) and how these insights can be used for targeted defenses (Section 3.4). Finally, we study the impact of HTTP caching (Section 3.5) and of evaluating too small datasets (Section 3.6).

3.1 Modeling HTTPS Traffic for Fingerprinting

Web fingerprinting aims to map an HTTPS flow, or a set of HTTPS flows, to a label (e.g., domain or page name). Following the methodology of previous studies [24, 28, 71], we model an HTTP flow as a sequence of reassembled TCP payloads accumulated until the direction of communication changes (bursts). Concretely, for a flow $F = (P_0, P_1, \dots)$, the subset of payloads $(P_0, P_2 \dots, P_{2k}, \dots)$ consists of client $\longrightarrow$ server requests, and the subset $(P_1, P_3 \dots, P_{2k+1}, \dots)$ contains the server $\longrightarrow$ client responses. P_0 is equivalent to the TLS Client Hello request, and P_1 includes the Server Hello and Certificate. For each payload P_i in a flow F, the attacker extracts a set of observable features $f(P_i) = (f_{i1}, f_{i2}, \dots, f_{iN_{feats}})$. For a flow F, $f(F) = (f(P_0), f(P_1), \dots)$ denotes the observed features from all the payloads in the flow. For our experimental section, the feature set consists of the request's payload length and direction: 1 for a request and -1 for a reply. We consider the *length of a payload* the difference between the maximum and minimum TCP sequence number until the flow direction changes or the flow ends. This enables us to analyze the traffic without handling the overhead of TCP reassembly and retransmission. Concretely,

$$f(P_i) = \begin{cases} (\text{length}(P_i), 1), & \text{if } P_i \text{ request}, i = 2k \\ (\text{length}(P_i), -1), & \text{if } P_i \text{ response}, i = 2k+1 \end{cases}$$

If any flow in **F** has less than $N_{payloads\_per\_flow}$ payloads, a dummy value $*$ is used at training and inference time to mark the absence. If any flow in **F** has more than $N_{payloads\_per\_flow}$ payloads, the flow is cropped to $N_{payloads\_per\_flow}$ payloads.

Given a list of HTTPS flows $\mathbf{F} = (F_1, F_2, \dots, F_{N_{flows}})$, the eavesdropper seeks to extract their features and create a $3D$ tensor of shape $(N_{flows}, N_{payloads\_per\_flow}, N_{feats})$. The attacker can train a classifier M to solve a specific task based on this tensor. The architectures included in benchmarks include logistic regression [21], k-nearest neighbors [72], random forests [9], XGBoost [15], multi-layer perceptron [39], RNN/GRU/LSTM [34], CNN [50], and Transformers [94]. Regardless of the architecture, the learned output $M(F)$ is the probabilities of a set of labels.

3.2 Experimental Setup

Dataset	**Clf. Type**	**# Samples**	**# Labels**
Tranco [73]	domain (§ 3)	129,440	6,472
Wikipedia [31]	page (§ 4)	101,850	1,455
9GAG [37]	page (§ 4)	139,560	2,326
IMDB [25]	page (§ 4)	111,440	1,592

Table 2: *Datasets for Evaluation.* We demonstrate the attacks and defenses using the Tranco dataset for domain classification. We use the Wikipedia, 9GAG, and IMDB datasets for the page classification experiments.

Datasets. We employ four closed-world datasets, described in Table 2. Each dataset has relevant particularities for our analysis: (1) The Tranco domain dataset is a well-established list for research purposes and supports our domain classification experiments, where the focus is closer to the beginning of communication; the rest of the datasets are used for page classification, which we will discuss in Section 4; (2) The Wikipedia dataset is a lighter page classification task, having a small network footprint, but irregular URL lengths; and (3-4) The 9GAG and IMDB datasets are page classification tasks, rich in additional flows (for images, videos, etc.), and with constant URL lengths. These datasets are not representative Internet models, but they allow the modeling of various scenarios to illustrate a set of attacks and defenses for encrypted traffic fingerprinting. The data collection process is described in Appendix A.

Evaluation. We work in a model-agnostic setup. To that end, we benchmark each classification model - Logistic Regression, KNN, Random Forest, XGBoost, MLP, LSTM, CNN, Transformer - for each detection task using 5-fold cross-validation and select the highest score. To get the best insights from the data, we model each test as One-vs-Rest (OvR). This allows us to a fine-grained debugging and to identify anomalies in the datasets. However, from this perspective, we always work with imbalanced datasets and must be careful about the reported metrics. The details of the evaluation metrics can be found in Appendix B.

3.3 Protocol-Centric Communication Analysis

Several recent HTTPS fingerprinting works demonstrated high classification accuracy, with a similar methodology to ours (see Section 3.1). Interestingly, none of the prior studies have investigated which parts of the communication the classification entropy stems from. Thus, the exact reasons for the reported high classification accuracies remain unknown. Prior work like [23] attempted to understand the importance of features inside neural networks. In contrast, we want to create a model-agnostic method for identifying the regions of traffic that can generate a fingerprint.

The importance of our protocol-centric approach is threefold: (1) An attacker can optimize the data collection process by focusing on the exact source of entropy in the observed network traffic (boosting both efficiency and effectiveness), (2) We gain an understanding

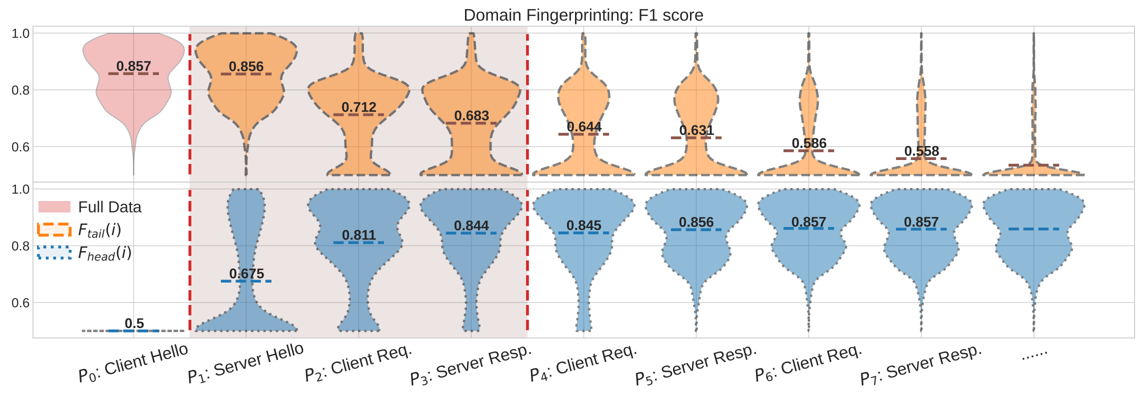

Figure 2: *Detection of the Critical Window of prediction support for Domain fingerprinting (6472 domains) as violin plot.* Each violin plot shows the distribution of F1 scores (y-axis) of all tested domains, differentiated by which HTTPS payloads are included (x-axis). We highlight two perspectives: *a)* Mask before P_i: At $F_{\text{tail}}(i)$, we classify only the payloads starting with index i (orange, dashed border, top) – i.e., the effect of excluding payloads $P_0 - P_{i-1}$. The upper lane also includes the performance on the entire flow (red, solid border, first on the left); and *b)* Mask after P_i: At $F_{\text{head}}(i)$, we consider only the first i payloads (blue, dotted border, bottom), i.e., the effects of including only payloads $P_0 - P_i$ for fingerprinting. The dashed horizontal line is the mean of the scores of each violin next to the numerical value of it. The critical window for prediction support (Definition 3.1) is located between red dashed vertical bars and with a gray overlay.

of the stages of communication at which classification becomes possible (e.g., to enable proactive blocking), and (3) Finding the characteristic communication phases can aid us to design optimized defenses, e.g., adding padding to the "risky" communication window only. We thus ask the following research question:

RQ1: Which phases of web communication carry characteristic patterns that allow for domain fingerprinting?

To answer this question, we study which windows of payload positions in a flow carry entropy. In stark contrast to the data analytical approach, we argue that the source of prediction power lies in a few outstanding payloads rather than the entire flow.

Definition 3.1. ***Critical window for prediction support.*** Given any flow of interest $F = (P_0, P_1, \ldots, P_N)$, we aim to identify the *minimal interval* of indices $[I, J] \subseteq [0, N]$ - denoted the critical window for prediction support - so that $F' = (P_I, P_{I+1}, \ldots, P_J)$ and for classifier M, the performance of $M(F')$ does not significantly differ to the one of $M(F)$ – i.e., the F1 score of the model M satisfies $|Score(M(F)) - Score(M(F'))| < 0.02$.

To define the payload inclusion criteria in the critical window of prediction support, we introduce two new models:

- *Mask traffic* ***before*** *a payload:* Evaluate while masking flow payloads *before* index s: $F_{\text{tail}}(s) = (P_s, P_{s+1}, \ldots, P_N)$. In this scenario, the predictive performance is based on the tail flow bursts – without the beginning of the communication up index $s - 1$.
- *Mask traffic* ***after*** *a payload:* Evaluate while masking flow payloads *after* index s: $F_{\text{head}}(s) = (P_0, P_1, \ldots, P_s)$. In this scenario, the predictive performance is based on each flow's first s bursts, starting with the TLS handshake and cropping session payloads at the flow tail.

Definition 3.2. ***Inclusion criteria for payloads in the critical window of prediction support.*** A payload P is included in the critical window of prediction support if, given a benchmarking metric – F1 score here – P contributes significantly to at least one modeling direction. Concretely, we determine the start of the critical window of prediction support (i.e., I_1) as the first packet whose omission significantly reduces the F1 score, as can be derived from the F_{tail} analysis. Likewise, we determine the end of the critical window to be the first packet whose inclusion in the F_{head} analysis fulfills the criteria defined in Definition 3.1.

The violin plot in Figure 2 illustrates the F_{tail} and F_{head} scenarios for domain fingerprinting using the F1 score metric. The x-axis represents the message in the communication, with the $P_0 - P_1$ illustrating the TLS handshake (Client Hello and Server Hello) and the $P_2 - \ldots$ representing the TLS session data. The violin for P_i encompasses the classification scores for detecting each domain while including payload P_i, with the horizontal dashed lines in each violin representing the mean score across all evaluations. We use three perspectives in analyzing the data: (1) no traffic masking, containing the complete flow, in the first red violin on the top-left, (2) $F_{\text{tail}}(i)$, masking all the traffic *before* payload i, using the orange violins with dashed border (top row), and (3) $F_{\text{head}}(i)$, masking all payloads *after* payload i, using the blue violins with dotted border (bottom). The critical window of prediction support is highlighted between the *red-dashed vertical lines, with a gray overlay.*

The violin plot is the first evidence of the benefits of the protocol-centric approach: We can visualize, debug, and identify the traffic section that leaks traffic information. For the domain fingerprinting problem, we can observe that a few payloads are instrumental in a high-quality prediction. P_0 (Client Hello) does not impact the prediction quality because, the Chrome browser, our data collection backend, pads this payload, making it uninformative. Using the steps in Definition 3.2, we decide that P_1 (Server Hello and Certificate) is the start of the critical window of prediction support, as its exclusion significantly degrades the fingerprinting performance in the F_{tail} analysis (orange violin). However, the certificate alone is not enough for domain fingerprinting, as it can be shared across multiple domains, creating thus metadata collisions. P_2 (Client Request) further improves the fingerprinting performance, but his inclusion is not enough to satisfy the criteria in Definition 3.1. Instead, P_3 (First server response) is the first payload whose inclusion in the critical window leads to an F1 fingerprinting score of 0.844, 0.013 lower than the performance using the entire flow. We conclude that the minimal interval that satisfies Definition 3.1 is $P_1 - P_3$, and therefore, the critical window for domain prediction support.

This nuanced analysis of the payload entropy allows us to assess the potential fingerprinting defenses depending on their focus. For example, seeing that masking P_1 does not entirely undermine domain fingerprinting, we conclude that defenses that only mask the server certificates (e.g., Encrypted Client Hello [79] combined with TLS handshake padding) are ineffective against domain fingerprinting. Instead, defenses *have* to hide the characteristic patterns within $P_1 - P_3$ to destroy classification performance.

In addition, Figure 13 illustrates the same analysis from the perspective of the AUPRC metric, and Appendix E investigates the critical window of prediction support from an information-theoretic point of view, using the WeFDE framework [51].

So far, the findings are on par with prior work ([23, 35]), which also identifies through other means that the beginning of the communication is critical for traffic fingerprinting.

Takeaways: Domain fingerprinting models rely on a specific communication window, denoted as the critical window of prediction support. Models will not get enough entropy outside that window to perform reasonably well. This indicates that (i) fingerprinting is possible more efficiently by recording just a few payload sizes per flow, (ii) fingerprinting can be done at an early point in time, and (iii) defenses must address the critical window of prediction support.

3.4 Targeted Fingerprinting Defenses

We validate our findings from the other side of the problem – the web fingerprinting defense. Concretely, we test the effects of applying a defense only on the critical window of prediction support. One of the popular defensive techniques against fingerprinting is traffic padding ([4, 5, 35, 45, 47, 63, 75, 91]).

The general padding defense has several known limitations, the main one being a significant traffic overhead. We intend to address this limitation as a protocol-centric traffic analysis use case. Our method is not designed to be scalable or general-purpose but to test whether a targeted padding strategy limits the attacker's possibilities.

Using our insights from **RQ1**, we thus wonder:

RQ2: Can we use the identified critical window of prediction support to defend against domain fingerprinting more efficiently?

For domain fingerprinting, the primary source of entropy lies in three payloads. We want to test if adding padding to any or all of these payloads affects an attacker. We design an experiment using an attacker who knows the padding strategy and augments their training dataset accordingly.

Padded Section	F1 Score	AUPRC
Server Cert.	0.718 ± 0.01	0.608 ± 0.01
Client Req.	0.799 ± 0.01	0.755 ± 0.01
Srv. Resp.	0.802 ± 0.01	0.756 ± 0.01
Cert + Req	0.711 ± 0.01	0.612 ± 0.01
Req. + Resp.	0.790 ± 0.01	0.741 ± 0.01
Cert. + Req. + Resp.	0.659 ± 0.01	0.585 ± 0.01
Entire flow	0.636 ± 0.01	0.545 ± 0.01
No padding	0.844 ± 0.01	0.812 ± 0.01

Table 3: *Padding Effects on the Critical Window of Prediction Support.* The testing scenario uses an attacker who knows the padding strategy and augments the training dataset with padding accordingly. The tests are conducted on 1000 domains, using a padding block of 64 bytes, using the same dataset as in Figure 2. The scores are reported as the *mean* ± *95% confidence interval.*

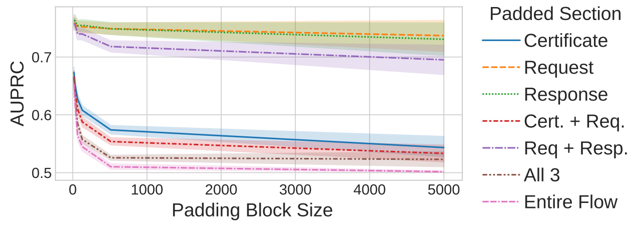

Figure 3: *Domain fingerprinting performance by the size of the padding block, using the AUPRC metric.* The critical window of prediction support is labeled using 'All 3': certificate, client request, and server response ($P_1 - P_3$).

Table 3 illustrates the effects of padding various payloads in communication using padding blocks of 64 bytes. By masking $P_1 - P_3$, the fingerprinting AUPRC - 0.585 - suggests that the attacker has a low chance of discriminating between domains. This score is significantly lower than when padding only single payloads or when padding is not used at all. For comparison, we benchmark the performance by padding the entire flow, which is considerably more costly. As expected, this leads to a slightly lower AUPRC (0.545) but not significantly lower than when padding only the 3-payload window (0.585). These findings confirm that a protocol-centric approach can also find valuable defense insights.

Finally, we investigate the impact of varying the padding block size between 16 and 5000 bytes. Figure 3 highlights the effect of the padding block size when applied to the critical window of prediction support (labeled 'All 3'), using the fingerprinting mean performance. The attacker might be able to handle a padding block of less than 64 bytes with careful training, but the low AUPRC score suggests a low chance of discrimination between domains. Similar to Table 3, the AUPRC score is slightly lower on the entire flow padding but at a higher communication overhead. Nevertheless, for padding blocks larger than 400 bytes, the AUPRC is ~ 0.5 for the critical window of prediction support, making the traffic impossible to fingerprint.

Takeaways: Learning the critical window of prediction support allows for targeted padding defenses against domain fingerprinting. Padding only the $P_1 - P_3$ traffic region forces

the attackers to confuse the domain labels, confirmed by an AUPRC prediction score of ~ 0.5.

3.5 Performance Impact of HTTP Caching

We showed that an attacker can adapt and efficiently fingerprint the communication depending on application-layer particularities. However, other factors such as the *HTTP caching* can impact web communication. This feature does not have a direct workaround at inference time and must be addressed during data collection and training.

The servers can instruct the client on how to cache a page in the HTTP response headers. The `Cache-Control` directive specifies how and for how long the client should cache a page. This can include the `no-store` header (forbids the client to cache the page), the `no-cache` header (instructs the client to ask the server about the cached content), or the `max-age` value (specifies the maximum time-to-live of the cache). In the case of the `no-cache` directive, the client can ask the server about the newer content using either a small token `ETag` or a time-based strategy with the `Last-Modified` header in the request. Irrespective of the chosen caching strategy, caching influences the request/response behavior within the HTTPS session. We thus ask:

RQ3: How can the HTTP Caching layer affect the domain fingerprinting quality?

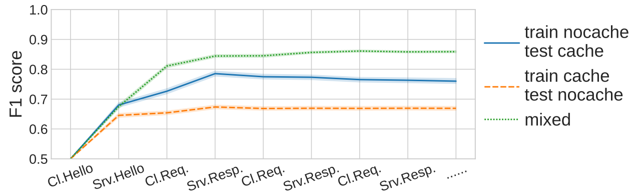

Figure 4: *HTTP(S) Caching impact on the Domain fingerprinting problem.* The scenarios are: (1) train without any HTTP(S) caching examples and test on caching samples only (blue solid line), (2) vice versa (orange dashed line), and (3) train and test on a mixture of data with/without caching samples (green dotted line).

Figure 4 shows the pitfalls of ignoring HTTP caching. Training a model only on HTTP cached data will decrease performance significantly if the model encounters non-cached HTTP data (orange scenario). This results from additional/modified HTTP request headers or potentially missing responses. In the reversed scenario, when we train on HTTP data without caching and test on cached data (blue line), the models can generalize better due to the availability of more data. However, they still perform below an F1 score of 0.7. A mixture of data (green line) is ideal among the scenarios, and the mean F1 score is 0.858.

We discuss other sources of performance degradation in Section 4.5 and Section 5.1.

Takeaways: Attackers can adapt to specific *HTTP* particularities and create an efficient fingerprint. However, some *HTTP* behaviors cannot be circumvented at data preprocessing and must be addressed at dataset collection. The data collection procedure must augment the dataset with *HTTP* cached/non-cached traffic.

3.6 Performance Impact of Number of Labels

When reviewing recent fingerprinting approaches, we noticed huge discrepancies concerning the number of labels on which the methods were evaluated. Some benchmarks use datasets with fewer than 200 labels ([23, 53, 56, 64, 83, 84, 100]), $200 - 1000$ labels ([102]), or more than 1000 labels ([10, 26, 27]). In the previous experiment, though, we noticed that the classification accuracy changes significantly when the diversity of the tested dataset varies. To evaluate this further, we raise our next research question:

RQ4: To what extent does a different number of labels (e.g., domains) influence model accuracy?

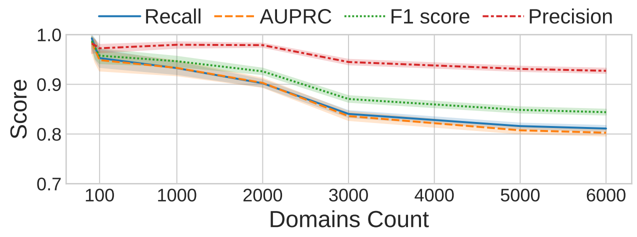

Figure 5: *Domain detection performance by the number of distinct labels in the dataset.* A model evaluated with fewer than 100 labels will report close to perfect scores. The performance decreases with the number of labels due to collisions in the certificate lengths, requests, and responses.

In order to answer RQ4, we benchmark the performance by the number of labels in the dataset. Figure 5 highlights the performance changes on all metrics when we vary the number of labels available in the dataset. A fingerprinting benchmark conducted with fewer than 100 labels quickly achieved perfect or close to perfect scores on all metrics. Increasing the number of labels in the dataset - and implicitly the number of samples - can affect the domain classification problem in various ways. One possible issue is that more and more labels may share the same server certificate (in the additional subjects section), reducing the importance of that TLS stage. Another potential issue is the feature set - we rely on payload lengths, and value collisions are more likely with more samples. It is critical to evaluate the models in a more practical setup.

Takeaways: Observing the performance on a larger number of labels is critical to understanding the collision effect. Evaluating on too small datasets *severely* over-approximates a model's performance in real-world setups.

4 HTTPS PAGE FINGERPRINTING

So far, we have been concerned with domain classification, i.e., identifying the domain (not the subpage) a client visited. Page identification, i.e., finding the subpage on a given domain a client visits, is significantly more challenging. In Section 3.3, we highlighted that the server certificate, first request, and first response provide high entropy for the domain classification. In the page classification problem, the server certificate is not informative, since it is the same for all the pages from the domain. The client request is also not helpful, especially for websites with a constant URL length (e.g., IMDB or 9GAG). Furthermore, the first response might look similar, if not identical, for dynamic-content pages. These constraints seem to contradict the general intuition that the beginning of the communication leaks the most information in the burst metadata (e.g., [23, 36]) and might force the observer to search for fingerprint entropy later in the flow or a different TLS flow— *but which and where?*

To study this, we now turn to the three datasets for which we retrieved thousands of pages (see Table 2): Wikipedia, 9GAG, and IMDB. Instead of varying the domain, in the following, we vary and aim to identify the visited page. We start with an attempt to use the *first* TLS flow (Section 4.1), then show how picking a different, more characteristic flow improves performance (Section 4.2), and explore how combining *multiple* can further boost accuracy (Section 4.3). Finally, we evaluate targeted defenses (Section 4.4) and how much HTTP-layer variations impact classification (Section 4.5).

4.1 Page Classification using the First TLS Flow

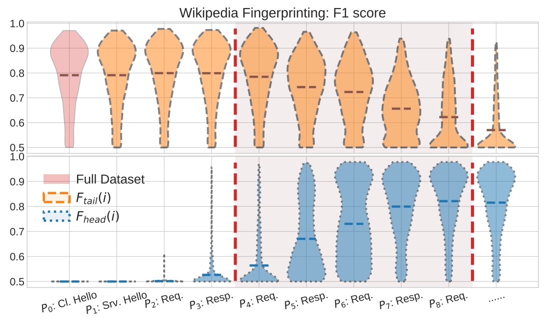

Figure 6: *Critical window of prediction support (F1 score) for Page Detection on the Wikipedia dataset.* The beginning of the communication is not as informative as in the domain fingerprinting task. The legend and representation have the same meaning as in Figure 2.

The first naïve attempt at classification is to use the first TLS flow of the web communication as we did for domain fingerprinting. Indeed, the first flow carries the encrypted HTTP request path we are trying to detect. In this section, we thus investigate:

> **RQ5: Which phases of web communication carry distinctive patterns allowing for page fingerprinting?**

We use *Wikipedia* as an illustrative example that helps better understand the differences between domain and page classification. The main characteristics of the samples are that each *Wikipedia* page load triggers a small number of additional TCP flows, the communication in the application layer is done using *HTTP/2*, and pages have varying URL lengths. We bootstrap the steps defined in Definition 3.1 and Definition 3.2 to identify the critical window of prediction support. In contrast to domain classification, we consider a longer traffic window in search of entropy to compensate for our hypothesis that the TLS handshake and initial payloads are no longer as informative.

Figure 6 shows the highly predictive region in the traffic. We observe that the front part of the traffic ($P_0 - P_3$) does not help detect the page. While the pages have various URL lengths, the first request does not have enough variety to be distinctive, especially when comparing many pages. The first payloads retrieve generic content, which is uninformative at the payload length level, while later packets retrieve page-specific content. We observe that P_4 is the first packet whose exclusion from P_{tail} massively degrades the fingerprinting performance - hence, the start of the critical window of prediction support. We also observe that P_8 is the first payload whose inclusion in the critical windows leads to an indistinguishable fingerprinting performance compared to the entire flow. We conclude that the critical window of prediction for this task is $P_4 - P_8$.

This is another example of why the protocol-centric approach to data analysis is beneficial. The current test scenario proves that the beginning of communication is not always information-rich, as generally believed. Instead, the fingerprinting task can vary the critical window of prediction support, and our proposed methodology can uncover its exact location.

Takeaways: Unlike domain classification, the entropy source of page classification can be found later in the connection and not necessarily at the beginning of the flow.

4.2 Identifying Characteristic Subdomains

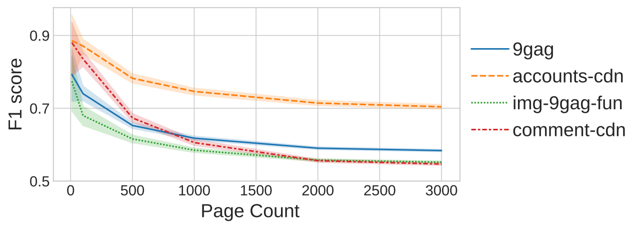

Figure 7: *Page Detection on 9GAG using a Single Flow.* The main flow *9gag.com* (blue) leaks less information than the *accounts-cdn* (orange) subdomain.

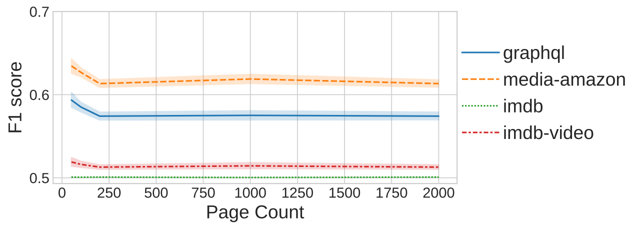

Figure 8: *Page Detection on IMDB using a Single Flow.* The flow *imdb.com* (green) offers less entropy than the *api.graphql* (blue) and *media-amazon* (orange) subdomains.

Fingerprinting models might have little success on the first TCP flow. One reason is that various websites load a standard webpage template on the first request, which populates the page with dynamic content in *different* flows. From the TLS payload size point of view, the initial flow cannot carry enough useful entropy. A more powerful fingerprint could be derived from page content like images or videos [26, 27, 95], loaded on separate TLS flows. In other

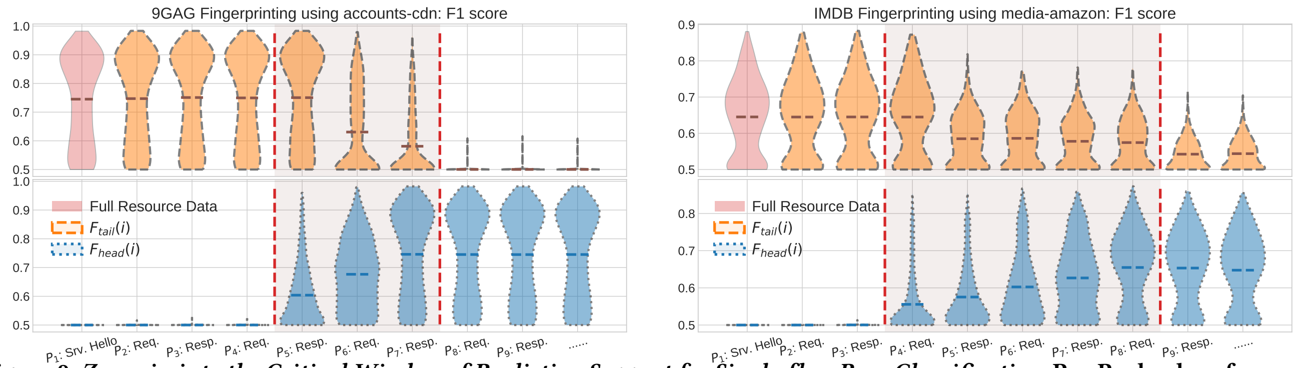

Figure 9: *Zoom-in into the Critical Window of Prediction Support for Single-flow Page Classification.* Per-Payload performance for each flow that carries important information on (left) a flow from 9GAG and (right) a flow from IMDB, using 1000 labels from each dataset.

words, the flow that contains the HTTP path we are trying to fingerprint can have insufficient entropy. We thus examine the utility of *subsequent* TLS flows:

RQ6: Can other TLS flows in a page trace contain a critical window of prediction support for accurate page fingerprinting?

We illustrate these situations using the 9GAG and IMDB datasets. Their distinctive features are that the first flow loads generic templates, which retrieve dynamic content and trigger other TCP flows to various subdomains, and that the pages have constant URL lengths and are rich in large resources like images or videos.

Figure 7 highlights the utility of subsequent flows from the *9GAG* dataset by the number of pages in the dataset. The blue line is the initial flow containing the encrypted HTTP path we want to fingerprint. However, that flow is not rich enough for fingerprinting - likely related to dynamic content-loading. Instead, a better predictor is the flows to the *accounts-cdn* subdomain. The flows load a set of user-related images and information from the comment section of each page. Similarly, Figure 8 highlights the usage of flows in fingerprinting a page from the *IMDB* dataset. The main flow (in red) to *imdb.com* is not informative enough for similar reasons as for 9GAG. In contrast to the 9GAG dataset, IMDB has *multiple* subdomain flows that create a useful fingerprint, namely the flows to *api.graphql* (page-specific database queries) and *media-amazon* (page-specific images).

We can optimize the data collection even further. For each of the subdomains that are rich in fingerprinting information, we can follow a protocol-centric approach and identify the exact window of interest by applying the same technique as on Tranco (Figure 2) and Wikipedia (Figure 6). Figure 9 exemplifies this process, with an informative flow for 9GAG (left) and IMDB (right). Similar to *Wikipedia*, each subdomain flow metadata starts to leak information at later stages. Following Definition 3.1 and Definition 3.2, the 9GAG critical window of prediction support is found between payloads $P_5 - P_7$, while for the IMDB flow is between $P_4 - P_8$. This confirms our finding in Section 4.1 and re-enforces the need for the current methodology to extract the window of prediction support.

Takeaways: The critical window of prediction support for page classification might be located in subsequent TLS flows, which could provide a better fingerprint than the initial main flow. We can identify which flows and sections carry the most information and use it to improve fingerprinting.

4.3 Page Classification using Multiple Flows

As seen in Section 4.1 and Section 4.2, single flows can create a valuable fingerprint for page classification. However, these flows might create even more powerful fingerprints when combined. To visualize this, we use a similar approach as the violin plots from Figure 2, but instead of masking before and after *payloads*, we hide before and after *subdomain flows*. We then study the following question:

RQ7: Does combining multiple flows improve page classification quality?

Consider loading a page on *9GAG*. The page will get populated with content from a set of subdomains on loading. Practically, $F_{\text{9gag.com}} = (P_0, P_1, \ldots, P_N)$, and it will subsequently trigger a set of flows $\mathbf{F}_{\text{9gag.com}} = (F_{\text{9gag.com}}, F_{\text{subdomain1}}, F_{\text{subdomain2}}, \ldots)$. We aim to combine multiple subdomain flows and test whether they leak more information this way. We cannot assume a fixed order in which the relevant subdomains are contacted. Consequently, during training, we enforce an order of the subdomains of interest in the tensor, e.g., "9gag.com" flow is always in the first position, "subdomain1" flow is always in the second position, etc. At inference time, we populate the exact slice in the tensor allocated for the detected subdomain, thus preventing unwanted effects from a different order of flow creation. Concretely, the observer allocates a tensor of shape $(N_{subdomains}, N_{payloads}, N_{feats})$, where the first dimension is about the subdomains of interest, the second dimension contains the window of interest from each connection, and the last dimension includes the features from each burst.

The multi-flow classification benchmarks on the 9GAG dataset are illustrated in Figure 10 (left). Compared to single-flow fingerprinting in Figure 7, the multi-flow fingerprinting shows important improvements in both the F1 score and the AUPRC metric, both

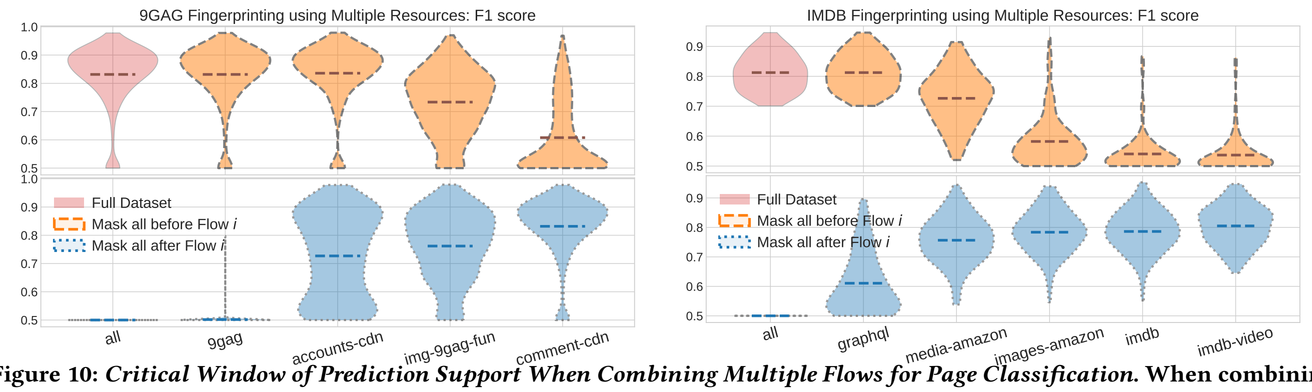

Figure 10: *Critical Window of Prediction Support When Combining Multiple Flows for Page Classification.* When combining multiple TLS flows, the predictive performance for the 9GAG (left) and IMDB (right) datasets, using 1000 labels in each dataset.

Dataset	Subdomain(s)	F1 Score	AUPRC
Wiki	wikipedia	0.791 ± 0.01	**0.703 ± 0.01**
	wikimedia	0.635 ± 0.01	0.528 ± 0.01
	Combined	**0.804 ± 0.01**	0.677 ± 0.02
9GAG	img-9gag-fun	0.585 ± 0.01	0.519 ± 0.01
	comment-cdn	0.606 ± 0.01	0.529 ± 0.01
	9gag	0.618 ± 0.01	0.558 ± 0.01
	accounts-cdn	0.746 ± 0.01	0.645 ± 0.01
	Combined	**0.831 ± 0.01**	**0.803 ± 0.01**
IMDB	imdb	0.536 ± 0.01	0.500 ± 0.01
	api.graphql	0.714 ± 0.01	0.680 ± 0.01
	media-amazon	0.679 ± 0.01	0.538 ± 0.01
	Combined	**0.807 ± 0.01**	**0.805 ± 0.01**

Table 4: Page Detection Performance when combining flows from multiple subdomains.

being boosted above 0.8 on average, while individual flows were reporting scores below 0.65 (AUPRC) and 0.75 (F1), respectively. The flow breakdown in the diagram shows that flows from *accounts-cdn*, *img-9gag*, and *comment-cdn* subdomains contribute to improving the fingerprint. Numerically, Table 4 shows that stacking flows leads to a $\Delta = 0.085$ improvement for the F1 score over the best individual flow and to a $\Delta = 0.158$ increase for the AUCPRC score.

We apply the same approach to the IMDB dataset, illustrated in Figure 10 (right). We also observe improvements over the single flow fingerprinting in Figure 8. The combined flows from *api.graphql* and *media-amazon* subdomains lead to a distinctive fingerprint. Table 4 supports these observations, showing a jump above 0.8 in both the F1 score ($\Delta = 0.093$) and AUPRC ($\Delta = 0.125$) metrics - compared to the best performing individual flow - indicating the model's chance at discriminating pages.

For completeness, we also tested the third dataset for page classification, the Wikipedia dataset. However, Table 4 shows no improvement for either metric, especially not for AUPRC. This could be related to the data collection process. The *wikimedia* flows, which are retrieved over *HTTP/2*, might lack a detectable pattern due to multiplexing, and they degrade the information offered in combination with the *wikipedia* flows. In our data collection process, we do not attempt to address the multiplexing complexity but rather rely on collecting a larger number of samples, which might lead to detectable patterns.. We further discuss the HTTP/2 multiplexing limitations in Section 5.1.

Takeaways: Combining multiple flows can lead to better page classifiers, as shown for the 9GAG and IMDB datasets. However, the Wikipedia dataset shows that individual flows might be more beneficial for some domains. Which strategy works best is thus task-specific and can be discovered using a protocol-centric approach.

4.4 Targeted Page Fingerprinting Defenses

The analysis for **RQ6** and **RQ7** highlighted that attackers could identify page fingerprints on auxiliary flows that carry dynamic, page-specific content. From a different perspective, prior work ([10, 86]) showed that combining subdomain names can create page fingerprints. Knowing how to identify which flows carry the most entropy for page classification, we next want to investigate the following research question:

RQ8: Can we use the critical prediction windows to deploy page-specific defenses against fingerprinting?

Similar to **RQ2**, we aim to test if the protocol-centric findings also hold for the defensive side. To this end, we again model an attacker who knows the padding strategy and augments the training dataset accordingly (see Section 3.4). We benchmark the page classification performance under various padding block sizes. For each padding block N, each relevant payload in a flow is padded to the next multiple of N. For each of the page classification datasets, we pad the flows uncovered as predictive in the previous section: (1) For the Wikipedia dataset, we pad the first TLS flow; (2) For the 9GAG dataset, we pad the relevant payloads from *accounts-cdn*, *img-9gag*, and *comment-cdn*; (3) For the IMDB dataset, we pad the important payloads from the *api.graphql* and *media-amazon* subdomains.

Figure 11 illustrates the effects of various padding block sizes against fingerprinting. The attacker has a different challenge for each dataset. The Wikipedia padding with a small block size is adequate since the primary entropy source lies in the text body. This also suggests that any page edit might make the models mispredict, having a similar effect. The 9GAG dataset is the most difficult to defend, needing a padding block of more than 2000 bytes to hide

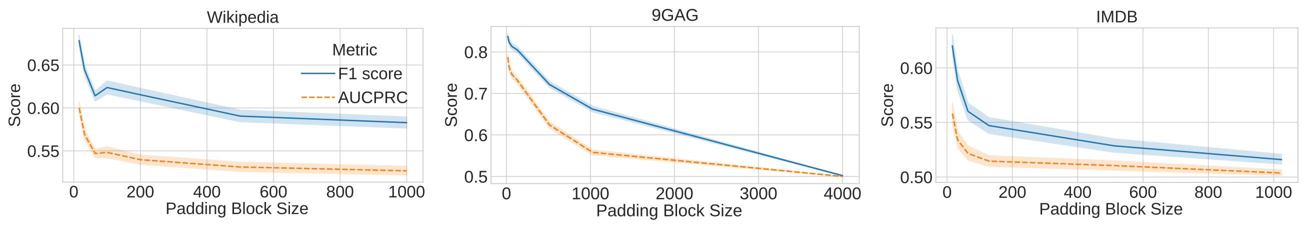

Figure 11: *Performance when padding the flows against page fingerprinting.* Left: Wikipedia. Middle: The 9GAG. Right: IMDB. We assume an advanced attacker who knows the padding strategy for each dataset and augments the training dataset.

the patterns efficiently. This might be related to the large resources that create the unique fingerprint from *accounts-cdn*, *img-9gag*, and *comment-cdn*. The IMDB dataset can be padded with blocks below 200 bytes to hide particularities. The *api.graphq* subdomain is used for database requests and does not have a larger footprint, and this justifies why a smaller padding block is enough. As established in the isolated flow tests, *media-amazon* alone is insufficient for good fingerprint detection.

Similar to **RQ2**, we work in a toy defensive setup to test the value of the identified entropy windows. While this setup is not intended to scale and address all attack channels (like timing-based), it confirms that protecting a set of messages can be effective against fingerprinting. Therefore, a protocol-centric analysis can facilitate more effective and tailored defenses.

Takeaways: Uncovering the combination of flows that fingerprint a website can also be used for defenses. The protocol-centric analysis could reveal a suitable padding block size for each page defense task.

4.5 User Agents Impact on Page Classification

The fingerprinting performance is not only susceptible to padding, as shown in the previous section, but also to any alteration in the HTTP layer. In **RQ3**, we reviewed the impact of HTTP caching on domain fingerprinting. Another potential pitfall is the HTTP user agents, which can instruct the servers to adapt the page design to the client's device. This raises the following research question:

RQ9: How do the HTTP User Agents affect the page fingerprinting quality?

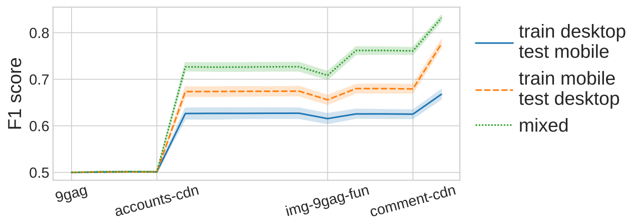

Figure 12: *HTTP User Agent Impact on the Page classification problem.* The scenarios are: (1) train on web traffic with desktop user agents and test on traffic with mobile user agents (blue line), (2) vice versa (orange line), and (3) train and test on a mixture of user agents, i.e., both desktop and mobile (green line).

In order to answer RQ9, we regenerate the 9GAG dataset using the Chrome Browser on a Linux device and by varying the User-Agents in the same browsing environment. The setup is relevant to our research question because the servers adapt the response content (such as image sizes and layout) to the User-Agent headers.

Figure 12 illustrates the effects of user agents on the 9GAG dataset, which has different page layouts for desktop (e.g., computers, laptops) and mobile versions (e.g., phones, tablets). We compare by separating or mixing desktop and mobile user agents. Besides altering the response (and its size), the custom user-agent header also modifies the request size. This situation is reflected in the performance of downstream models: Models trained on traffic generated with desktop user agents show a significant degradation when tested on mobile traffic, with an AUPRC below 0.6 for all the flows combined. The models trained on traffic with mobile user agents and tested on traffic from desktop devices show a slightly better performance, but it is still worse than the reference models trained on a mixture of user agents. The results highlight the importance of augmenting the datasets with various user agents from multiple types of devices (laptops, phones, tablets, etc). Note that training and testing on a mixed dataset leads to an equally good performance as training and testing on a pure dataset. This shows that user-agent diversity does not degrade performance if (and only if) it is considered during training.

A future investigation could simulate the same dataset using different physical devices and browsers. While the page content should be similar - since it is guided by the User-Agent header - various devices and browsers might employ diverse fetching techniques - with different versions of TLS and HTTPS - bringing even more diversity to the dataset.

Takeaways: Like HTTP caching, various HTTP user agents impact the performance of page classification. The data collection procedure should augment the dataset with various HTTP user agents from various devices, operating systems, and browsers.

5 DISCUSSION

In this section, we review some limitations that might threaten the validity of the results from the protocol and ML perspectives (Section 5.1); we analyze the benefits of our investigation to other defense strategies (Section 5.2); and we discuss some other fingerprinting strategies, that we did not cover in this work (Section 5.3).

5.1 Limitations of the HTTPS Fingerprinting

TLS versions. The fingerprinting models should be tailored to the TLS versions and cipher suites, as the behavior of the encryption algorithms might vary. In this work, we focus our research on the latest version, TLS 1.3, and the behavior of its cipher suites based on AES-GCM and Poly1305 symmetric encryption. As we highlight in Section 2.2, these algorithms do not add padding during encryption by default. Older versions of TLS include other cipher suites, which have different behaviors and might be considered deprecated nowadays. For example, TLS 1.2 includes AES with CBC (Cipher Block Chaining) mode, a deprecated variant of AES that adds padding during encryption by design. In consequence, different TLS versions and cipher suites can alter the burst information differently and require separate modeling.

HTTP versions. We showed that HTTP caching and user agents impact fingerprinting but can be addressed during training. Different HTTP versions can pose additional challenges and require a dedicated discussion. HTTP/1.1 behavior is the most accessible for fingerprinting tasks because it follows a deterministic request/response pattern. However, HTTP/2 and HTTP/3 include several new features that can decay the available burst information. For example, HTTP/2 multiplexing allows resources to be fetched asynchronously, thus breaking the request/response pattern of HTTP/1 [64]. The fingerprinting models rely on the order and atomicity of operations and thus can degrade under *HTTP/2*. Future works could investigate robust *HTTP/2* dataset creation techniques.

We emphasize that the HTTP versions mainly affect page classification tasks, not domain classification tasks. For domain classification tasks, a big part of the critical window of prediction support is in the TLS handshake, which is not dependent on the HTTP version. In contrast, the critical window for page classification tasks can be later in the HTTP connection, and various HTTP versions and features can alter metadata. In consequence, different HTTP versions require different modeling strategies for fingerprinting.

Distribution shifts. From the ML perspective, one of the most significant limitations of fingerprinting models is the performance degradation under distribution shifts [76]. This paper highlights the critical window of prediction support for fingerprinting models, but the identified flows and payloads might evolve and change over time. While the interpretability factor can help us prepare and scale such models, it is not a perfect solution. One example is in the 9GAG datasets, where the components of the comment section were identified as predictive. The content is expected to be volatile, and the training datasets must be frequently refreshed to stay relevant. The exact impact of temporal distribution shifts requires a separate in-depth analysis.

5.2 Implications to Existing Defense Strategies

Section 3.4 and Section 4.4 discussed the fingerprinting defenses only from the perspective of traffic padding. As presented in Section 2.3, other popular defenses exist: fixed-rate traffic, traffic morphing, chaff traffic, adversarial perturbations, multihoming/traffic splitting, and application layer defenses.

At first glance, our findings could be adapted to optimize these other techniques, but an in-depth investigation is required: (1) Fixed-rate traffic defenses ([11–13, 58]) force the transmission of payloads at a fixed rate, disrupting the timing information leakage. While our focus was on packet lengths, the steps could be adapted to identify the critical window of prediction support for timing attacks, and it might be different from the packet length critical window. The fixed-rate traffic defense then could focus on delaying only the payloads inside the timing critical window of prediction support; (2) Traffic morphing techniques ([5, 17, 77, 91, 96, 97]) assume knowledge of server pages and attempt to morph the burst information of the current page in order to create a collision with other pages on the server. This defense technique could be optimized to morph only a specific subset of payloads included in the critical window of prediction support; (3) Chaff traffic techniques ([4, 5, 17, 36, 45, 61, 75, 91]) insert random payloads in the communication to disrupt any burst information. This defense could add dummy payloads only inside the critical window of prediction support in order to save bandwidth; (4) Adversarial perturbations ([67, 77]) target the attacker's model decision boundary by adding tailored noise to payloads to force it to misclassify the correct label. Similar to traffic morphing, the noise could be added only in the critical window; (5) Traffic splitting ([24, 40]) is a technique popular in Tor and multipath TCP [30] for splitting the communication over multiple TCP connections. With this defense, an attacker is forced to correlate multiple connections for the correct label, which is a difficult task. This defense could be extended to distribute the payloads inside the critical window of prediction across multiple connections; (6) Application layer defenses ([60]) focus on HTTP features, such as ranged requests, to disrupt sources of information leakage. These defenses could be extended to identify the critical window of prediction support and randomly split the bytes requested inside this window across multiple ranged requests, thus optimizing the entire defense; (7) Other untargeted defenses could rise out of ad and script blocking. These alterations to traffic can affect multi-flow fingerprinting strategies if an ad or script is included in the critical window of prediction.

Our aim was to test the importance of the discovered windows of prediction support in a toy scenario using only padding. We do not claim to have a scalable defensive solution, and we do not circumvent all the limitations of the padding defenses that other methods offer. We hope to see some protocol-centric solutions applied to the other defenses. Our insights could also be combined with these techniques and might improve performance as we can hint at which parts of the network trace have to be targeted.

5.3 Other Fingerprinting Perspectives

Domain Classification using Multiple Flows. Section 3.3 focused on domain fingerprinting using a single TLS flow, highlighting the source of classification improvement. Section 4.3 reviewed the page classification problem and showcased that multiple TLS flows can be combined to train a potent page classifier. One omitted perspective is to use multiple TLS flows to identify domains – previously studied from the DNS perspective in [10, 86]. The defenses can be extended by identifying which additional TLS flows help create the domain fingerprint outside the specified window of entropy in Section 3.3.

Other Sources of Information. In this work, we focus only on burst-based fingerprinting techniques. Other methods include timing information, which is not covered in this work. While the timing information can defeat padding techniques, this source is biased towards the training network only, thus ignored by recent works ([100]). The main challenge is finding an invariant across multiple networks in the timing measurements. In consequence, a dedicated discussion about the generalization of timing attacks across networks would be another interesting future direction.

6 CONCLUSION

In this work, we analyzed traffic fingerprinting from a *protocol-centric* perspective, focusing on the particularities of TLS and HTTP – instead of the common model-centric approach. This novel perspective can lead to tailored and optimized solutions for fingerprinting attacks and defenses. It also provides an interpretable way to understand the differences between various fingerprinting tasks and their limitations. One direct insight is that, contrary to previous works, the beginning of the communication is not always relevant for fingerprinting. Instead, the critical window of prediction support differs from task to task and can be uncovered using our traffic analysis methodologies.

We also have shown that the success of web fingerprinting is highly sensitive to many experimental parameters. Performance depends *significantly* on which parts of an HTTPS trace are considered, how many labels are in the dataset, whether HTTP caching is present, and how the traces are modeled underneath. These influences underline that future experiments require more careful experimentation and that the design decisions must be documented.

As a follow-up to these findings, we believe that further aspects of fingerprinting have not yet been sufficiently studied. Future works could analyze the impact of modern HTTP features (e.g., multiplexing) on the training data collection. At the same time, a protocol-centric approach could be combined with other side-channel attacks, like timing-based, to uncover other windows of prediction support.

Finally, we argue that future traffic fingerprinting research should focus on a data-centric perspective, emphasizing the targeted protocol rather than solely developing new models.

ACKNOWLEDGMENTS

We want to thank our shepherd and reviewers for their insightful comments.

REFERENCES

[1] [n. d.]. *Selenium*. https://www.selenium.dev/
[2] 2020. *CIRA-CIC-DoHBrw-2020*. Technical Report. https://www.unb.ca/cic/datasets/dohbrw-2020.html Available at https://www.unb.ca/cic/datasets/dohbrw-2020.html.
[3] 2023. TLS ClientHello extension permutation. https://chromestatus.com/feature/5124606246518784 Available at https://chromestatus.com/feature/5124606246518784.
[4] Ahmed Abusnaina, Rhongho Jang, Aminollah Khormali, DaeHun Nyang, and David Mohaisen. 2020. DFD: Adversarial Learning-based Approach to Defend Against Website Fingerprinting. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. IEEE, Toronto, ON, Canada, 2459–2468. https://doi.org/10.1109/INFOCOM41043.2020.9155465
[5] Khaled Al-Naami, Amir El-Ghamry, Md Shihabul Islam, Latifur Khan, Bhavani Thuraisingham, Kevin W. Hamlen, Mohammed Alrahmawy, and Magdi Z. Rashad. 2021. BiMorphing: A Bi-Directional Bursting Defense against Website Fingerprinting Attacks. *IEEE Transactions on Dependable and Secure Computing* 18, 2 (March 2021), 505–517. https://doi.org/10.1109/TDSC.2019.2907240
[6] Blake Anderson and David McGrew. 2016. Identifying encrypted malware traffic with contextual flow data. In *Proceedings of the 2016 ACM workshop on artificial intelligence and security*. 35–46.
[7] Blake Anderson and David McGrew. 2017. Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity. In *Proceedings of the 23rd ACM SIGKDD International Conference on knowledge discovery and data mining*. 1723–1732.
[8] Alireza Bahramali, Ramin Soltani, Amir Houmansadr, Dennis Goeckel, and Don Towsley. 2020. Practical traffic analysis attacks on secure messaging applications. *arXiv preprint arXiv:2005.00508* (2020).
[9] Leo Breiman. 2001. Random forests. *Machine learning* 45 (2001), 5–32.
[10] Jonas Bushart and Christian Rossow. 2020. Padding ain't enough: Assessing the privacy guarantees of encrypted {DNS$}$. In *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*.
[11] Xiang Cai, Rishab Nithyanand, and Rob Johnson. 2014. CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. ACM, Scottsdale Arizona USA, 121–130. https://doi.org/10.1145/2665943.2665949
[12] Xiang Cai, Rishab Nithyanand, and Rob Johnson. 2014. Cs-buflo: A congestion sensitive website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. 121–130.
[13] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. 2014. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Scottsdale Arizona USA, 227–238. https://doi.org/10.1145/2660267.2660362
[14] Jiahao Cao, Zijie Yang, Kun Sun, Qi Li, Mingwei Xu, and Peiyi Han. 2019. Fingerprinting SDN applications via encrypted control traffic. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. 501–515.
[15] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.
[16] Heyning Cheng and Ron Avnur. 1998. Traffic analysis of SSL encrypted web browsing. *Project paper, University of Berkeley* (1998).
[17] Giovanni Cherubin, Jamie Hayes, and Marc Juarez. 2017. Website Fingerprinting Defenses at the Application Layer. *Proceedings on Privacy Enhancing Technologies* 2017, 2 (April 2017), 186–203. https://doi.org/10.1515/popets-2017-0023
[18] Cloudflare. 2023. JA3 Fingerprint. https://developers.cloudflare.com/bots/concepts/ja3-fingerprint/ Available at https://developers.cloudflare.com/bots/concepts/ja3-fingerprint/.
[19] Mauro Conti, Luigi Vincenzo Mancini, Riccardo Spolaor, and Nino Vincenzo Verde. 2015. Analyzing android encrypted network traffic to identify user actions. *IEEE Transactions on Information Forensics and Security* 11, 1 (2015), 114–125.
[20] Scott E Coull and Kevin P Dyer. 2014. Traffic analysis of encrypted messaging services: Apple imessage and beyond. *ACM SIGCOMM Computer Communication Review* 44, 5 (2014), 5–11.
[21] David R Cox. 1958. The regression analysis of binary sequences. *Journal of the Royal Statistical Society Series B: Statistical Methodology* 20, 2 (1958), 215–232.
[22] Tianyu Cui, Gaopeng Gou, Gang Xiong, Zhen Li, Mingxin Cui, and Chang Liu. 2021. {SiamHAN$}$:${$IPv6$}$ address correlation attacks on ${TLS$}$ encrypted traffic via siamese heterogeneous graph attention network. In *30th USENIX Security Symposium (USENIX Security 21)*. 4329–4346.
[23] Thilini Dahanayaka, Guillaume Jourjon, and Suranga Seneviratne. 2020. Understanding traffic fingerprinting CNNs. In *2020 IEEE 45th Conference on Local Computer Networks (LCN)*. IEEE, 65–76.
[24] Wladimir De la Cadena, Asya Mitseva, Jens Hiller, Jan Pennekamp, Sebastian Reuter, Julian Filter, Thomas Engel, Klaus Wehrle, and Andriy Panchenko. 2020. Trafficsliver: Fighting website fingerprinting attacks with traffic splitting. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 1971–1985.
[25] Utsho Dey. [n. d.]. *Kaggle 25k IMDb Movie Dataset*. https://www.kaggle.com/datasets/utsh0dey/25k-movie-dataset
[26] Mariano Di Martino, Peter Quax, and Wim Lamotte. 2019. Realistically fingerprinting social media webpages in HTTPS traffic. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*. 1–10.
[27] Mariano Di Martino, Pieter Robyns, Peter Quax, and Wim Lamotte. 2018. IUPTIS: A Practical, Cache-resistant Fingerprinting Technique for Dynamic Webpages.. In *WEBIST*. 102–112.
[28] Christian J Dietrich, Christian Rossow, and Norbert Pohlmann. [n. d.]. CoCoSpot: Clustering and Recognizing Botnet Command and Control Channels using Traffic Analysis. ([n. d.]).
[29] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. 2016. Characterization of encrypted and vpn traffic using time-related. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*. 407–414.
[30] Alan Ford, Costin Raiciu, Mark Handley, and Olivier Bonaventure. 2013. *TCP extensions for multipath operation with multiple addresses*. Technical Report.
[31] Wikimedia Foundation. [n. d.]. *Wikimedia Downloads*. https://dumps.wikimedia.org

[32] Chuanpu Fu, Qi Li, Meng Shen, and Ke Xu. 2021. Realtime robust malicious traffic detection via frequency domain analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 3431–3446.

[33] Chuanpu Fu, Qi Li, and Ke Xu. 2023. Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis. *arXiv preprint arXiv:2301.13686* (2023).

[34] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 2000. Learning to forget: Continual prediction with LSTM. *Neural computation* 12, 10 (2000), 2451–2471.

[35] Jiajun Gong and Tao Wang. [n. d.]. Zero-delay Lightweight Defenses against Website Fingerprinting. ([n. d.]).

[36] Jiajun Gong and Tao Wang. 2020. Zero-delay lightweight defenses against website fingerprinting. In *29th USENIX Security Symposium (USENIX Security 20)*. 717–734.

[37] Harshita Gupta. [n. d.]. *Kaggle 9GAG Hot Posts*. https://www.kaggle.com/datasets/harshitagpt/9gag-hot-posts

[38] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A robust scalable website fingerprinting technique. In *25th USENIX Security Symposium (USENIX Security 16)*. 1187–1203.

[39] Simon Haykin. 1994. *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

[40] Sébastien Henri, Gines Garcia-Aviles, Pablo Serrano, Albert Banchs, and Patrick Thiran. 2020. Protecting against Website Fingerprinting with Multihoming. *Proceedings on Privacy Enhancing Technologies* 2020, 2 (April 2020), 89–110. https://doi.org/10.2478/popets-2020-0019

[41] Rebekah Houser, Zhou Li, Chase Cotton, and Haining Wang. 2019. An investigation on information leakage of DNS over TLS. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. 123–137.

[42] Martin Husák, Milan Čermák, Tomáš Jirsík, and Pavel Čeleda. 2016. HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting. *EURASIP Journal on Information Security* 2016 (2016), 1–14.

[43] Daniel Jarrett, Bogdan C Cebere, Tennison Liu, Alicia Curth, and Mihaela van der Schaar. 2022. Hyperimpute: Generalized iterative imputation with automatic model selection. In *International Conference on Machine Learning*. PMLR, 9916–9937.

[44] Steven Jorgensen, John Holodnak, Jensen Dempsey, Karla de Souza, Ananditha Raghunath, Vernon Rivet, Noah DeMoes, Andrés Alejos, and Allan Wollaber. 2023. Extensible machine learning for encrypted network traffic application labeling via uncertainty quantification. *IEEE Transactions on Artificial Intelligence* (2023).

[45] Marc Juárez, Mohsen Imani, Mike Perry, Claudia Dıaz, and Matthew Wright. 2015. WTF-PAD: toward an efficient website fingerprinting defense for tor. *CoRR, abs/1512.00524* (2015).

[46] Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. 2016. Toward an efficient website fingerprinting defense. In *Computer Security–ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I 21*. Springer, 27–46.

[47] Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. 2016. Toward an Efficient Website Fingerprinting Defense. http://arxiv.org/abs/1512.00524 arXiv:1512.00524 [cs].

[48] Been Kim, Rajiv Khanna, and Oluwasanmi O Koyejo. 2016. Examples are not enough, learn to criticize! criticism for interpretability. *Advances in neural information processing systems* 29 (2016).

[49] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *International conference on machine learning*. PMLR, 1885–1894.

[50] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. 2010. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE international symposium on circuits and systems*. IEEE, 253–256.

[51] Shuai Li, Huajun Guo, and Nicholas Hopper. 2018. Measuring Information Leakage in Website Fingerprinting Attacks and Defenses. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Toronto Canada, 1977–1992. https://doi.org/10.1145/3243734.3243832

[52] XuKui Li, Wei Chen, Qianru Zhang, and Lifa Wu. 2020. Building auto-encoder intrusion detection system based on random forest feature selection. *Computers & Security* 95 (2020), 101851.

[53] Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. 2022. Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification. In *Proceedings of the ACM Web Conference 2022*. 633–642.

[54] Chang Liu, Zigang Cao, Zhen Li, and Gang Xiong. 2018. Lafft: Length-aware fft based fingerprinting for encrypted network traffic classification. In *2018 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 1–6.

[55] Chang Liu, Zigang Cao, Gang Xiong, Gaopeng Gou, Siu-Ming Yiu, and Longtao He. 2018. Mampf: Encrypted traffic classification based on multi-attribute markov probability fingerprints. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[56] Chang Liu, Longtao He, Gang Xiong, Zigang Cao, and Zhen Li. 2019. Fs-net: A flow sequence network for encrypted traffic classification. In *IEEE INFOCOM 2019-IEEE Conference On Computer Communications*. IEEE, 1171–1179.

[57] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. 2020. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing* 24, 3 (2020), 1999–2012.

[58] David Lu, Sanjit Bhat, Albert Kwon, and Srinivas Devadas. 2018. DynaFlow: An Efficient Website Fingerprinting Defense Based on Dynamically-Adjusting Flows. In *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*. ACM, Toronto Canada, 109–113. https://doi.org/10.1145/3267323.3268960

[59] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. *Advances in neural information processing systems* 30 (2017).

[60] Xiapu Luo, Peng Zhou, Edmond WW Chan, Wenke Lee, Rocky KC Chang, Roberto Perdisci, et al. 2011. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows.. In *NDSS*, Vol. 11.

[61] Xiapu Luo, Peng Zhou, Edmond W W Chan, Wenke Lee, Rocky K C Chang, and Roberto Perdisci. [n. d.]. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. ([n. d.]).

[62] Nate Mathews, James K Holland, Se Eun Oh, Mohammad Saidur Rahman, Nicholas Hopper, and Matthew Wright. 2023. SoK: A Critical Evaluation of Efficient Website Fingerprinting Defenses. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, San Francisco, CA, USA, 969–986. https://doi.org/10.1109/SP46215.2023.10179289

[63] Vasilios Mavroudis and Jamie Hayes. 2023. Adaptive Webpage Fingerprinting from TLS Traces. http://arxiv.org/abs/2010.10294

[64] Brad Miller, Ling Huang, Anthony D Joseph, and J Doug Tygar. 2014. I know why you went to the clinic: Risks and realization of https traffic analysis. In *Privacy Enhancing Technologies: 14th International Symposium, PETS 2014, Amsterdam, The Netherlands, July 16-18, 2014. Proceedings 14*. Springer, 143–163.

[65] Christoph Molnar. 2020. *Interpretable machine learning*. Lulu. com.

[66] Boris Muzellec, Julie Josse, Claire Boyer, and Marco Cuturi. 2020. Missing data imputation using optimal transport. In *International Conference on Machine Learning*. PMLR, 7130–7140.

[67] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. [n. d.]. Defeating DNN-Based Traffic Analysis Systems in Real-Time With Blind Adversarial Perturbations. ([n. d.]).

[68] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. 2018. Deepcorr: Strong flow correlation attacks on tor using deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1962–1976.

[69] Y. Nir and A. Langley. 2018. *RFC8439: ChaCha20 and Poly1305 for IETF Protocols*. Technical Report RFC 8439. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/rfc8439 Available at https://datatracker.ietf.org/doc/html/rfc8439.

[70] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. 2016. Website Fingerprinting at Internet Scale.. In *NDSS*.

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

[72] Leif E Peterson. 2009. K-nearest neighbor. *Scholarpedia* 4, 2 (2009), 1883.

[73] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczynski, and Wouter Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *Proceedings 2019 Network and Distributed System Security Symposium*. Internet Society. https://doi.org/10.14722/ndss.2019.23386

[74] Neoklis Polyzotis and Matei Zaharia. 2021. What can data-centric AI learn from data and ML engineering? *arXiv preprint arXiv:2112.06439* (2021).

[75] Tobias Pulls. 2020. Towards Effective and Efficient Padding Machines for Tor. http://arxiv.org/abs/2011.13471 arXiv:2011.13471 [cs].

[76] Joaquin Quinonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. 2008. *Dataset shift in machine learning*. Mit Press.

[77] Mohammad Saidur Rahman, Mohsen Imani, Nate Mathews, and Matthew Wright. 2021. Mockingbird: Defending Against Deep-Learning-Based Website Fingerprinting Attacks With Adversarial Traces. *IEEE Transactions on Information Forensics and Security* 16 (2021), 1594–1609. https://doi.org/10.1109/TIFS.2020.3039691

[78] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. https://datatracker.ietf.org/doc/html/rfc8446 Available at https://datatracker.ietf.org/doc/html/rfc8446.

[79] Eric Rescorla. 2023. TLS Encrypted Client Hello. https://datatracker.ietf.org/doc/draft-ietf-tls-esni/ Available at https://datatracker.ietf.org/doc/draft-ietf-tls-esni/.

[80] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1135–1144.

[81] J. Salowey, A. Choudhury, and D. McGrew. 2008. *RFC 5288: AES Galois Counter Mode (GCM) Cipher Suites for TLS.* Technical Report RFC 5288. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/rfc5288 Available at https://datatracker.ietf.org/doc/html/rfc5288.
[82] Nabeel Seedat, Jonathan Crabbé, and Mihaela van der Schaar. 2022. Data-SUITE: Data-centric identification of in-distribution incongruous examples. In *International Conference on Machine Learning.* PMLR, 19467–19496.
[83] Meng Shen, Zhenbo Gao, Liehuang Zhu, and Ke Xu. 2021. Efficient fine-grained website fingerprinting via encrypted traffic analysis with deep learning. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS).* IEEE, 1–10.
[84] Meng Shen, Yiting Liu, Siqi Chen, Liehuang Zhu, and Yuchao Zhang. 2019. Webpage fingerprinting using only packet length information. In *ICC 2019-2019 IEEE International Conference on Communications (ICC).* IEEE, 1–6.
[85] Meng Shen, Yiting Liu, Liehuang Zhu, Xiaojiang Du, and Jiankun Hu. 2020. Fine-grained webpage fingerprinting using only packet length information of encrypted traffic. *IEEE Transactions on Information Forensics and Security* 16 (2020), 2046–2059.
[86] Sandra Siby, Ludovic Barman, Christopher Wood, Marwan Fayed, Nick Sullivan, and Carmela Troncoso. 2022. You get PADDING, everybody gets PADDING! You get privacy? Evaluating practical QUIC website fingerprinting protections for the masses. *arXiv preprint arXiv:2203.07806* (2022).
[87] Sandra Siby, Marc Juarez, Claudia Diaz, Narseo Vallina-Rodriguez, and Carmela Troncoso. 2019. Encrypted DNS–> Privacy? A traffic analysis perspective. *arXiv preprint arXiv:1906.09682* (2019).
[88] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2019. Deep inside convolutional networks: Visualising image classification models and saliency maps. arXiv 2013. *arXiv preprint arXiv:1312.6034* (2019).
[89] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. 2018. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.* 1928–1943.
[90] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. 2019. Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security.* 1131–1148.
[91] Jean-Pierre Smith, Luca Dolfi, Prateek Mittal, and Adrian Perrig. [n. d.]. QCSD: A QUIC Client-Side Website-Fingerprinting Defence Framework. ([n. d.]).
[92] Van Tong, Hai Anh Tran, Sami Souihi, and Abdelhamid Mellouk. 2018. A novel QUIC traffic classifier based on convolutional neural networks. In *2018 IEEE Global Communications Conference (GLOBECOM).* IEEE, 1–6.
[93] Thijs Van Ede, Riccardo Bortolameotti, Andrea Continella, Jingjing Ren, Daniel J Dubois, Martina Lindorfer, David Choffnes, Maarten van Steen, and Andreas Peter. 2020. Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic. In *Network and distributed system security symposium (NDSS)*, Vol. 27.
[94] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
[95] Kailong Wang, Junzhe Zhang, Guangdong Bai, Ryan Ko, and Jin Song Dong. 2021. It's not just the site, it's the contents: intra-domain fingerprinting social media websites through CDN bursts. In *Proceedings of the Web Conference 2021.* 2142–2153.
[96] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. [n. d.]. Effective Attacks and Provable Defenses for Website Fingerprinting. ([n. d.]).
[97] Tao Wang and Ian Goldberg. [n. d.]. Walkie-Talkie: An Effective and Efficient Defense against Website Fingerprinting. ([n. d.]).
[98] Pengfei Wei, Zhenzhou Lu, and Jingwen Song. 2015. Variable importance analysis: A comprehensive review. *Reliability Engineering & System Safety* 142 (2015), 399–432.
[99] Steven Euijong Whang, Yuji Roh, Hwanjun Song, and Jae-Gil Lee. 2023. Data collection and quality challenges in deep learning: A data-centric ai perspective. *The VLDB Journal* 32, 4 (2023), 791–813.
[100] Renjie Xie, Jiahao Cao, Enhuan Dong, Mingwei Xu, Kun Sun, Qi Li, Licheng Shen, and Menghao Zhang. 2023. Rosetta: Enabling Robust {TLS$}$ Encrypted Traffic Classification in Diverse Network Environments with ${$TCP-Aware$}$ Traffic Augmentation. In *32nd USENIX Security Symposium (USENIX Security 23).* 625–642.
[101] Junchi Xing and Chunming Wu. 2020. Detecting anomalies in encrypted traffic via deep dictionary learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS).* IEEE, 734–739.
[102] Shi-Jie Xu, Guang-Gang Geng, Xiao-Bo Jin, Dong-Jie Liu, and Jian Weng. 2022. Seeing traffic paths: encrypted traffic classification with path signature features. *IEEE Transactions on Information Forensics and Security* 17 (2022), 2166–2181.
[103] Pengwei Zhan, Liming Wang, and Yi Tang. 2021. Website fingerprinting on early QUIC traffic. *Computer Networks* 200 (2021), 108538.
[104] Qingyuan Zhao and Trevor Hastie. 2021. Causal interpretations of black-box models. *Journal of Business & Economic Statistics* 39, 1 (2021), 272–281.

A DATA COLLECTION

We visit the URLs from Table 2 with isolated Docker containers based on Selenium [1] and the Chrome browser. We patch the image to parameterize it with various options, such as HTTP caching or custom user agents. We collect the generated traffic in PCAP network traces using *tcpdump* and process the traces using *pyshark*. Our experiments use various combinations of HTTP caching and HTTP user agent options. We construct each dataset by identifying and extracting payload features from each TCP flow in the traces. The features we include are the burst length and direction of communication.

B EVALUATION METRICS

We focus on two evaluation metrics in this work, both suitable for imbalanced datasets. First, we report on the F1 score, the harmonic mean of precision and recall. The F1 score captures the model's ability to make accurate positive predictions (precision = TP/(TP+FP)) while capturing a significant ratio of actual positive instances (recall = TP/(TP+FN)). It is computed as follows:

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (1)$$

A high F1 score indicates a good balance between precision and recall, meaning the model can make accurate positive predictions while minimizing FP and FN. A low F1 score suggests an imbalance between precision and recall, translating to the model being too restrictive with the positive class (low recall) or too permissive (high recall).

Second, we report the *AUPRC* (Area Under the Precision-Recall Curve), which quantifies the area under the Precision-Recall curve under different decision thresholds:

$$\text{AUPRC} = \int_0^1 \text{Precision}(\text{Recall})\, d\text{Recall} \quad (2)$$

A high AUPRC means the model makes optimistic predictions while minimizing false positives. A low AUPRC suggests the model struggles with too many false positive predictions or not capturing enough positive instances.

For every metric, we report *macro* averages, i.e., take the arithmetic mean of all the per-class scores.

C DATA AVAILABILITY

To ease reproducibility, we have released the code containing (1) the scripts used to collect the network traces, (2) the machine learning models we benchmarked, and (3) the experiments we conducted.

https://github.com/bcebere/Understanding-and-Explaining-Web-Fingerprinting-with-a-Protocol-Centric-Approach.

D MORE RESULTS

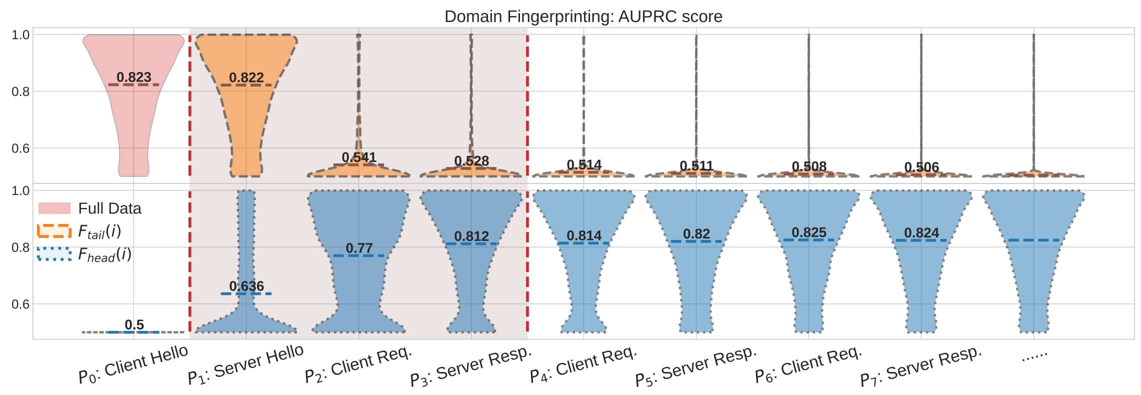

Figure 13: ***Detection of the Critical Window of prediction support for Domain fingerprinting (6472 domains) as violin plot.*** **Each violin plot shows the distribution of AUPRC scores (y-axis) of all tested domains, differentiated by which HTTPS payloads are included (x-axis). The legend and representation have the same meaning as in Figure 2.**

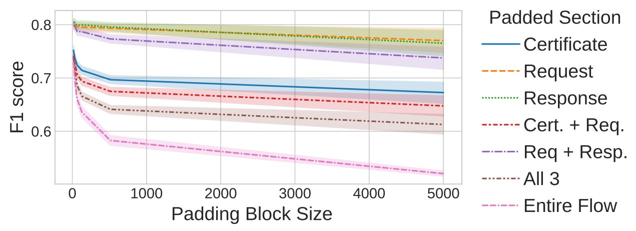

Figure 14: ***Attack performance by the size of the padding block (F1 score).***

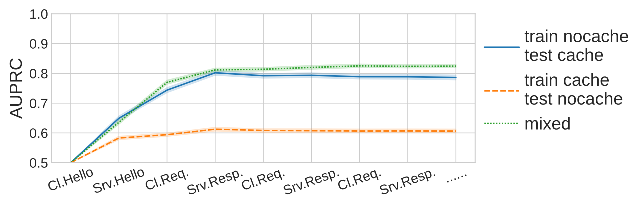

Figure 15: ***HTTP Caching impact on the Domain fingerprinting problem (AUPRC score).*** **The scenarios are: (1) train on data without any HTTP caching example and test on data with HTTP(S) caching (blue line), (2) train on data with any HTTP(S) caching example and test on data without HTTP(S) caching (orange line), and (3) train on a mixture of data with and without caching and also test on an unseen mixture (green line).**

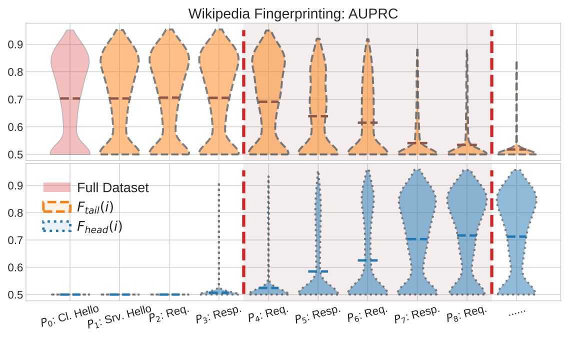

Figure 16: ***Critical Window Of Prediction Support for Page Detection on the Wikipedia dataset (AUPRC score).***

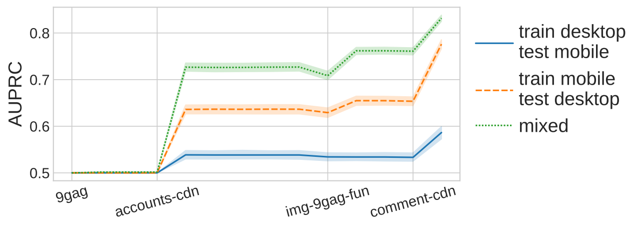

Figure 17: ***HTTP User Agent Impact on the Page classification problem (AUPRC score).*** **The scenarios are: (1) train on HTTPS traffic with desktop user agents and test on HTTPS traffic with mobile user agents (blue line), (2) vice versa (orange line), and (3) train and test on a mixture of user agents, i.e., both desktop and mobile (green line).**

E INFORMATION LEAKAGE IN DOMAIN FINGERPRINTING

Li et al. [51] introduced WeFDE, a technique for benchmarking fingerprinting attacks and defenses by measuring the information leakage in the dataset. The core idea is to measure the reduction of the website label's Shannon entropy (prediction uncertainty) - $H(W)$ - in the presence of the fingerprinting information. For example, a website with 512 subpages will have 9 bits of entropy (uncertainty) for fingerprinting, $H(W) = 9$. Then, we measure the entropy of the labels in the presence of fingerprinting metadata F - $H(W|F)$. Finally, the information leakage of the features F is computed as $I(F;W) = H(W) - H(W|F)$. WeFDE approximates $H(W|F)$ by computing the individual entropy of each feature in the dataset, removing redundant features due to high mutual information with other features, and by modeling the best-performing features using Kernel Density Estimators (KDE). As an analogy,

Model perspective	P_0 (Cl. Hello)	P_1 (Srv. Cert.)		P_2 (Req.)		P_3 (Resp.)		P_4 (Req.)		P_5 (Resp.)		...
	Leaked bits	Leaked	Δ bits	Leaked	Δ	Leaked	Δ	Leaked	Δ	Leaked	Δ	
$F_{\text{tail}}(i)$	8.95	8.81	↓ 0.14	8.75	↓ 0.13	7.42	↓ 1.33	7.04	↓ 0.38	6.88	↓ 0.16	
$F_{\text{head}}(i)$	1.18	2.31	↑ 1.13	6.84	↑ 4.53	8.41	↑ 1.57	8.67	↑ 0.26	8.80	↑ 0.13	

Table 5: *Information leakage (bits) breakdown per communication stage for the domain fingerprinting task, using 512 domains.* The models are identical to the ones in Section 3.3, by masking the connection before and after each payload.

Model perspective	P_0 (Cl. Hello)	P_1 (Srv. Cert.)		P_2 (Req.)		P_3 (Resp.)		P_4 (Req.)		P_5 (Resp.)		...
	F1 score	Score	Δ score	Score	Δ	Score	Δ	Score	Δ	Score	Δ	
$F_{\text{tail}}(i)$	0.857 ± 0.01	0.856 ± 0.01	↓ 0.01	0.712 ± 0.01	↓ 0.14	0.683 ± 0.01	↓ 0.03	0.644 ± 0.01	↓ 0.04	0.631 ± 0.01	↓ 0.01	
$F_{\text{head}}(i)$	0.5 ± 0.0	0.675 ± 0.01	↑ 0.17	0.811 ± 0.01	↑ 0, 13	0.844 ± 0.01	↑ 0, 03	0.845 ± 0.01	↑ 0.01	0.856 ± 0.01	↑ 0.01	

Table 6: *F1 score numerical breakdown per communication stage, for domain fingerprinting, using 6472 domains.* The table can be used as reference for Table 5.

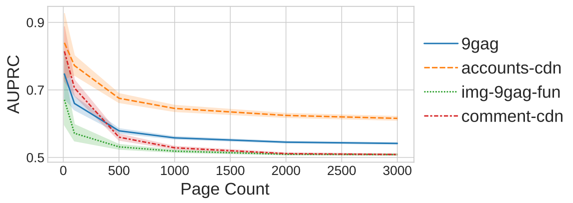

Figure 18: *Page Detection on 9GAG using a Single Flow (AUPRC score).* The main flow *9gag.com* (blue) is outperformed by the subdomain *accounts-cdn* (orange).

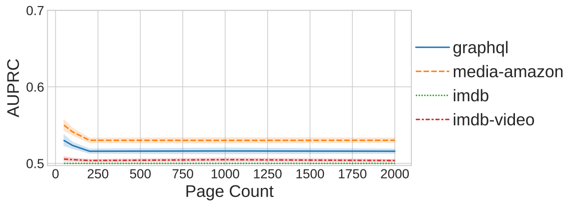

Figure 19: *Page Detection on IMDB using a Single Flow (AUPRC score).* The flow *imdb.com* (green) performs worse than *api.graphql* (blue) and *media-amazon* (orange).

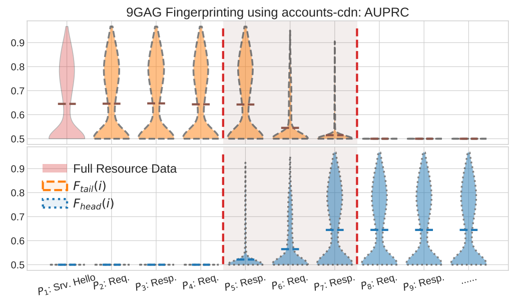

Figure 20: *Critical Window Of Prediction Support Zoom-in For Single-flow Page Classification (AUPRC score).* Per-payload performance for each flow that carries important information is demonstrated on a flow from 9GAG. The benchmarks are done using 1000 labels from each dataset.

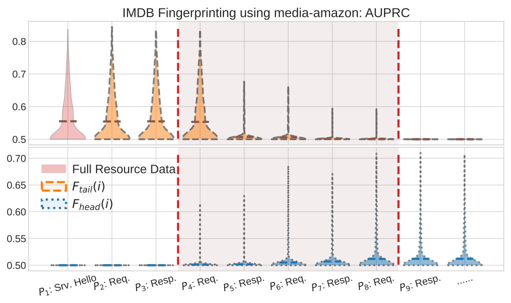

Figure 21: *Critical Window Of Prediction Support Zoom-in For Single-flow Page Classification (AUPRC score).* Per-payload performance for each flow that carries important information is demonstrated on a flow from IMDB. The benchmarks are done using 1000 labels from each dataset.

KDE can be considered the equivalent of histograms but for continuous variables. Using the trained KDE, the framework extracts the prediction probabilities and approximates $H(W|F)$.

This section employs the information leakage framework on the domain classification dataset. The benchmarking steps are similar to those in Section 3.3, and we measure the information leakage by masking payloads before and after each packet in the connection, using the F_{head} and F_{tail} models. In the information leakage benchmark, we reduce the dataset to 512 labels due to scalability issues.

Table 5 illustrates the bits of leakage per communication stage for the domain fingerprinting task. In the F_{head} modeling (second row), we can observe how including $P_1 - P_3$ leads to 8.41 bits leaks, leaving only 0.59 bits of uncertainty for the task. In the reversed scenario - F_{tail} - we observe a smaller drop in leakage when excluding $P_1 - P_3$, with ~ 7 bits leaked even when observing traffic only after P_4. However, that still leaves 2 bits of uncertainty for the label prediction, meaning there is a $\frac{1}{4}$ chance of guessing the correct label by using only the $P4 - \ldots$ information, leading to a low F1 score. For convenience, we report the matching F1 scores of the same stages in Table 6, using 6472 domains. In conclusion, the results are consistent with the findings observed using the F1 score in Figure 2 and Table 6.