Data-Explainable Website Fingerprinting with Network Simulation

Rob Jansen
U.S. Naval Research Laboratory
Washington, DC, USA
rob.g.jansen@nrl.navy.mil

Ryan Wails
U.S. Naval Research Laboratory
Georgetown University
Washington, DC, USA
ryan.wails@nrl.navy.mil

ABSTRACT

Website fingerprinting (WF) attacks allow an adversary to associate a website with the encrypted traffic patterns produced when accessing it, thus threatening to destroy the client-server unlinkability promised by anonymous communication networks. Explainable WF is an open problem in which we need to improve our understanding of (1) the machine learning models used to conduct WF attacks; and (2) the WF datasets used as inputs to those models. This paper focuses on explainable datasets; that is, we develop an alternative to the standard practice of gathering low-quality WF datasets using synthetic browsers in large networks without controlling for natural network variability. In particular, we demonstrate how network simulation can be used to produce explainable WF datasets by leveraging the simulator's high degree of control over network operation. Through a detailed investigation of the effect of network variability on WF performance, we find that: (1) training and testing WF attacks in networks with distinct levels of congestion increases the false-positive rate by as much as 200%; (2) augmenting the WF attacks by training them across several networks with varying degrees of congestion decreases the false-positive rate by as much as 83%; and (3) WF classifiers trained on completely simulated data can achieve greater than 80% accuracy when applied to the real world.

KEYWORDS

Tor, website fingerprinting, network simulation

1 INTRODUCTION

Tor is an anonymous communication network [14] with an estimated 8 million daily users [32]. Tor promises many privacy features to its users, including protection against tracking and surveillance, and resistance to browser fingerprinting and censorship [5]. Tor is valuable as a privacy-enhancing technology because it enables users to anonymously connect to internet destinations through its novel application of onion routing [45].

Website fingerprinting (WF) attacks attempt to break Tor users' anonymity by linking them to their online activities and thus may be considered an existential threat to Tor. In a WF attack, an adversary that can observe a Tor user's connection entering into the Tor network can first train machine learning (ML) models to recognize the network traffic patterns associated with requests of a website, and then use the models to predict when the user accesses that website. WF attacks work against encrypted traffic and generally only require the series of packet directions (and sometimes timestamps) that result from a website request in order to accurately predict the website and deanonymize the user.

A major problem in the study of WF is that a lack of explainability of WF prediction results leads to an incomplete understanding of the severity of the threat facing the Tor network. A lack of explainability is due to two related but distinct forms of uncertainty introduced in the WF evaluation methodology. First, the application of *black-box ML models* that are trained on the website prediction task yields decisions that cannot be fully explained. Although some progress has been made, e.g., by evaluating feature importance [17, 23, 40], model explainability is still in general an open problem in ML [11]. Second, WF studies have been conducted primarily using *low-quality datasets* collected using automated browsers from the live Tor network. Automated browsers have been criticized for being overly synthetic and unrealistic across several axes [28, 38]. However, collecting WF datasets from the live Tor network may be an even greater contributor to uncertainty since researchers do not account for their lack of control over the Tor network during data collection. Tor is a large and diverse network with an extremely large number of variables that are continuously changing in unknown ways; e.g., inaccurate load balancing [22], CPU bottlenecks [7], port exhaustion [16], DoS attacks [26, 27], censorship events [4], natural relay churn [2], and Tor version updates [3] can all lead to significant performance effects and relay congestion, which has been shown to alter traffic patterns and WF conclusions [40]. Thus, without a controlled data collection process, WF evaluations will continue to produce questionable results.

In this paper, we explore how network simulation can be used to increase our control over the collection of WF datasets and thus improve our ability to understand WF results. Our main insight is that, thanks to recent advances in Tor network simulation methodology and architecture [24, 25], we can now design a *controlled* WF evaluation methodology by running web clients and servers directly inside of a deterministic Tor simulation in Shadow [21, 24]. This paper focuses on the study of the following research question: *How does Tor network composition and performance affect our ability to train and test WF classifiers?* To answer this question, we address the following three subquestions.

RQ1: How can WF attacks be simulated in Shadow? We need to be able to understand Shadow's fidelity in order to properly contextualize WF on simulated data. To measure fidelity, in § 3 we investigate and quantify the extent to which Shadow can reproduce the traffic characteristics associated with Tor Browser webpage fetches. We measure and compare traffic in the live Tor network

Proceedings on Privacy Enhancing Technologies 2023(4), 1–19
Distribution Statement A. Approved for public release: distribution is unlimited.
https://doi.org/XXXXXXX.XXXXXXX

and in Shadow and find a high degree of similarity among common WF features. In a multiclass closed-world experiment, we find that a classifier trained entirely in Shadow can classify traffic collected on the live Tor network with greater than 86% accuracy, and that a classifier trained on the live Tor network can classify traffic collected in Shadow with at least 88% accuracy. Overall, we find that Shadow is a viable tool for evaluating WF attacks.

RQ2: How sensitive are WF classifiers to changing network composition and performance? The Tor network continuously changes in composition and performance [1, 2]. Using WF data collected from 18 Shadow simulations, in § 4 we investigate how changes in network effects such as relay congestion influence the extent to which an adversary can accurately classify webpage visits. We find that the WF performance of classifiers that are trained on an open-world binary classification task using data from networks with low congestion and then transferred to networks with high congestion significantly degrades: across four state-of-the-art WF classifiers, three congestion levels, and three distinct networks, we find that true- and false-positive rates may be misstated by as much as 106% and 367%, respectively. We also find that training in networks with different relay compositions but with similar congestion introduces error in the true- and false-positive rates by as much as 41% and 700%, respectively. Overall, our results indicate that WF evaluations are significantly affected by relay congestion and will be more informative when controlling for congestion.

RQ3: How can WF classifiers be made more robust to changing network composition and performance? We seek to better understand how classifiers might be made more robust to changing congestion levels so that we can (1) develop better defenses against such robust classifiers, and (2) improve our WF threat estimates to better prioritize future work. To this end, in § 5 we investigate the extent to which WF classifiers can be augmented using data gathered from simulations of networks with varying degrees of congestion. Across our tested classifiers and network variations, we find that classifiers trained on an open-world binary classification task are more robust to congestion and can improve true- and false-positive rates by as much as 19% and 83%, respectively. Additionally, we find that we can classify real-world Tor traffic with greater than 80% accuracy using classifiers that are *exclusively trained using network simulation*. Overall, our results indicate that network simulation is a viable strategy for improving WF classifier generalizability.

We argue that novel advancements in methods to increase the quality of WF datasets (i.e., model inputs) will lead to meaningful improvements in our ability to explain WF results. We demonstrate in this paper that network simulation is a viable strategy for improving our scientific understanding of the causal relationships between network changes and WF performance, and that this can be done in a simulation environment that is completely private, safe, and controlled. We *also* believe that genuine WF datasets [13] will ultimately improve estimates of the severity of WF as a threat against the live Tor network, and hope that future work will consider safely gathering such data.

Contributions. We summarize the primary novel contributions of our work as follows:

- We demonstrate Shadow's fidelity in simulating WF attacks and find similar classifier performance when training and testing in both live and simulated Tor networks.
- We establish the first methodology for running end-to-end WF evaluations using data collected during network simulation.
- We confirm and greatly extend results from prior work [40] finding that relay congestion can have a strong negative effect on classifier performance.
- We are the first to discover that WF classifiers exclusively trained on simulated data can be made more robust to variations in network composition and relay congestion, and can be transferred to the real world with high accuracy.

2 BACKGROUND

2.1 Tor

Tor Browser, a hardened fork of Firefox, is a popular tool for accessing websites through the Tor network. Embedded inside of Tor Browser is the Tor client software that (1) exposes a SOCKS proxy to enable communication with the browser, and (2) forwards browser requests for resources through the Tor network to external internet services that need not be aware of Tor. The Tor client forwards the browser requests and other data from peer connections, generally called *streams*, through long-lived cryptographic tunnels called *circuits* that each contains a series of three Tor relays: an *entry*, a *middle*, and an *exit*. A circuit is created such that the entry may observe the client but not the destination, and the exit may observe the destination but not the client; thus, Tor clients are said to be unlinkable (i.e., anonymous) to the destinations they visit.

An application may create a number of streams through Tor that connect to various internet destinations, and the Tor client software must decide how to multiplex these streams over circuits. Isolating each stream to its own circuit would cause significant cryptographic overhead, enable malicious servers to cause a client to build many circuits, and increase a client's exposure to compromised relays. Thus, Tor prefers to group and multiplex related streams over the same circuit. The stream isolation strategy used in Tor Browser is to group all first and third party streams with respect to the first-party domain that appears in the URL bar, and isolate groups of streams with distinct first-party URL bar domains to separate circuits.

To hinder traffic analysis, all data sent through Tor is packaged into *cells* which are padded to a fixed size of 514 bytes and encrypted once for each circuit relay. The padded cells limit information leakage by adding some protection to the amount of application data being forwarded. However, the number of cells and their directionality and timing can still be observed by each relay in the circuit, enabling website fingerprinting attacks against Tor users.

2.2 Website Fingerprinting

In a *website fingerprinting* (WF) attack, an adversary that can observe communication between a Tor client and its entry relay can use traffic metadata to predict the website visited by the client. In this paper, we consider an adversary that runs an entry relay and therefore has access to circuit and cell metadata, i.e., the timing and directionality of cells sent through a circuit. However, note that prior work has demonstrated that WF is still effective for adversaries that do not have access to Tor metadata [48].

We assume that the adversary can collect metadata about the circuits and cells it observes. We define a cell as the pair (t_i, d_i) where i is the cell's position in the ordered sequence of cells that

have been forwarded through a circuit since it was created, t_i is the time that the ith cell was observed relative to circuit creation, and $d_i \in \{-1, 1\}$ is the direction in which the ith cell was forwarded (-1 indicates the cell was sent by the client, 1 indicates it was received by the client). We define a *cell trace* C as $\langle (t_i, d_i) \text{ or } \emptyset \rangle_{i=1}^{N}$, i.e., a fixed-length vector of N components where $C[i] = (t_i, d_i)$ if cell i was observed and $C[i] = \emptyset$ if the data stream was shorter than i cells. We use N=5,000 for consistency with prior work.

We assume that the adversary defines a finite set of webpages that are considered sensitive, and that it wants to detect when and which sensitive webpages are visited (e.g., for the purpose of censorship, targeted surveillance, or tracking). To accomplish this goal the adversary will train machine learning classifiers, which will require a training set of labeled cell traces. We define the following sets of labels, e.g., webpage URLs:

W_α: webpages that the adversary labels as sensitive
W_β: webpages that the adversary labels as benign
$W_\emptyset$: unlabeled webpages (they are $\in$ world, but $\notin W_\alpha$ and $\notin W_\beta$)

The adversary will train machine learning classifiers on two tasks. First, it will train a classifier in the *binary* classification setting to predict when a given cell trace is either sensitive (in W_α) or not. Second, for cell traces that are predicted as sensitive, it will train a classifier in the *multiclass* classification setting to predict the label $w \in W_\alpha$ corresponding to the cell trace. Training these classifiers requires that the adversary collects labeled training data, e.g., by visiting the webpages in W_α and in W_β using Tor Browser and its own entry relay, recording the cell traces for each webpage visit, and labeling each cell trace with the visited URL. Once trained, the adversary can test the classifiers against unlabeled cell traces observed when users visit webpages using its relay. We highlight that webpages $w \in W_\emptyset$ are not considered by the adversary during training, but may be presented during testing; we consider the effect of such pages in our evaluations in § 4 and § 5.

2.3 Machine Learning Classifiers

2.3.1 Supervised Learning. Given a dataset $D = \{(X_i, y_i)\}_{i=1}^{n}$ of examples $X_i \in \mathbb{R}^M$ and corresponding labels $y_i \in \mathbb{Z}^+$, the goal of supervised learning is to learn the parameters θ of a parameterized classifier $f_\theta(X) = \hat{y}$ that minimizes loss over the training set:

$$\min_\theta \sum_i L(y_i, f_\theta(X_i)),$$

where L is a loss function that penalizes misclassification.

The learned classifier can then be used to predict the label of a new example X' for which the label is not necessarily known. To assess the degree to which f_θ generalizes to new examples, the training dataset is partitioned into two disjoint subsets: $D_{\text{Train}} = \{(X_i, y_i)\}_{i=1}^{j}$ and $D_{\text{Test}} = \{(X_i, y_i)\}_{i=j+1}^{n}$. Then, the model parameters are learned using only D_{Train}. Generalization error is then determined by applying the classifier to examples in D_{Test} and comparing the known label to the predicted label.

2.3.2 Performance Metrics. Depending on the experiment, different measurements are used to assess the quality of the learned classifier (i.e., the severity of the adversary's attack). In a balanced setting, where each label is represented in the dataset D roughly an equal number of times, accuracy (Acc) can be used:

$$\text{Acc} = \frac{\mathcal{I}(y_i, \hat{y_i})}{|D_{\text{Test}}|}$$

where $\hat{y_i} = f_\theta(X_i)$ is the predicted label for $(X_i, y_i) \in D_{\text{Test}}$ and $\mathcal{I}$ is an indicator function that outputs 1 if its arguments are equal and 0 otherwise. Accuracy represents the fraction of predictions that were made correctly by the classifier.

Accuracy is not appropriate for use in an imbalanced setting, when the distribution of labels in the test set is not uniform (in this setting, high accuracy can be achieved by a trivial and useless classifier that always predicts the most frequently occurring label). In the imbalanced setting, *true-positive rate* and *false-positive rate* are more appropriate measures of classifier performance. These rates are defined over the total number of the four possible classification outcomes (assuming binary classification):

Outcome	Notation	True Label y	Predicted Label $\hat{y}$
True positive	TP	1	1
True negative	TN	0	0
False positive	FP	0	1
False negative	FN	1	0

The metrics are then defined as follows:

True positive rate (Recall or TPR): Defined as $\text{TP}/(\text{TP}+\text{FN})$. Recall indicates the rate at which monitored examples are correctly detected by the adversary.

False positive rate (FPR): Defined as $\text{FP}/(\text{FP}+\text{TN})$. FPR is the rate at which benign examples are falsely flagged as belonging to the monitored set.

Some classifiers output confidence scores associated with the label, i.e., $f_\theta(X_i) \mapsto (\hat{y_i}, c_i)$ for a confidence value $c_i \in [0, 1]$. Classifiers with this property may be tuned according to confidence. For example, any classification made with too-low confidence could be "discarded" and assigned a default label (e.g., 0) regardless of the predicted label. Different confidence thresholds yield different true and false positive rates; the set of all possible true and false positive rates for a classifier are what defines its receiver operating characteristic (ROC) curve. In this work, when applicable, we just use the standard decision threshold of 0.5[1] and do not consider metrics exploring the trade off between TPR and FPR, such as the area under the ROC curve. Using TPR and FPR—as opposed to using ROC curves—allows us to consistently compare all predictive models, not just those that output confidence values.

2.3.3 Classical and Deep Learning. We will consider both classical machine learning models and more recent deep learning models.

Classical machine learning models, such as decision trees or support vector machines, have low complexity in comparison to their deep learning counterparts. *Feature selection* is a characteristic requirement of classical machine learning. Feature selection is the process of taking a cell trace C and producing a corresponding feature vector $X \in \mathbb{R}^M$ for input to the classifier. Each feature should encode some information about the relationship of labels and cell traces. For example, the components of X could incorporate summary statistics about C such as the average rate of cell transmission

[1] In the multi-class setting, the label with maximal confidence is used instead.

or the total number of cells sent between the communicating parties. Determining an informative set of features requires expert domain knowledge and is error-prone—it is difficult for experts to precisely define all of the features that are useful for prediction. Moreover, for classical models to work well, the number of features used must be kept limited [8].

In contrast, deep learning models, such as convolutional neural networks, are highly complex and often contain millions of parameters. A benefit of deep models is that they do not require feature selection but instead take as input "raw" features, such as the vector of packet directions $\langle d_i \rangle_{i=1}^{M}$. During training, neural networks automatically learn a set of features important for classification in the hidden layers of the model.

In addition to the advantage of automatic feature selection, deep learning models tend to outperform partially due to their higher capacity to encode complex relationships between examples and labels. Classical methods are not totally obviated, however. Classical methods require less data to train, are more interpretable, and can produce a prediction using fewer steps of computation.

2.4 Collecting Cell Traces

In order to evaluate the effectiveness of WF attacks, we require a method for collecting and labeling cell traces. We follow the approach outlined above in § 2.2 in which an entry relay records cell metadata from circuits it observes. Collecting entry metadata requires modifications to the Tor software, but enables us to use an identical collection process in the live Tor network and in Shadow.

Our Tor metadata collection process follows the design established by Cherubin et al. [13]. When our client creates a circuit through our entry relay, the client sends a new, custom cell type to the entry to indicate that the circuit is ours and to communicate the webpage label that is being accessed though the circuit. The entry relay then marks the circuit for measurement, records the label, and begins recording a cell trace for the circuit by appending cell metadata items to a list whenever a cell is forwarded in either direction (cell traces are defined in § 2.2). The relay stops recording the cell trace when either it has observed N cells (recall, we use N=5,000) or the circuit is closed, whichever occurs first. The relay then exports the labeled cell trace in an asynchronous event emitted through Tor's control interface. Cell traces can be collected using `stem` or any other control client that registers for the events.

2.5 Shadow

Shadow is a discrete-event network simulator and the state-of-the-art tool for running large-scale Tor network simulations [21]. Shadow's core architecture was recently redesigned, and as of `v2.0.0` it now directly executes unmodified applications as individual Linux OS processes and co-opts them into the simulation environment using system call interposition and efficient shared-memory data transfer mechanisms [24]. As a result of the new architecture, Shadow is able to directly execute unmodified applications without rebuilding them and benefits from improved compatibility, correctness, and maintainability. The applications are connected through a simulated network stack that includes implementations of communication protocols such as TCP and UDP as well as other networking, event notification, and file descriptor facilities that Linux normally provides through its system call interface. We use Shadow in this paper to run large-scale Tor network simulations following recent improvements in Tor network experiment design and analysis [25]. We primarily use existing tools to set up and run our Tor experiments and we make only slight modifications for the purpose of generating and gathering WF data to use for subsequent training and testing of machine learning classifiers.

3 FIDELITY

In this section, we investigate RQ1: *How can WF attacks be simulated in Shadow?* We seek to answer this question by quantifying the extent to which Shadow can reproduce the traffic characteristics associated with Tor Browser webpage fetches. Quantifying Shadow's fidelity will enable us to contextualize the WF evaluation and analysis presented throughout the paper. We describe our measurement methodology and the data we generated before quantifying Shadow's fidelity in terms of traffic features, simulated network performance, and a closed-world WF analysis.

3.1 Methodology

Our ultimate goal is to quantify the fidelity of simulating Tor network webpage fetches in Shadow. Below we describe the set of webpages that we fetch, the web server and client software we use, and the measurement tasks we perform.

3.1.1 Web Server. We choose an identical set of webpages that we will serve for measurement in both the live Tor network and in a Shadow-simulated Tor network to guarantee webpage and server consistency across measurements. We use the `zimply` python module to mirror the "all maxi" version of the English Wikipedia from 2023-01.[2] The mirror contains a total of 23,264,812 pages that `zimply` makes available for download through an HTTP server. We form a subset W_α of all pages available in this mirror by conducting 10 random walks that each starts at the top-level index page of the mirror. For each walk, we add a random subpage from the index page into W_α and then fetch the subpage. We choose a random page linked on the subpage, add it to W_α and fetch it, and repeat recursively until we reach a recursion depth of 10. After 10 random walks, our set W_α contains 98 unique pages that we deem as "sensitive" for the adversary (see § 2.2) and use for measurement and evaluation throughout the paper.

3.1.2 Web Client. Using Tor Browser to fetch the pages in W_α in both the live Tor network and in a Shadow-simulated Tor network would lead to the most direct comparison between the real and simulated networks. When fetching webpages in the live Tor network, we use tor-browser-selenium (TBS) as a surrogate for Tor Browser. TBS has become the standard tool for collecting data for website fingerprinting studies; it is a python wrapper around the `selenium` module and is used to automatically configure and drive the instance of Firefox that is embedded inside of Tor Browser. Thus, we assume that Tor Browser and TBS produce identical results when identically configured.

Unfortunately, our attempts to run major web browsers in Shadow (including Firefox and Chromium) failed due to missing system calls (e.g., `fork()` and `exec()` were not yet supported by Shadow at the

[2] Available at https://dumps.wikimedia.org/other/kiwix/zim/wikipedia/

time of writing). Moreover, web browsers are resource-intensive tools and a lighter-weight solution is desirable since we intend to run large-scale Shadow-simulated Tor networks with thousands of clients and relays. Thus, our methodology considers the use of `wget2` as a surrogate for Tor Browser when simulating webpage fetches in Shadow.[3]

3.1.3 Measurement Tasks. We design three measurement tasks to help us quantify the fidelity of running `wget2` in a Shadow-simulated Tor network as a surrogate for running Tor Browser (i.e., TBS) in the live Tor network:

Task 1: fetch W_α with TBS in the live Tor network;
Task 2: fetch W_α with `wget2` in the live Tor network; and
Task 3: fetch W_α with `wget2` in a Shadow-simulated Tor network.

We identically configure the `zimply` Wikipedia mirror from § 3.1.1 in all tasks for consistency across measurements.

Task 1: TBS → live Tor network. During this measurement, we make two major sets of configuration changes to the default configuration that is set by TBS. First, we configure all of the Firefox preferences that Tor Browser normally sets in order to run at the "Safest" Security Level; this includes disabling JavaScript, some fonts, and HTML5 media. Although recent work has found that changes in the security level do not meaningfully reduce attack performance [33], we found that the "Safest" level does improve browser consistency across repeated fetches of the same page. Second, we configure Firefox to disable several preferences associated with HTTPS-first and HTTPS-only. Because `zimply` serves the Wikipedia pages over HTTP, the HTTPS options result in errors and timeouts that extend the overall measurement time.

We run TBS and the `zimply` Wikipedia mirror on a dedicated machine in Chameleon Cloud, an experimental platform offering access to bare-metal machines for computer science systems research [29]. TBS is configured to send its requests through a locally running Tor client process, out through the Tor network, and then back to a locally running `zimply` Wikipedia mirror. The Tor client is configured to isolate each webpage fetch from our server to a new circuit and to pin a Tor relay under our control (running on a dedicated machine in New York, USA) as the entry relay for all circuits it builds. Our pinned entry relay collects cell traces from our client's circuits following the approach described in § 2.4.

Task 2: `wget2` → *live Tor network.* During this measurement, we follow the same methodology as in Task 1 except that we use `wget2` instead of TBS to fetch the pages in W_α. `wget2` includes support for fetching webpages through an HTTP proxy, but not through a SOCKS proxy such as Tor. Thus, we slightly extend `wget2` to add SOCKS client support, and use the modified version to fetch the pages in W_α through our Tor client and relay. We configure `wget2` to fetch all embedded objects on each page (`--page-requisites`) concurrently (`--max-threads=30`) and to set the same user agent that is set by TBS. Finally, we configure `wget2` to filter out some resource URLs (`--reject-regex=/w/|\.js$`), and verified through local testing that our TBS and `wget2` configurations download an identical set of resources from the `zimply` server.[4]

Task 3: `wget2` → *Shadow-simulated Tor network.* The goal in this task is to mirror the measurement process from Task 2 inside of Shadow. To do this, we first use recently published modeling tools (i.e., `tornettools`) [25] to produce a private Tor network configuration suitable for Shadow using Tor metrics data from 2023-01. We generate a Shadow configuration that represents the live Tor network at a scale of 25%;[5] it contains 1,536 relays and 2,169 traffic generation processes that create 746,757 circuits every 10 minutes and emulate the simultaneous traffic load of 188,085 users. We extend the generated configuration to add hosts running the `zimply` Wikipedia mirror and the `wget2` processes (one for each webpage fetch). As in Tasks 1 and 2 above, our relay (now running in Shadow) is pinned as the entry relay for the Tor client used by `wget2`, and the pinned entry collects cell traces as described in § 2.4.

3.1.4 Datasets. We use an identical process and format for collecting and storing cell traces from our Tor entry relay, and consistently process the data collected during our three measurement tasks. During each measurement task, we fetch each of the 98 unique pages in W_α 200 times. However, we observe slightly more than 200 circuits per page for some pages in W_α, which we believe may be due to errors and retries in TBS, `wget2`, or Tor. Thus, we apply the random sample consensus (RANSAC) method [15], a standard outlier detection technique, with a simple k-Nearest Neighbors model to find and eliminate outliers. Each trace is preprocessed into a number of features for input to the k-nearest neighbors model: we use the first five features defined in § 3.2 along with the maximum number of cells of a burst in the trace, and the CUMUL representation of the trace at the 25th, 50th, 75th, and 100th indicies [36]. Each of these features has been shown in prior work to be informative characteristcs of traces [17, 36, 47]. The learned model is used to prune the data from each measurement task such that no more than 200 cell traces are available for each page in W_α. We also remove all cells that the entry relay identifies as Tor control cells, since those do not carry web payloads. We define the resulting, cleaned datasets of cell traces as:

$D(\text{tbs, tor})$: 200 cell traces $\forall w \in W_\alpha$ from Task 1
$D(\text{wget2, tor})$: 200 cell traces $\forall w \in W_\alpha$ from Task 2
$D(\text{wget2, shadow})$: 200 cell traces $\forall w \in W_\alpha$ from Task 3

where $D(\cdot)[i, j]$ is the cell trace from the jth fetch of the ith page.

3.1.5 Ethics. The measurements described in this section involve use of the live, public Tor network. We took several steps to limit the impact of our measurements on the network and its users. We limit our measurement to the small set W_α of 98 pages and we use our own Tor clients to create measurement circuits and fetch these pages: in total, we create approximately 40k circuits to fetch 40k pages. The load we add to the network is low relative to the billions of circuits Tor is estimated to handle every day [32] and the 300 Gbit of traffic handled every second [1]. Still, we spread our measurement load over the course of several days to limit the amount of overhead we add to the network at any one time, and we use our own entry relay for all circuits we create. Our relay only collects cell traces from circuits created by clients under our control, which is enforced by the use of unique Tor control cells

[3] `wget2` offers significant improvements over `wget`, most notably including the simultaneous download of multiple resources using multi-threading.
[4] Each tool issues one unique request not issued by the other—TBS for `favicon.ico` and `wget2` for `robots.txt`—each of which results in an HTTP 404 error from `zimply`.
[5] We used the `tornettools` setting `--load_scale=2` to create a configuration whose Shadow-simulated performance is representative of Tor during 2023-01 (see § 3.3.1).

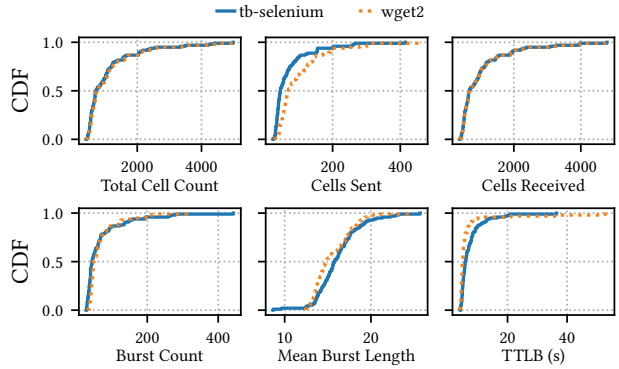

Figure 1: The cumulative distribution of the per-page median feature values, as defined in Eqn. 1, for the six feature functions shown in the subplots. The feature values from `wget2` closely align with those from TBS.

sent between our client and relay. Finally, although the method of fetching webpages over Tor has been peer-reviewed many times, a high-level goal in this work is to establish a new method for evaluating WF using Shadow simulations that are safe, private, and add absolutely no additional overhead or risk to Tor or its users.

3.2 Browser Fidelity

In this section we evaluate the fidelity of `wget2` as a surrogate for TBS; that is, we evaluate the differences between the cell traces in $D(\text{tbs, tor})$ and those in $D(\text{wget2, tor})$. We quantify the differences using a basic set of features that are commonly used in WF literature, i.e., functions $f(C) \mapsto v$ that each takes as input a cell trace C and returns as output a single value v:

Total Cell Count: the total number of cells in a trace.
Cells Sent: the number of cells sent by the client.
Cells Received: the number of cells received by the client.
Burst Count: the number of cell bursts in either direction.
Mean Burst Length: the mean number of cells in a burst.
Time to Last Byte (TTLB): the elapsed time between the client sending the first cell and receiving the last cell.

A cell burst is a cell trace subsequence in which all consecutive cells have the same direction (i.e., all are sent or received); a burst ends when a cell is observed with a direction that is opposite of the direction of the previous cell.

We analyze the cell traces in $D(\text{tbs, tor})$ and in $D(\text{wget2, tor})$. We compute the feature functions described above for each cell trace in each dataset, and collapse the 200 feature values into a single value for each page i by computing the median:

$$F^i_{(\text{tool,tor})} \leftarrow \text{Median}\left(\forall j : f(D(\text{tool, tor})[i, j])\right) \quad (1)$$

where f is a feature function and *tool* is either *tbs* or *wget2*. We plot in Fig. 1 the cumulative distributions $\text{CDF}\left(\forall i : F^i_{(\text{tbs,tor})}\right)$ and $\text{CDF}\left(\forall i : F^i_{(\text{wget2,tor})}\right)$ for each feature (each CDF is calculated over 98 median values). The subplots show remarkable similarity in the distributions of the feature medians when comparing TBS to `wget2`. The largest discrepancy is due to the *Cells Sent* and *TTLB* features: `wget2` tends to send more cells than TBS with higher variance in download times, the latter of which may be an expected consequence of variable circuit performance.

We further analyze the features on a per-page basis to supplement the distribution analysis. We compute the absolute difference between the per-page medians for each feature:

$$F^i_{\text{diff}} \leftarrow F^i_{(\text{tbs,tor})} - F^i_{(\text{wget2,tor})} \quad (2)$$

We analyze $\text{CDF}(\forall i : F^i_{\text{diff}})$ for all features and observe that `wget2` sends between roughly 1 and 50 more cells than TBS over the entire life of the circuit, and that *TTLB* no longer stands out as particularly inaccurate. (See Fig. 6 in Appendix A for more details.)

Finally, we compute 99% confidence intervals for each of the 98 pages over all 6 features (by replacing median in Eqn. 1 with mean and standard deviation). We find that the TBS and `wget2` CIs overlap in 233/588=40% of cases, indicating that fetching the same page with the same tool multiple times may sometimes produce as much variability as fetching the page with the other tool. (See Fig. 7 in Appendix A for more details.)

Key Takeaway. We conclude from our analysis that the fidelity of `wget2` as a surrogate for TBS is acceptable with respect to the cell traces used for WF analyses and for the purpose of enabling controlled WF evaluations using network simulation.

3.3 Simulation Fidelity

In this section, we quantify the fidelity of Shadow in simulating `wget2` fetches through a Tor network. We first conduct a performance analysis to characterize the network performance of our Shadow-simulated Tor network relative to the live Tor network, and we then conduct a website fingerprinting analysis using $D(\text{wget2, tor})$ and $D(\text{wget2, shadow})$.

3.3.1 Network Performance. We run Shadow simulations using the configuration described in Task 3 in § 3.1.3. We run six repeated trials using Shadow `v2.4.0`, Tor `v0.4.7.10`, and `tornettools` at commit `75e59fa`; we use unique Shadow seeds for each simulation following the state-of-the-art Tor experimentation methodology [25]. We run the simulations using a blade server cluster in which each blade contains identical hardware: 1 TiB of RAM and 2×18 core Intel Xeon Gold 6354 CPUs (36 total cores and 72 total hyperthreads). Each blade is configured to run a minimal version of Debian 11 with Linux kernel `v5.10.0`, and the simulations are run in containers using Singularity [30]. In each simulation, `wget2` is used to repeatedly fetch webpages through the Shadow-simulated Tor network for 25 simulated minutes to collect data for $D(\text{wget2, shadow})$, mirroring the process that was used to collect $D(\text{wget2, tor})$ on the live Tor network. In addition, several performance metrics are collected and parsed by `tornettools` and are available for analysis.

Across the six trials we find that the maximum RAM used by Shadow was 685 GiB, which includes the memory used by the `tor`, `wget2`, and traffic generator processes running during the simulation (recall that we simulate the Tor network at a scale of 25%). We find that each of the six trials completed in fewer than 28 hours. The performance characteristics of the simulated network across a variety of client performance metrics are shown in Fig. 2. Because `tornettools` measures the same metrics as those that are

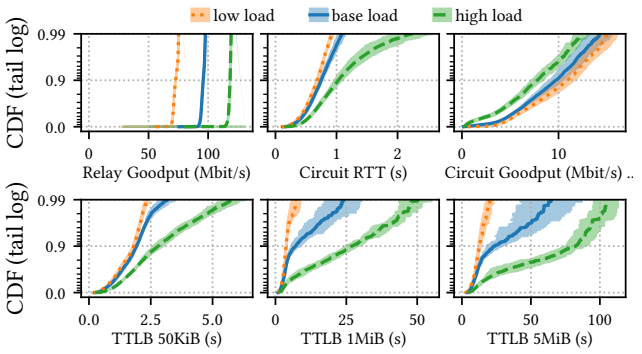

Figure 2: Client performance in the live Tor network in 2023-01 compared to our Shadow-simulated Tor network. The shaded regions show 95% CIs computed over 6 trials [25].

gathered in the live Tor network [1], we are able to directly compare the Shadow-simulated performance to the live Tor performance. Overall, we find that the long-tail performance in Tor is worse than in Shadow, perhaps because Shadow does not reproduce all possible network failures. However, the *expected* performance is much more accurate than the *long-tail* performance: for example, the median circuit build time is 2.33 seconds in both Shadow and Tor, the median circuit goodput is 9.84 Mbit/s in Shadow and 10.8 Mbit/s in Tor, and the median TTLB for 1 MiB downloads is 8.32 seconds in Shadow and 11.6 seconds in Tor.

Key Takeaway. Based on our performance analysis and our outlier detection and removal process from § 3.1.4, we expect minor impact due to fidelity on the cell traces collected under simulation.

3.3.2 Website Fingerprinting. With this next analysis, we aim to establish that cell traces generated using `wget2` with Shadow accurately represent cell traces generated using `wget2` with the live Tor network in a WF attack evaluation. Ideally, traces generated using Shadow should be indistinguishable from traces collected on the live Tor network, but in practice Shadow simulations do not completely capture the every intricate complexity of the live network. Hence, we assess trace similarity using the ultimate task of WF classification. We set up four experiments:

– *Same dataset for training and testing:*
  (1) Train and test classifiers using traces from $D(\text{wget2, tor})$.
  (2) Train and test classifiers using traces from $D(\text{wget2, shadow})$.
– *Cross datasets for training and testing:*
  (3) Train classifiers using traces from $D(\text{wget2, tor})$ and test using traces from $D(\text{wget2, shadow})$.
  (4) Train classifiers using traces from $D(\text{wget2, shadow})$ and test using traces from $D(\text{wget2, tor})$.

If the Shadow cell traces are representative of the live network traces, two properties should hold. First, classification performance in the same-dataset experiments should be similar—it would signal modeling inconsistencies if WF attacks succeeded on the live network or in Shadow but did not succeed in the opposite setting. Second, classification performance in the cross dataset experiments should not be extraordinarily degraded with respect to the same dataset experiments. If examples from $D(\text{wget2, shadow})$ and $D(\text{wget2, tor})$ are similarly distributed, then a classifier trained on one dataset should accurately label examples from the other.

Table 1: Multiclass closed world classification accuracy for `wget2` cell traces collected from the live Tor network and from our Shadow-simulated Tor network.

Train →	$D(\text{wget2, tor})$				$D(\text{wget2, shadow})$			
Test ↓	CUMUL	k-FP	DF	TT	CUMUL	k-FP	DF	TT
$D(\text{wget2, tor})$	0.97	0.97	0.97	0.97	0.86	0.83	0.76	0.61
$D(\text{wget2, shadow})$	0.93	0.88	0.95	0.92	1.0	1.0	1.0	0.99

We carry out this experiment with four WF classifiers, representing both classical and deep learning approaches: (1) CUMUL [36], (2) k-Fingerprinting (k-FP) [17], (3) Deep Fingerprinting (DF) [42], and (4) Tik-Tok (TT) [40]. (Complete details regarding these classifiers are given in § 4.3.) CUMUL uses a support vector machine and k-Fingerprinting uses a random forest classifier to perform classification. Both attacks are implemented using high-level statistics as the features of each cell trace, including those described in § 3.2. Deep fingerprinting and Tik-Tok are implemented using a deep neural network that takes cell traces as input; the DF attack uses cell direction, whereas the TT attack uses cell directions and times.

The classifiers are evaluated using a typical multiclass closed world classification experiment. The classification task is to predict which page a trace belongs to of the 98 pages constituting W_α (recall from § 3.1.4 that $D(\text{wget2, tor})$ and $D(\text{wget2, shadow})$ contain 200 cell traces for each of the 98 pages). We randomly partition each dataset into a 60% training set and 40% testing set. The train-test split is stratified by webpage so that 160 traces from each webpage appear in the training set and 80 traces from each webpage appear in the test set. Then, each classifier is trained on either the $D(\text{wget2, tor})$ or $D(\text{wget2, shadow})$ training sets and tested on both test sets. Because each class appears equally frequently in the test set, we use accuracy to measure classification performance.

The results of this experiment are shown in Table 1. The reported accuracy values are between 0 and 1, where maximal accuracy indicates perfect webpage prediction. A classifier that outputs random guesses achieves an accuracy of $1/98 \approx 0.01$ in expectation. When the same dataset is used for training and testing, all classifiers achieve near-perfect accuracy, and hence we have consistency as was desired. The cross-dataset experiment results also conform to the desired outcome: classification performance is relatively high, even when the classifier is tested on a different source of data than that on which it was trained. The strength of this result is worthy of emphasis: *we find that a classifier trained entirely in simulation can classify traces collected on the live Tor network with greater than* 85% *accuracy (CUMUL).* Note that we do find that training on live network data and testing on Shadow data produces higher accuracy than the converse case. We suspect this effect is due to higher intra-page trace variance produced on the live Tor network, whereas Shadow simulations do not perfectly reproduce all sources of variance (see Fig. 2).

Key Takeaway. Our findings show that Shadow can be used to produce Tor cell traces that are well within the distribution of traces that are produced on the live Tor network. We conclude that it is

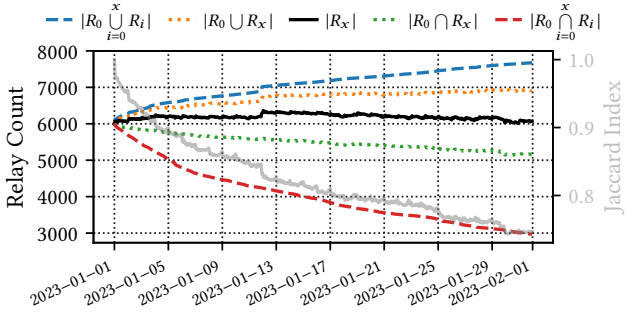

Figure 3: Relay churn over all 744 consensuses from 2023-01. R_0 is the set of relays present in the first consensus on 2023-01-01, and R_x is the set of relays in subsequent consensuses. The Jaccard index, a measure of set similarity, is defined as $|R_0 \cap R_x|$ / $|R_0 \cup R_x|$ and plotted in grey against the right y-axis.

viable to implement and evaluate WF attacks using data collected from Shadow simulations.

4 SENSITIVITY

In this section, we investigate RQ2: *How sensitive are WF classifiers to changing network composition and performance?* We seek to answer this question by studying WF in a variety of unique network environments to better understand how network effects such as relay congestion influence the extent to which an adversary can accurately classify webpage visits. We describe (1) the methodology we used to collect cell traces across various network conditions in Shadow, (2) an analysis of the cell traces, and (3) an evaluation of both closed and open world WF attacks.

4.1 Methodology

Our ultimate goal is to evaluate the extent to which WF classifiers that are trained using data collected under certain network conditions can be transferred and applied under differing conditions. To accomplish our goal, we use Shadow to generate several Tor networks with distinct properties, collect cell traces in each network, and use the collected data in WF evaluations.

4.1.1 Tor Network Configurations. We again use `tornettools` to generate Tor network configurations that are suitable for running in Shadow [25]. Due to resource limitations, and to remain consistent with § 3, we generate all networks at a scale of 25% of the live Tor network using Tor metrics data from 2023-01. When generating the network configurations, we vary the network conditions in two ways: (1) we re-randomize the selection of relays to use in each network; and (2) we increase and decrease the traffic load generated in each network to adjust relay congestion.

Re-randomizing Relay Selection. `tornettools` offers the ability to deterministically generate network configurations, or to change the random seed in order to re-randomize the selection of relays that will appear in the simulated Tor network. Although the live Tor network consisted of between 6,000 and 6,500 relays during the month of January, 2023 [1], Fig. 3 shows that there is considerable relay churn during this period: (1) ~2,000 new relays joined during the month (blue dashed line), ~1,000 of which were still present at the end of the month (orange dotted line); (2) ~3,000 relays left at some point during the month (red dashed line), ~1,000 of which did not come back (green dotted line); and (3) the Jaccard index was 0.75 at the end of the month, indicating a 25% turnover in relays. When building networks, `tornettools` will consider all relays that were present sometime during 2023-01. Thus, we run `tornettools` with three unique seeds in order to generate three independent networks with a distinct set of relays selected to participate in each. This relay variation represents the natural fluctuation of the live network during a typical month.

Adjusting Relay Congestion. `tornettools` offers a network generation option (`--load_scale`) to allow us to increase or decrease the circuit and stream creation rate of the background traffic generators, which in turn will generally increase or decrease relay congestion. For each of the three random seeds, we generate a network with each of the following three load profiles:

Base Load: baseline load from § 3.3.1 (`--load_scale=2`)
Low Load: 25% less load than baseline (`--load_scale=1.5`)
High Load: 25% more load than baseline (`--load_scale=2.5`)

The generation process produces a total of nine distinct network configuration files that can be run in Shadow; to each of these nine configurations we add web clients and servers in order to collect cell traces while each network is being simulated.

4.1.2 Web Clients and Servers. In our WF experiments, we will require cell traces for each of the three types of webpage sets described in § 2.2: labeled sensitive pages W_α, labeled benign pages W_β, and pages unlabeled by the adversary $W_\emptyset$. For the sensitive set of webpages, we use the same W_α defined in § 3.1.1. Additionally, we form another set of webpages V following the same random walk procedure described in § 3.1.1, this time conducting 100 independent walks each of depth 1,000 and then removing duplicates and pages from W_α. We find that $|V|$=67,718. We then randomly split V into two subsets W_β and $W_\emptyset$ such that $W_\beta \cup W_\emptyset = V$, $W_\beta \cap W_\emptyset = \{\emptyset\}$, and $|W_\beta| = |W_\emptyset| = 33{,}859$.

We extend each of the nine generated Tor network configurations with an identical set of web clients and servers to facilitate downloading pages from W_α, W_β, and $W_\emptyset$. We add 40 web client hosts that run `wget2` to fetch pages in W_α, 136 web client hosts that run `wget2` to fetch pages in $W_\alpha \cup W_\emptyset$, and 69 web server hosts that run `zimply` to serve our Wikipedia mirror from § 3.1.1 and balance load across the clients. The Tor clients used by `wget2` are configured to choose a new entry relay for each circuit (guards are disabled), and to isolate each webpage fetch to its own circuit. Every entry is configured to record cell traces on web clients' circuits.

4.1.3 Datasets. We use Shadow to simulate the Tor network configurations and then collect the cell traces that are recorded by the Tor relays during the simulation. In each Shadow simulation, the web clients fetch 200 instances of each webpage in W_α and one instance of each webpage in $W_\alpha \cup W_\emptyset$. For each of the nine network configurations we run two independent Shadow simulations and collate the resulting cell traces, yielding the following nine datasets:

- 200×2 cell traces $\forall w \in W_\alpha$ recorded with load ℓ and seed s
- 1×2 cell traces $\forall w \in W_\beta$ recorded with load ℓ and seed s
- 1×2 cell traces $\forall w \in W_\emptyset$ recorded with load ℓ and seed s

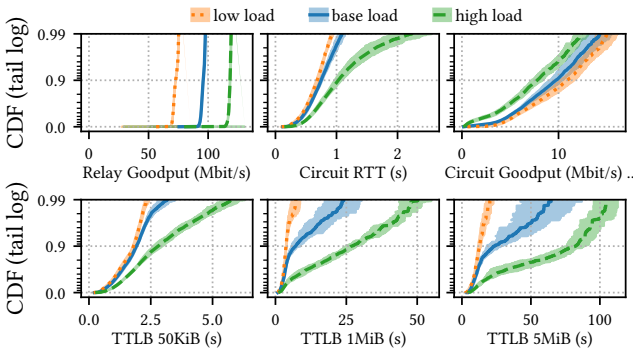

Figure 4: Performance in our Shadow-simulated Tor networks. The shaded regions show 95% CIs computed across six trials for each load profile. Higher load causes more congestion and decreases client performance.

for all load values $\ell \in \{\text{Base}, \text{Low}, \text{High}\}$ and seed values $s \in \{1, 2, 3\}$.[6] We use $\mathcal{D}(\ell, s)$ to denote the collection of pages gathered at load value ℓ and with network seed s.

4.1.4 Ethics. Our experiments are run in a completely safe and private environment in Shadow and carry no ethical risks to Tor or any of its users or stakeholders.

4.2 Network Performance Analysis

In this section we analyze the performance of the Shadow-simulated Tor networks configured as described in § 4.1.1 to demonstrate the differences in the network conditions during the collection of our datasets of cell traces. Our analysis will guide us to better understand the effects of congestion in our WF evaluation.

We run 18 Shadow simulations in total (see § 4.1.1) using the same machines and containers described in § 3.3.1. We analyze performance with respect to the load profiles defined in Fig. 4.1.1, since load is the primary contributing factor to client performance and relay congestion. That is, we treat each simulation with the same load profile as an independent trial run of that configuration. Thus, we collapse the 18 simulations into three experiments (base, low, and high load) each containing six simulation trials. We find that each simulation in the low, base, and high load experiments respectively consumes at most 707 GiB, 858 GiB, and 992 GiB of RAM and completes in fewer than 24 hours, 31 hours, and 42 hours.

Network performance is shown in Fig. 4. We observe that total relay goodput increases when configuring higher load levels as expected: the median relay goodput is 70.6 Gbit/s, 94.1 Gbit/s, and 117 Gbit/s at low, base, and high levels of load. We also observe that higher load levels decrease the client performance, especially in the tails of the distributions. For example, the time to download 1 MiB files increases from 3.8s to 7.2s to 27s in the median and from 4.8s to 16s to 42s in the third quartile, for low, base, and high levels of load, respectively. We observe that the other client performance metrics follow similar trends.

[6]Seed here refers to the seed used to generate the Tor network configuration with `tornettools`, not the seed used to run the Shadow simulation.

Key Takeaway. The client performance metrics indicate that relays become more congested in networks with higher load, which will help contextualize the results of our WF evaluations.

4.3 Website Fingerprinting Attacks

In this section we evaluate the effects that relay composition and relay load have on WF attacks. We will do this by evaluating *out-of-distribution* classification performance. In practice, it is difficult for the adversary to train a classifier in the exact network conditions that will be experienced during testing (due to natural load variations and relay churn in the network). We quantify out-of-distribution performance by testing the WF attacks under different network conditions than those under which they were trained.

4.3.1 Attacks. As mentioned in §3.3.2, we consider four WF attacks in this work: CUMUL [36], k-Fingerprinting [17], Deep Fingerprinting [42], and Tik-Tok [40].

CUMUL makes classifications using a support vector machine (SVM) with the radial basis function kernel. The SVM takes as input a cumulatively summed directional cell sequence $\langle c_i \rangle$ of a trace, where $c_i = \sum_{j=0}^{i} d_j$ and $d_j \in \{-1, 1\}$ indicates the direction of the jth cell. The cumulative sequence is post-processed with linear scaling to have exactly 100 elements.

k-Fingerprinting (k-FP) pairs a random forest classifier with a k-nearest neighbors classifier. If the classification task is multiclass, the random forest classifier is exclusively used. The random forest classifier takes as input a number of summary statistics of a trace, many of which are related to the timing of packets, such as the minimum, maximum, mean, and standard deviation of cell inter-arrival times. For binary classification, a two-step classification scheme is used. First, the random forest classifier is trained. The classification leaf index of each tree in the forest is stored for each training example, which the authors call a *fingerprint*. To classify new test examples, the trained random forest is first evaluated to obtain the example's fingerprint. Then, the k closest fingerprints in the training database are computed using Hamming distance. If all k agree that the page is sensitive, then the predicted page label is sensitive; otherwise, the page is predicted to be benign.

Deep Fingerprinting (DF) is implemented using a deep convolutional neural network which consists of a repeated stack of convolutional, batch normalization, activation, pooling, and dropout layers. Cell directions from a trace $\langle d_i \rangle_{i=1}^{5{,}000}$, $d_i \in \{-1, 1\}$ are provided directly as input to the network. (The input is zero-padded if the original cell sequence contains fewer than 5,000 cells.) DF has been shown to outperform k-FP and CUMUL in prior work [42].

Tik-Tok (TT) uses the same convolutional neural network as the DF attack, but instead of using only cell directions as input, it uses cell *directional times* computed by multiplying each cell's direction with its time $\langle t_i \cdot d_i \rangle_{i=1}^{5{,}000}$, $d_i \in \{-1, 1\}$. Hence, TT uses strictly more information than DF (time *and* direction), but note that timing information may be noisy or spuriously correlated with webpages.

These attacks cover two important dimensions of WF attack design space as shown pictorially in Fig. 5: classical or neural, and time-aware or time-oblivious. WF attacks using neural networks tend to outperform classical ML attacks and do not require manual feature engineering, but (1) are less interpretable, and (2) are more

	Time Oblivious	Time Aware
Classical	CUMUL [36]	k-Fingerprinting (k-FP) [17]
Neural	Deep Fingerprinting (DF) [42]	Tik-Tok (TT) [40]

Figure 5: Properties of the four website fingerprinting classifiers considered in this work. We consider both classical models and neural networks, as well as models that use time-based features and models that do not.

expensive to train and evaluate. Time-aware attacks use cell timing information as input to the classifier, whereas time-oblivious attacks do not. Using timing information can improve classifier performance, but also may make the classifier more brittle and sensitive to exogenous network conditions [40].

Hyperparameter Selection. Each of the attacks requires some model hyperparameters (i.e., model parameters not automatically learned during training) to be selected by the implementer. For Tik-Tok and k-Fingerprinting, we use the same hyperparameters as the original authors. For Deep Fingerprinting, we used the same hyperparameters settings as the original work except the number of training epochs. The original work used 30 training epochs, but this value did not lead to convergence on our datasets. Instead, we use 100 training epochs which is consistent with the Tik-Tok attack. For CUMUL, we use nested 3-fold validation (over only training data) to determine the γ and C parameters of the classifier. A grid search is used to select the best parameter pair for $2^{-15} \leq \gamma \leq 2^3$ and $2^{-5} \leq C \leq 2^{15}$ for each trained classifier.

4.3.2 Experimental Methodology. Attack Scenarios. In this section, we quantify the effectiveness of two types of WF attacks: (1) an open-world binary classification attack, and (2) a closed-world multiclass classification attack (refer to § 2.2 for details). In the open-world binary attack, we use TPR and FPR as measures of performance (see § 2.3; since the class labels are imbalanced, the benign pages outnumber the sensitive pages). In the closed-world multiclass attack, we use accuracy as the measure of performance because the classes are balanced. We randomly select 5 pages from W_α to serve as the sensitive set of pages in the open-world binary attack.

Datasets. We randomly partition each of the 9 datasets ($\mathcal{D}(\ell, s)$ for $\ell \in \{\text{Base, Low, High}\}, s \in \{1, 2, 3\}$) into a ~60% training set and ~40% testing set. We use stratified random sampling to keep the balance of page labels consistent in the training and testing sets.

Concretely, for the closed-world multiclass setting, the setup results in 180 training examples and 120 testing examples for each of the 98 pages in W_α. For the open-world binary setting, the setup results in 180 training examples for each of the 5 pages in the sensitive set, and 18,000 training examples of benign pages from W_β. The test set consists of 120 examples for each of the 5 pages in sensitive set, and 12,000 examples from the pages in W_β and $W_\emptyset$ (50% each). For convenience of notation, let $Train(\ell, s)$ denote the training set part and $Test(\ell, s)$ denote the testing set part of $\mathcal{D}(\ell, s)$.

Experiments. For both the open-world binary attack and the closed-world multiclass attack, we train each of the 4 classifiers on all 9 training sets $Train(\ell, s), \ell \in \{\text{Base, Low, High}\}, s \in \{1, 2, 3\}$. Then, we test all of the 4×9 classifiers on all 9 test sets $Test(\ell, s), \ell \in \{\text{Base, Low, High}\}, s \in \{1, 2, 3\}$.

Performance Metrics. For the binary classification attack, we record the TPR and FPR on the test set. For the multiclass attack, we record the accuracy obtained on the test set.

4.3.3 Results. All non-aggregated results from these experiments are included in the Appendix; see Table 7 for an overview.

Table 2a and Table 2b report the average classification results from these experiments. The "Baseline" columns of the tables show the case in which the training load and seed are the same as the testing load and seed; i.e., the attack is evaluated in-distribution. The "Variable Load" columns show averages computed over all of the experiments where the training load is different than the testing load, but the training seed is the same as the testing seed; in this case, average accuracy is defined as

$$\text{Avg}_{\ell,s} \left\{ \text{Acc}\left(C(\ell, s), Test(\ell', s)\right) \right\}_{\ell' \neq \ell} \tag{3}$$

where $C(\ell, s)$ is a classifier trained with load ℓ and seed s, and $\text{Acc}(C, Test)$ is the accuracy of applying classifier C to test set $Test$. The "Variable Seed" columns show averages computed analogously, but show the effects of varying the seed s (i.e., network composition) instead of varying the load ℓ. Note that we never simultaneously vary both the testing seed and the testing load so that we can separately examine these two effects.

Table 2a shows only a 1–3% difference in closed-world classification accuracy across the tested conditions, indicating that the WF classifiers are not greatly affected by out-of-distribution testing in this particular scenario. However, Table 2b shows that larger differences in the TPR and FPR are observed in the open-world binary experiments as the conditions are changed. For example, k-FP's average TPR degrades from 97% to 78% and average FPR increases by approximately one order of magnitude when testing with varying network loads. In general, varying the network load had a greater reduction on classifier performance than varying the network composition. Additionally, time-based classifiers were more severely affected: TT experienced a higher relative increase in FPR than DF in out-of-distribution testing, and k-FP had a higher FPR increase than did CUMUL.

Out-of-distribution performance was particularly poor when training on data from a low-load network and testing on data from a high-load network. In this case, for example, TT achieved a TPR of only 65% and a large FPR of 5% (for seed $s = 3$, see Table 14); this false positive rate is *200%* larger than measured at the baseline. (FPR is a particularly important metric when quantifying the scale at which a classifier may be applied, and there has been some focus on reducing FPRs [46].) There were also particular cases when out-of-distribution network composition (i.e., seed) had a large effect. For example, k-FP's recall degraded to 69% when trained on data from the $s = 1$ networks but tested on data from the $s = 3$ networks.

In Table 3a and Table 3b, we present the results from our experiments considering a different perspective: that the in-distribution performance of a classifier could be used as a predictor of out-of-distribution performance. This perspective models the scenario in

Table 2a and Table 2b: Average classifier performance (see Eqn. 3) for the (a) closed-world multiclass and (b) open-world binary experiments. We consider baseline conditions, varying the network load, and varying the seed (i.e., network composition).

(a) Closed-world multiclass experiments.

	Accuracy		
	Baseline	Variable Load	Variable Seed
CUMUL	.99	.96	.96
k-FP	.99	.98	.99
DF	.99	.98	.98
TT	.99	.98	.98

(b) Open-world binary experiments.

	Baseline		Variable Load		Variable Seed	
	TPR	FPR	TPR	FPR	TPR	FPR
CUMUL	.99	1.65×10^{-3}	.89	1.55×10^{-3}	.89	1.59×10^{-3}
k-FP	.97	4.44×10^{-4}	.78	2.37×10^{-3}	.86	5.00×10^{-4}
DF	.99	1.46×10^{-3}	.89	2.90×10^{-3}	.93	1.46×10^{-3}
TT	.98	1.06×10^{-3}	.89	6.57×10^{-3}	.93	1.23×10^{-3}

Table 3a and Table 3b: Average and maximum error of in-distribution performance as a predictor of out-of-distribution performance (see Eqn. 4) for the (a) closed-world multiclass and (b) open-world binary experiments.

(a) Closed-world multiclass experiments.

	Accuracy			
	Variable Load		Variable Seed	
	Avg %-Error	Max %-Error	Avg %-Error	Max %-Error
CUMUL	3	7	3	5
k-FP	2	4	1	2
DF	2	5	1	3
TT	2	5	1	3

(b) Open-world binary experiments.

	Variable Load				Variable Seed			
	Avg %-Error		Max %-Error		Avg %-Error		Max %-Error	
	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR
CUMUL	13	47	42	137	13	26	37	69
k-FP	35	75	106	100	14	47	41	300
DF	15	70	69	367	6	45	20	250
TT	15	82	53	250	6	78	18	700

which data is collected from a Tor network over a short time period and then used to evaluate WF attacks without considering that the network changes over the longer term. To illustrate, suppose we want to compute percent error of out-of-distribution load for a performance metric M (e.g., TPR). Let $M_{\ell,s}$ denote the metric computed on a test set with load ℓ and seed s. For two distinct measurements $M_{\ell,s}$ and $M_{\ell',s}$ with $\ell \neq \ell'$, percent error can be computed according to the standard definition $|M_{\ell,s} - M_{\ell',s}|/M_{\ell',s}$. *Average* error can be computed over all seeds and load values as

$$\text{Avg}_{\ell,s} \left\{ \frac{\left|M_{\ell,s} - M_{\ell',s}\right|}{M_{\ell',s}} \right\}_{\ell' \neq \ell} \tag{4}$$

Maximum error and error due to network composition (different network seeds) are computed similarly.

The results in Table 3a show that the error is relatively low in the closed-world multiclass experiments. However, the results in Table 3b suggest that in-distribution TPR and FPR are generally poor predictors of out-of-distribution TPR and FPR when considering the open-world binary experiments. For example, the average %-error is as high as 82% for TT when load is varied. Furthermore, there are individual instances where the estimation error is as high as 700%. With such high error rates, real-world classification performance metrics may become unreliable as the Tor network naturally evolves. Based on these results, we recommend that practitioners and researchers should directly measure classifier performance under a range of different conditions to help quantify this uncertainty.

Key Takeaway. Closed-world multiclass accuracy is not significantly affected by out-of-distribution network composition and load. However, classifier TPR and FPR is sensitive to both of these conditions. The sensitivity is pronounced for classifiers that rely on time-based features, such as k-Fingerprinting and Tik-Tok.

5 ROBUSTNESS

In this section, we investigate RQ3: *How can WF classifiers be made more robust to changing network composition and performance?* We seek to answer this question by augmenting WF classifiers using training examples from multiple unique network environments and evaluating the extent to which the augmentation improves classification accuracy. We describe the data we use for augmentation, our WF experiments, and our evaluation results.

5.1 Methodology

Our methodology for generating Tor network configurations, running simulations in Shadow, and collecting cell traces is identical to the methodology from § 4.1. We do not run any new Shadow simulations for this section; we use the same datasets of cell traces defined in § 4.1.3 to carry out our WF evaluation.

However, instead of training the classifiers on the training sets we previously defined, we create *mixture* training sets from the original ones. The intuition here is that the mixture training sets will contain a richer distribution and more intra-class example variance that the classifier can use to produce more generalizable decision boundaries. This idea is similar to dataset augmentation, which is a popular, classifier-agnostic method to improve generalization [20, 34].

We create 3 mixed training datasets in total: $\mathit{Train}(S = \{1, 2\})$, $\mathit{Train}(S = \{1, 3\})$, and $\mathit{Train}(S = \{2, 3\})$. $\mathit{Train}(S)$ is a uniform mixture of the training sets $\{\mathit{Train}(\ell, s)\}_{\ell \in \{\text{Base,Low,High}\}, s \in S}$; i.e., a trace chosen at random from $\mathit{Train}(\{1, 2\})$ is equally likely to have been sampled from any of the training sets $\{\mathit{Train}(\ell, s)\}$ for $\ell \in \{\text{Base, Low, High}\}, s \in \{1, 2\}$.

We keep the number of training examples in the mixed training datasets equal to the original datasets (180 examples × 98 pages for the closed-world and 180 examples × 5 pages + 18,000 examples

for the open-world). In practice, increasing the number of training examples can additionally improve classifier performance. However, we keep the number of training examples constant so we can experiment with the effects of training set distribution in a controlled manner.

We trained each of the 4 classifiers on the 3 mixed datasets. Classifiers are tested on the datasets with seeds not present in the training set. For example, classifiers trained on $Train(\{1,2\})$ are tested on $Test(\ell, 3)$ for $\ell \in \{\text{Base}, \text{Low}, \text{High}\}$. This training setup effectively assumes that the adversary can explore a realistic range of load values during training, but will be unlikely to determine the exact network composition that will be encountered during testing. We also tested each of the trained classifiers on $D(\text{wget2, tor})$ (the traces we collected directly on the live Tor network referenced in § 3.3.2) to see if the augmented classifiers experienced boosted real-world performance.

5.2 Results

In Table 4, we report average classifier performance, similar to the results presented in Table 2a and Table 2b. The "New" columns report the average performance of each classifier when applied to all loads and for the test seed not in the training set. For example, average accuracy is defined as

$$\text{Avg}_{S \in \mathcal{S}} \{\text{Acc}(C(S), Test(\ell, s))\}_{\ell \in \{\text{Base,Low,High}\}, s \in \{1,2,3\} \setminus S} \quad (5)$$

where $\mathcal{S} = \{\{1,2\},\{2,3\},\{1,3\}\}$, $C(S)$ is the classifier trained on seeds S, and $\text{Acc}(C, Test)$ is the accuracy of the classifier C on test set $Test$. The "Old" column shows the analogous "Variable Load" performance metrics reported in Table 2a and Table 2b; we want to examine if the classifiers trained across various load values outperform the load-sensitive classifiers. Finally, the "%Δ" column displays the percent difference between the old (sensitive) and new (robust) classification metrics.

We observe only slight improvement in closed-world classification accuracy with robust training because accuracy was already near-maximal prior to our augmentation. We observe larger improvements open-world TPR: CUMUL, Deep Fingerprinting, and Tik-Tok's recall improves between 6-8%, and k-Fingerprinting is boosted by 19%. The largest improvements are experienced in FPR: k-Fingerprinting, Deep Fingerprinting, and Tik-Tok all had reductions in FPR of at least 50%. The time-aware classifiers both had larger reductions in FPR than their time-oblivious counterparts, suggesting that time-aware classifiers may benefit more from the robust training process we propose.

Table 5a and Table 5b shows classifier performance when applied to the three distinct test load levels individually without averaging across them (but still averaged across the test seeds). Classification accuracy, TPR, and FPR is best in the low- and base-load conditions, where collected cell traces are likely to be less noisy. Classification performance in the high-load condition is degraded slightly, but performance is improved with respect to the sensitive classifiers. For example, k-Fingerprinting achieved a TPR of at most 56% when trained with low load and tested with high load (see Table 14), whereas the robust classifier achieves ~83% TPR (Table 5b). When trained on low load and tested on high loads, Tik-Tok had a false-positive rate as high as 5%. With the robust training procedure, FPR was an order of magnitude lower (2×10^{-3}).

Table 4: Overall average robust classifier performance compared to the classifier performance sensitive to network load reported in Table 2a and Table 2b. The left 1/3 of the table shows improvements in accuracy for the closed-world multiclass experiments, and the right 2/3 shows improvements in TPR and FPR for the open-world binary experiments.

	Accuracy			TPR			FPR		
	Old	New	%Δ	Old	New	%Δ	Old	New	%Δ
CUMUL	.96	.99	+3	.89	.96	+8	1.55×10^{-3}	1.49×10^{-3}	−4
k-FP	.98	.99	+1	.78	.93	+19	2.37×10^{-3}	5.83×10^{-4}	−75
DF	.98	.99	+1	.89	.95	+7	2.90×10^{-3}	1.49×10^{-3}	−49
TT	.98	.99	+1	.89	.94	+6	6.57×10^{-3}	1.13×10^{-3}	−83

Finally, Table 6 show the performance of the classifiers when tested against the $D(\text{wget2, tor})$ dataset in the closed-world multiclass setting. Recall that data in $D(\text{wget2, tor})$ was collected using `wget2` directly in the live Tor network. The "Original" row shows the accuracy the classifiers when they were trained using $D(\text{wget2, shadow})$, and the "Robust" row shows the average performance of the three robustly trained variants of each classifier.

For the classical models, CUMUL and k-Fingerprinting, we find that classification accuracy was actually degraded instead of improved. It is possible that in this case robust training reduced in-distribution performance by preventing the classifiers from fitting the data too precisely. However, large performance improvements are achieved for both Deep Fingerprinting and Tik-Tok, the neural network models. The largest improvement is experienced by Tik-Tok, which receives a *36% improvement* in attack accuracy.

Key Takeaway. Overall, our results suggest that open-world binary classifiers can be made more robust to variable network congestion levels and compositions while the simpler, closed-world multiclass attacks may already be robust to these effects. We propose a technique to train classifiers across a range of network congestion levels and compositions, but other techniques may also be explored to improve robustness and validated via simulation.

6 RELATED WORK

WF Attacks. WF has been studied for well over a decade [10, 18, 19, 31, 44]. Early WF works focused on demonstrating the feasibility of attacks and on improving assumptions [12, 37, 47]. Following initial attacks, classical ML techniques were designed to use simple but robust features developed by domain experts [17, 36]. Later, WF attacks were improved with the application of deep learning [6, 9, 35, 41, 42], leading to better accuracy and robustness against the latest countermeasures. We use several of these attacks in our evaluations (see Fig. 5).

More recently, techniques were developed to improve classifier portability and to reduce the number of examples needed for training. In Triplet Fingerprinting, a triplet of neural networks is trained using older WF datasets and then N-shot learning is applied to reduce the number of real training examples that need to be gathered [43]. GANDaLF extends this idea, using a GAN to generate the initial data to train the neural network and then supplementing the GAN data with additional real training examples [34]. In contrast

Table 5a and Table 5b: Robust classifier performance when tested on circuits collected from differently loaded networks. The reported values are average over the network seeds. Table 5a shows the results from the closed-world multiclass experiments and Table 5b shows the results from the open-world binary experiments.

(a) Closed-world multiclass experiments.

	Accuracy		
	Test Load: Low	Test Load: Medium	Test Load: High
CUMUL	1.00	1.00	.98
k-FP	.99	1.00	.98
DF	.99	.99	.98
TT	.99	.99	.98

(b) Open-world binary experiments.

	Test Load: Low		Test Load: Base		Test Load: High	
	TPR	FPR	TPR	FPR	TPR	FPR
CUMUL	.99	1.56×10^{-3}	.99	1.50×10^{-3}	.91	2.89×10^{-3}
k-FP	.98	1.67×10^{-4}	.99	5.56×10^{-4}	.83	1.03×10^{-3}
DF	.99	1.03×10^{-3}	.98	1.08×10^{-3}	.87	2.36×10^{-3}
TT	.99	5.56×10^{-4}	.98	6.94×10^{-4}	.86	2.14×10^{-3}

Table 6: Robust classifier accuracy when applied to the D(wget2, tor) dataset used in § 3.3.2. The original accuracy values are taken from Table 1.

	Accuracy			
	CUMUL	*k*-FP	DF	TT
Original	0.86	0.83	0.76	0.61
Robust	0.50	0.70	0.82	0.83
%Δ	-42	-16	+8	+36

to these works, we show how WF classifiers can be trained *exclusively* in simulation, i.e., with no real-world training examples, and still produce high accuracies when transferred to the real world. Synthesizing both simulated and real-world data into the training phase is an interesting direction for further study.

WF Defenses. Numerous WF defenses have been proposed, each showing various levels of protection against the then-recent attacks. Many defenses are later shown to be much less effective than originally proposed. A more detailed comparison of WF defenses is out of scope for this paper but recent work by Mathews et al. provides a systematization of this knowledge [33]. Of relevance to this paper is recent work by Witwer et al. in which Shadow simulations were used to demonstrate that padding defenses that claim to be "zero-delay" actually do degrade network performance due to increased network queuing [49].

Real-World WF Considerations. Early WF work has been criticized for the use of unrealistic assumptions and methodologies [28, 38], and several later works have improved on early evaluation methods. Wang and Goldberg demonstrated how to overcome data freshness limitations [48], Panchenko et al. studied WF with a larger and more realistic set of 150k internal webpage URLs, and Rimmer et al. considered an even larger set of 400k websites [41]. More recently, Wang has proposed that WF attacks be optimized for a precision metric that accounts for base rates to avoid over-estimating attack performance [46], and Cherubin et al. proposed an online WF method that incorporates genuine cell traces from a Tor exit relay to safely evaluate real-world WF performance [13]. Pulls and Dahlberg demonstrate how the existence of a website oracle can reduce uncertainty about WF predictions. Our work considers how Tor network simulation can be used to improve WF explainability and make WF classifiers more robust in the real world.

7 CONCLUSION

In this paper, we demonstrate that network simulation is a viable strategy for producing datasets that lead to more explainable WF evaluations. We show how to simulate webpage requests in Shadow using `wget2` and quantify `wget2`'s fidelity in producing traffic patterns that are similar to tor-browser-selenium (TBS) both inside and outside of Shadow. We find that `wget2` produces highly similar flows to TBS and that a classifier trained entirely in Shadow can classify traffic collected on the live Tor network with greater than 85% accuracy. We investigate the sensitivity of WF classifiers to Tor network composition and Tor relay congestion and find that false-positive rates increase by as much as 700% when training and testing on data collected from networks with distinct levels of congestion. We also find that we can augment WF classifiers by training them on data collected *exclusively in network simulation* across a range of network congestion levels, and that such augmentation can decrease false positives rates by as much as 83%. We demonstrate that these robust classifiers can be transferred to the real world and some can achieve greater than 80% accuracy.

Limitations and Future Work. We propose the use of network simulation to increase control over WF evaluations, and we believe this is a useful methodology for studying causal relationships between network variations and WF performance. However, there are multiple opportunities for expanding our work and taking it in new directions. First, our study considers the use of `wget2` as a surrogate for Tor Browser configured to the "Safest" security level; this is done primarily to work around the lack of support for running web browsers in Shadow. Future work could further strengthen our methodology by developing Shadow support for Tor Browser (or tor-browser-selenium) and running it at the default security level (or multiple levels) instead. Second, our study considers many webpages but all from a single destination website: Wikipedia. While restricting our study to Wikipedia enabled us to quickly mirror over 20 million pages, it does not accurately represent the full website diversity that would be observed on the real Tor network, nor does it represent the traffic complexity that the adversary would need to manage when running WF attacks. Future work that uses our simulation methodology should strongly consider expanding the set of websites mirrored in Shadow to improve website diversity and traffic complexity. Finally, we believe there is also significant value in using genuine data from circuits that occur naturally in Tor [13] to more precisely estimate of the threat of WF. We leave it to future work to consider how to safely collect genuine WF data.

AVAILABILITY

The research artifacts produced as part of this paper—including the Tor cell trace datasets, ML classifier implementations and scripts, and Shadow simulation files—have been made publicly available at https://explainwf-popets2023.github.io.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their feedback and helpful suggestions to improve the paper. This work has been partially supported by the Office of Naval Research (ONR) and the National Science Foundation (NSF) under award CNS-1925497.

REFERENCES

[1] 2023. *The Tor Metrics Portal.* https://metrics.torproject.org
[2] 2023. *The Tor Metrics Portal: Network churn rate by relay flag.* https://metrics.torproject.org/networkchurn.html
[3] 2023. *The Tor Metrics Portal: Relays by Tor version.* https://metrics.torproject.org/versions.html
[4] 2023. *The Tor Metrics Portal: Top-10 countries by possible censorship events.* https://metrics.torproject.org/userstats-censorship-events.html
[5] 2023. *The Tor Project.* https://www.torproject.org
[6] Kota Abe and Shigeki Goto. 2016. Fingerprinting attack on Tor anonymity using deep learning. *Asia-Pacific Advanced Network* 42 (2016).
[7] Marco Valerio Barbera, Vasileios P. Kemerlis, Vasilis Pappas, and Angelos D. Keromytis. 2013. CellFlood: Attacking Tor Onion Routers on the Cheap. In *The 18th European Symposium on Research in Computer Security*, Vol. 8134. https://doi.org/10.1007/978-3-642-40203-6_37
[8] Richard Bellman. 1966. Dynamic Programming. *Science* 153, 3731 (1966). https://doi.org/10.1126/science.153.3731.34
[9] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. 2019. Var-CNN: A Data-Efficient Website Fingerprinting Attack Based on Deep Learning. *Proceedings on Privacy Enhancing Technologies* 2019, 4 (Oct. 2019). https://doi.org/10.2478/popets-2019-0070
[10] George Dean Bissias, Marc Liberatore, David D. Jensen, and Brian Neil Levine. 2005. Privacy Vulnerabilities in Encrypted HTTP Streams. In *The 5th Workshop on Privacy Enhancing Technologies*, Vol. 3856. https://doi.org/10.1007/11767831_1
[11] Nadia Burkart and Marco F Huber. 2021. A survey on the explainability of supervised machine learning. *Journal of Artificial Intelligence Research* 70 (2021).
[12] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. 2012. Touching from a distance: website fingerprinting attacks and defenses. In *The 19th Conference on Computer and Communications Security.* https://doi.org/10.1145/2382196.2382260
[13] Giovanni Cherubin, Rob Jansen, and Carmela Troncoso. 2022. Online Website Fingerprinting: Evaluating Website Fingerprinting Attacks on Tor in the Real World. In *The 31st USENIX Security Symposium.* https://www.usenix.org/conference/usenixsecurity22/presentation/cherubin
[14] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. 2004. Tor: The Second-Generation Onion Router. In *The 13th USENIX Security Symposium.*
[15] Martin A Fischler and Robert C Bolles. 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Commun. ACM* 24, 6 (1981).
[16] John Geddes, Rob Jansen, and Nicholas Hopper. 2014. IMUX: Managing tor connections from two to infinity, and beyond. In *The 13th Workshop on Privacy in the Electronic Society.*
[17] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *The 25th USENIX Security Symposium.*
[18] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. 2009. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier. In *The Workshop on Cloud Computing Security.*
[19] Andrew Hintz. 2002. Fingerprinting Websites Using Traffic Analysis. In *The 2nd Workshop on Privacy Enhancing Technologies*, Vol. 2482. https://doi.org/10.1007/3-540-36467-6_13
[20] Aaron Courville Ian Goodfellow, Yoshua Bengio. 2017. *Deep Learning.* MIT Press.
[21] Rob Jansen and Nicholas Hopper. 2012. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *The Network and Distributed System Security Symposium.*
[22] Rob Jansen and Aaron Johnson. 2021. On the Accuracy of Tor Bandwidth Estimation. *The 22nd Passive and Active Measurement Conference.*
[23] Rob Jansen, Marc Juárez, Rafa Galvez, Tariq Elahi, and Claudia Díaz. 2018. Inside Job: Applying Traffic Analysis to Measure Tor from Within. In *The Network and Distributed System Security Symposium.*
[24] Rob Jansen, Jim Newsome, and Ryan Wails. 2022. Co-opting Linux Processes for High-Performance Network Simulation. In *The USENIX Annual Technical Conference.* https://www.usenix.org/conference/atc22/presentation/jansen
[25] Rob Jansen, Justin Tracey, and Ian Goldberg. 2021. Once is Never Enough: Foundations for Sound Statistical Inference in Tor Network Experimentation. In *The 30th USENIX Security Symposium.* https://www.usenix.org/conference/usenixsecurity21/presentation/jansen
[26] Rob Jansen, Florian Tschorsch, Aaron Johnson, and Björn Scheuermann. 2014. The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network. In *The Network and Distributed System Security Symposium.*
[27] Rob Jansen, Tavish Vaidya, and Micah Sherr. 2019. Point Break: A Study of Bandwidth Denial-of-Service Attacks against Tor. In *The 28th USENIX Security Symposium.* https://www.usenix.org/conference/usenixsecurity19/presentation/jansen
[28] Marc Juárez, Sadia Afroz, Gunes Acar, Claudia Díaz, and Rachel Greenstadt. 2014. A Critical Evaluation of Website Fingerprinting Attacks. In *The 21st Conference on Computer and Communications Security.* https://doi.org/10.1145/2660267.2660368
[29] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. 2020. Lessons Learned from the Chameleon Testbed. In *The USENIX Annual Technical Conference.*
[30] Gregory M Kurtzer, Vanessa Sochat, and Michael W Bauer. 2017. Singularity: Scientific containers for mobility of compute. *PloS one* 12, 5 (2017).
[31] Marc Liberatore and Brian Neil Levine. 2006. Inferring the source of encrypted HTTP connections. In *The 13th Conference on Computer and Communications Security.* https://doi.org/10.1145/1180405.1180437
[32] Akshaya Mani, T Wilson-Brown, Rob Jansen, Aaron Johnson, and Micah Sherr. 2018. Understanding Tor Usage with Privacy-Preserving Measurement. In *The Internet Measurement Conference.*
[33] 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 *The Symposium on Security and Privacy.* https://doi.org/10.1109/SP46215.2023.00020
[34] Se Eun Oh, Nate Mathews, Mohammad Saidur Rahman, Matthew Wright, and Nicholas Hopper. 2021. GANDaLF: GAN for Data-Limited Fingerprinting. *Proceedings on Privacy Enhancing Technologies* 2021, 2 (April 2021). https://doi.org/10.2478/popets-2021-0029
[35] Se Eun Oh, Saikrishna Sunkam, and Nicholas Hopper. 2019. p1-FP: Extraction, Classification, and Prediction of Website Fingerprints with Deep Learning. *Proceedings on Privacy Enhancing Technologies* 2019, 3 (July 2019). https://doi.org/10.2478/popets-2019-0043
[36] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. 2016. Website Fingerprinting at Internet Scale. In *The Network and Distributed System Security Symposium.*
[37] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. 2011. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *The 10th Workshop on Privacy in the Electronic Society.*
[38] Mike Perry. 2013. *A Critique of Website Traffic Fingerprinting Attacks.* https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks (accessed: December 15, 2013).
[39] Tobias Pulls and Rasmus Dahlberg. 2020. Website Fingerprinting with Website Oracles. *Proceedings on Privacy Enhancing Technologies* 2020, 1 (Jan. 2020). https://doi.org/10.2478/popets-2020-0013
[40] Mohammad Saidur Rahman, Payap Sirinam, Nate Mathews, Kantha Girish Gangadhara, and Matthew Wright. 2020. Tik-Tok: The Utility of Packet Timing in Website Fingerprinting Attacks. *Proceedings on Privacy Enhancing Technologies* 2020, 3 (July 2020). https://doi.org/10.2478/popets-2020-0043
[41] Vera Rimmer, Davy Preuveneers, Marc Juárez, Tom van Goethem, and Wouter Joosen. 2018. Automated Website Fingerprinting through Deep Learning. In *The Network and Distributed System Security Symposium.*
[42] Payap Sirinam, Mohsen Imani, Marc Juárez, and Matthew Wright. 2018. Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning. In *The 25th Conference on Computer and Communications Security.* https://doi.org/10.1145/3243734.3243768
[43] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. 2019. Triplet Fingerprinting: More Practical and Portable Website Fingerprinting with N-shot Learning. In *The 26th Conference on Computer and Communications Security.* https://doi.org/10.1145/3319535.3354217
[44] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata N. Padmanabhan, and Lili Qiu. 2002. Statistical Identification of Encrypted Web Browsing Traffic. In *The Symposium on Security and Privacy.* https://doi.org/10.1109/SECPRI.2002.1004359
[45] Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. 1997. Anonymous Connections and Onion Routing. In *The Symposium on Security and Privacy.* https://doi.org/10.1109/SECPRI.1997.601314
[46] Tao Wang. 2020. High Precision Open-World Website Fingerprinting. In *The Symposium on Security and Privacy.* https://doi.org/10.1109/SP40000.2020.00015

[47] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective Attacks and Provable Defenses for Website Fingerprinting. In *The 23rd USENIX Security Symposium*.
[48] Tao Wang and Ian Goldberg. 2016. On Realistically Attacking Tor with Website Fingerprinting. *Proceedings on Privacy Enhancing Technologies* 2016, 4 (Oct. 2016). https://doi.org/10.1515/popets-2016-0027
[49] Ethan Witwer, James K. Holland, and Nicholas Hopper. 2022. Padding-Only Defenses Add Delay in Tor. In *The 21st Workshop on Privacy in the Electronic Society*. https://doi.org/10.1145/3559613.3563207

APPENDIX

A COMPLETE EXPERIMENTAL RESULTS

This appendix provides extended results for the experiments presented throughout this paper. Table 7 provides a hyperlinked listing and summary of each of the tables in which we provide complete ML experimental results.

Table 7: Hyperlinked listing of experimental results appearing in this appendix section.

Section	Attack Scenario	Train Load	Train Seed	Table
§4.3.3	Multiclass Closed	1.5	Variable	Table 8
§4.3.3	Multiclass Closed	2.0	Variable	Table 9
§4.3.3	Multiclass Closed	2.5	Variable	Table 10
§4.3.3	Multiclass Closed	Variable	1	Table 11
§4.3.3	Multiclass Closed	Variable	2	Table 12
§4.3.3	Multiclass Closed	Variable	3	Table 13
§4.3.3	Binary Open	1.5	Variable	Table 14
§4.3.3	Binary Open	2.0	Variable	Table 15
§4.3.3	Binary Open	2.5	Variable	Table 16
§4.3.3	Binary Open	Variable	1	Table 17
§4.3.3	Binary Open	Variable	2	Table 18
§4.3.3	Binary Open	Variable	3	Table 19

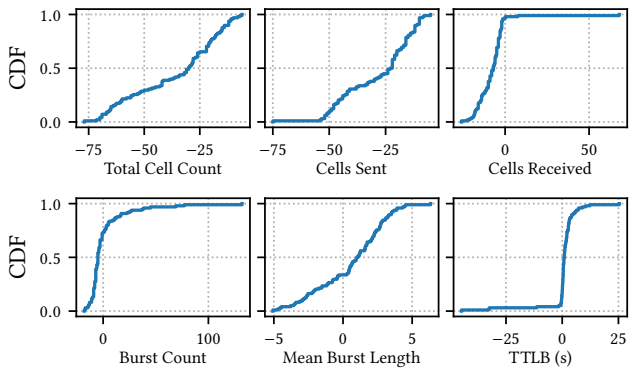

Figure 6: The cumulative distribution of the per-page difference in median feature values, as defined in Eqn. 2, for the six feature functions shown in the subplots. Values closer to $x = 0$ indicate more consistency across tools. The largest difference is due to `wget2` sending more cells than TBS. The symmetric long tails in TTLB indicates that circuit performance is variable, independent of the tool.

Figure 7: The mean and 99% confidence intervals (CIs) computed across multiple fetches of the 98 test pages, as described in § 3.2, for the six feature functions shown in the subplots. The black top part of each bar is the 99% CI; overlapping CIs indicate that fetching the same page with the same tool multiple times may produce the same result as fetching the page with the other tool. The numbers of overlapping CIs in the subplots, starting from the top, are 25/98, 0/98, 95/98, 19/98, 26/98, and 68/98.

Table 8: Multi-class classification closed world performance results when varying test network load with classifiers that were trained on a 1.5-load network.

		Baseline (Test Load = 1.5)	Test Load = 2.0		Test Load = 2.5	
		Acc	Acc	% Err	Acc	% Err
$s = 1$	CUMUL	1.00	0.97	2.44	0.96	4.04
	k-FP	0.99	0.99	0.11	0.97	2.39
	DF	1.00	0.99	0.31	0.97	2.76
	TT	1.00	0.99	0.45	0.96	3.60
$s = 2$	CUMUL	1.00	0.97	3.34	0.93	6.95
	k-FP	1.00	0.98	2.19	0.97	3.53
	DF	1.00	0.97	3.22	0.95	4.83
	TT	1.00	0.97	3.04	0.95	5.11
$s = 3$	CUMUL	1.00	1.00	0.31	0.94	6.13
	k-FP	1.00	1.00	0.40	0.96	4.04
	DF	1.00	1.00	0.25	0.96	4.54
	TT	1.00	1.00	0.30	0.96	4.50

Table 9: Multi-class classification closed world performance results when varying test network load with classifiers that were trained on a 2.0-load network.

		Baseline (Test Load = 2.0)	Test Load = 1.5		Test Load = 2.5	
		Acc	Acc	% Err	Acc	% Err
$s = 1$	CUMUL	1.00	0.98	2.18	0.96	3.49
	k-FP	0.99	1.00	0.14	0.98	1.90
	DF	0.99	0.99	0.02	0.97	2.03
	TT	0.99	1.00	0.02	0.98	1.76
$s = 2$	CUMUL	1.00	0.95	5.04	0.96	3.48
	k-FP	1.00	0.98	1.85	0.99	1.29
	DF	1.00	0.98	1.87	0.99	1.13
	TT	1.00	0.98	1.85	0.98	1.35
$s = 3$	CUMUL	1.00	0.99	0.72	0.94	5.85
	k-FP	1.00	1.00	0.14	0.96	3.83
	DF	1.00	1.00	0.08	0.96	4.07
	TT	1.00	1.00	0.04	0.96	3.99

Table 10: Multi-class classification closed world performance results when varying test network load with classifiers that were trained on a 2.5-load network.

		Baseline (Test Load = 2.5)	Test Load = 1.5		Test Load = 2.0	
		Acc	Acc	% Err	Acc	% Err
$s = 1$	CUMUL	0.98	0.98	0.05	0.97	0.94
	k-FP	0.98	0.99	1.15	0.99	1.15
	DF	0.98	0.99	0.92	0.99	0.76
	TT	0.98	0.99	1.06	0.99	0.73
$s = 2$	CUMUL	0.99	0.94	5.10	0.97	2.19
	k-FP	0.99	0.98	1.24	1.00	0.68
	DF	0.99	0.98	1.73	1.00	0.36
	TT	0.99	0.98	1.71	1.00	0.45
$s = 3$	CUMUL	0.99	0.95	4.10	0.95	4.36
	k-FP	0.99	0.98	1.02	0.98	1.13
	DF	0.99	0.98	1.15	0.98	1.31
	TT	0.99	0.98	0.81	0.98	1.03

Table 11: Multi-class classification closed world performance results when varying test network seed with classifiers that were trained with network seed = 1.

		Baseline (Test Seed = 1)	Test Seed = 2		Test Seed = 3	
		Acc	Acc	% Err	Acc	% Err
Load = 1.5	CUMUL	1.00	0.97	2.97	0.98	2.38
	k-FP	0.99	0.99	0.05	1.00	0.07
	DF	1.00	1.00	0.08	0.99	0.15
	TT	1.00	1.00	0.0	1.00	0.09
Load = 2.0	CUMUL	1.00	0.95	5.32	0.97	2.61
	k-FP	0.99	0.99	0.02	0.99	7.9×10^{-3}
	DF	0.99	0.98	0.48	0.99	0.22
	TT	0.99	0.99	0.24	0.99	0.12
Load = 2.5	CUMUL	0.98	0.95	3.51	0.96	1.62
	k-FP	0.98	0.98	0.17	0.98	0.07
	DF	0.98	0.99	0.36	0.98	0.08
	TT	0.98	0.98	0.24	0.98	0.11

Table 12: Multi-class classification closed world performance results when varying test network seed with classifiers that were trained with network seed = 2.

		Baseline (Test Seed = 2)	Test Seed = 1		Test Seed = 3	
		Acc	Acc	% Err	Acc	% Err
Load = 1.5	CUMUL	1.00	0.97	3.10	0.96	3.77
	k-FP	1.00	0.99	0.91	0.98	2.04
	DF	1.00	0.98	1.87	0.98	2.00
	TT	1.00	0.98	1.79	0.97	2.77
Load = 2.0	CUMUL	1.00	0.96	3.53	1.00	0.15
	k-FP	1.00	0.99	1.05	1.00	0.11
	DF	1.00	0.99	1.03	1.00	0.11
	TT	1.00	0.98	1.33	1.00	0.0
Load = 2.5	CUMUL	0.99	0.96	2.96	0.96	2.67
	k-FP	0.99	0.97	1.72	0.97	2.39
	DF	0.99	0.98	1.86	0.97	2.72
	TT	0.99	0.98	1.74	0.97	2.57

Table 13: Multi-class classification closed world performance results when varying test network seed with classifiers that were trained with network seed = 3.

		Baseline (Test Seed = 3)	Test Seed = 1		Test Seed = 2	
		Acc	Acc	% Err	Acc	% Err
Load = 1.5	CUMUL	1.00	0.96	4.20	0.95	5.43
	k-FP	1.00	0.99	1.04	0.98	2.08
	DF	1.00	0.99	1.05	0.98	1.98
	TT	1.00	0.99	1.04	0.98	2.04
Load = 2.0	CUMUL	1.00	0.96	3.56	0.97	2.38
	k-FP	1.00	0.98	1.64	1.00	0.02
	DF	1.00	0.98	1.90	1.00	0.27
	TT	1.00	0.97	2.39	1.00	0.12
Load = 2.5	CUMUL	0.99	0.96	2.85	0.96	2.86
	k-FP	0.99	0.97	2.05	0.97	1.76
	DF	0.99	0.98	1.07	0.98	1.64
	TT	0.99	0.98	0.95	0.98	1.38

Table 14: Binary classification open world performance results when varying test network load with classifiers that were trained on a 1.5-load network.

		Baseline (Test Load = 1.5)		Test Load = 2.0				Test Load = 2.5			
		TPR	FPR	TPR	% Err	FPR	% Err	TPR	% Err	FPR	% Err
$s=1$	CUMUL	0.99	7.5×10^{-4}	0.96	3.17	5.8×10^{-4}	28.58	0.71	39.91	2.7×10^{-3}	71.87
	k-Fingerprinting	0.97	2.5×10^{-4}	0.92	5.55	2.3×10^{-3}	89.28	0.47	106.14	6.6×10^{-3}	96.20
	Deep Fingerprinting	0.99	4.2×10^{-4}	0.96	3.22	9.2×10^{-4}	54.54	0.62	57.87	5.1×10^{-3}	91.80
	Tik-Tok	0.99	2.5×10^{-4}	0.97	2.12	3.7×10^{-3}	93.33	0.67	48.75	0.03	99.05
$s=2$	CUMUL	0.99	1.4×10^{-3}	0.98	1.53	1.4×10^{-3}	0.0	0.89	11.19	1.4×10^{-3}	0.0
	k-Fingerprinting	0.98	3.3×10^{-4}	0.81	20.74	3.6×10^{-3}	90.70	0.56	76.58	0.02	97.83
	Deep Fingerprinting	0.99	7.5×10^{-4}	0.82	20.49	6.7×10^{-4}	12.50	0.79	26.11	4.3×10^{-3}	82.35
	Tik-Tok	0.99	6.7×10^{-4}	0.83	18.16	1.3×10^{-3}	50.00	0.76	30.40	0.01	94.87
$s=3$	CUMUL	0.99	3.3×10^{-4}	0.94	5.66	7.5×10^{-4}	55.56	0.77	29.50	1.5×10^{-3}	77.78
	k-Fingerprinting	0.98	8.3×10^{-5}	0.95	3.17	7.5×10^{-4}	88.89	0.50	94.04	2.0×10^{-3}	95.83
	Deep Fingerprinting	0.99	1.7×10^{-4}	0.92	7.64	1.3×10^{-3}	86.67	0.58	69.14	0.01	98.78
	Tik-Tok	0.99	2.5×10^{-4}	0.98	1.36	3.5×10^{-3}	92.86	0.65	53.47	0.05	99.46

Table 15: Binary classification open world performance results when varying test network load with classifiers that were trained on a 2.0-load network.

		Baseline (Test Load = 2.0)		Test Load = 1.5				Test Load = 2.5			
		TPR	FPR	TPR	% Err	FPR	% Err	TPR	% Err	FPR	% Err
$s=1$	CUMUL	0.99	1.4×10^{-3}	0.96	3.28	1.2×10^{-3}	21.42	0.76	31.25	2.9×10^{-3}	51.43
	k-Fingerprinting	0.96	3.3×10^{-4}	0.93	4.07	0.0	inf	0.49	96.87	1.2×10^{-3}	71.43
	Deep Fingerprinting	0.99	1.7×10^{-3}	1.00	0.35	1.8×10^{-3}	8.3×10^{-3}	0.82	21.74	5.9×10^{-3}	70.42
	Tik-Tok	0.98	1.7×10^{-4}	0.97	0.30	1.7×10^{-4}	8.3×10^{-3}	0.66	49.22	2.2×10^{-3}	92.31
$s=2$	CUMUL	0.99	9.2×10^{-4}	0.96	3.12	1.2×10^{-3}	21.43	0.92	8.18	1.3×10^{-3}	31.25
	k-Fingerprinting	0.99	1.7×10^{-4}	0.95	4.23	2.5×10^{-4}	33.33	0.75	31.56	9.2×10^{-4}	81.82
	Deep Fingerprinting	0.98	1.8×10^{-3}	0.98	0.17	1.6×10^{-3}	10.53	0.92	6.88	3.1×10^{-3}	43.24
	Tik-Tok	0.98	3.3×10^{-4}	0.97	1.03	1.7×10^{-4}	100.00	0.90	9.44	1.6×10^{-3}	78.95
$s=3$	CUMUL	0.98	9.2×10^{-4}	0.69	41.83	5.0×10^{-4}	83.33	0.88	12.17	9.2×10^{-4}	0.0
	k-Fingerprinting	0.96	3.3×10^{-4}	0.77	25.32	0.0	inf	0.57	68.80	1.4×10^{-3}	76.47
	Deep Fingerprinting	0.99	1.2×10^{-3}	0.88	11.89	2.5×10^{-4}	366.67	0.78	27.53	4.0×10^{-3}	70.83
	Tik-Tok	0.99	7.5×10^{-4}	0.92	7.62	0.0	inf	0.77	28.63	4.1×10^{-3}	81.63

Table 16: Binary classification open world performance results when varying test network load with classifiers that were trained on a 2.5-load network.

		Baseline (Test Load = 2.5)		Test Load = 1.5				Test Load = 2.0			
		TPR	FPR	TPR	% Err	FPR	% Err	TPR	% Err	FPR	% Err
$s=1$	CUMUL	0.98	2.4×10^{-3}	0.94	3.87	1.3×10^{-3}	81.23	0.93	5.12	2.4×10^{-3}	0.0
	k-Fingerprinting	0.96	5.0×10^{-4}	0.88	9.19	2.5×10^{-4}	99.98	0.83	16.37	8.3×10^{-4}	40.00
	Deep Fingerprinting	0.98	3.2×10^{-3}	1.00	1.69	1.8×10^{-3}	77.26	0.99	0.85	2.2×10^{-3}	44.44
	Tik-Tok	0.98	1.7×10^{-3}	1.00	2.20	5.0×10^{-4}	249.97	0.99	1.19	1.2×10^{-3}	40.00
$s=2$	CUMUL	0.96	2.9×10^{-3}	0.98	1.19	2.6×10^{-3}	12.90	0.99	2.69	1.8×10^{-3}	59.09
	k-Fingerprinting	0.96	1.1×10^{-3}	0.96	0.0	6.7×10^{-4}	62.50	0.99	2.70	5.8×10^{-4}	85.71
	Deep Fingerprinting	0.99	1.3×10^{-3}	1.00	1.00	1.4×10^{-3}	11.76	1.00	1.00	1.3×10^{-3}	0.0
	Tik-Tok	0.98	2.8×10^{-3}	1.00	1.17	1.8×10^{-3}	54.55	1.00	1.50	1.3×10^{-3}	126.67
$s=3$	CUMUL	0.98	3.7×10^{-3}	0.82	18.79	1.6×10^{-3}	136.84	0.95	3.52	1.8×10^{-3}	114.29
	k-Fingerprinting	0.97	9.2×10^{-4}	0.84	14.99	6.7×10^{-4}	37.50	0.94	3.74	5.8×10^{-4}	57.14
	Deep Fingerprinting	0.98	2.7×10^{-3}	1.00	2.00	1.6×10^{-3}	68.42	1.00	1.84	1.5×10^{-3}	77.78
	Tik-Tok	0.98	2.5×10^{-3}	0.99	1.18	2.4×10^{-3}	3.45	0.99	1.51	1.8×10^{-3}	36.36

Table 17: Binary classification open world performance results when varying test network seed with classifiers that were trained with network seed = 1.

		Baseline (Test Seed = 1)		Test Seed = 2				Test Seed = 3			
		TPR	FPR	TPR	% Err	FPR	% Err	TPR	% Err	FPR	% Err
Load = 1.5	CUMUL	0.99	7.5×10^{-4}	0.78	27.56	6.7×10^{-4}	12.50	0.97	2.05	1.1×10^{-3}	30.77
	k-Fingerprinting	0.97	2.5×10^{-4}	0.84	15.84	4.2×10^{-4}	40.00	0.93	4.65	2.5×10^{-4}	0.0
	Deep Fingerprinting	0.99	4.2×10^{-4}	0.88	12.12	4.2×10^{-4}	0.0	0.98	1.02	1.7×10^{-4}	150.00
	Tik-Tok	0.99	2.5×10^{-4}	0.88	12.29	1.7×10^{-4}	50.00	0.98	1.19	0.0	inf
Load = 2.0	CUMUL	0.99	1.4×10^{-3}	0.98	1.52	1.3×10^{-3}	13.32	0.82	21.62	1.3×10^{-3}	13.32
	k-Fingerprinting	0.96	3.3×10^{-4}	0.98	1.59	4.2×10^{-4}	20.01	0.81	19.80	2.5×10^{-4}	33.32
	Deep Fingerprinting	0.99	1.7×10^{-3}	0.99	0.01	2.4×10^{-3}	27.59	0.85	16.39	1.3×10^{-3}	39.99
	Tik-Tok	0.98	1.7×10^{-4}	0.97	0.48	5.8×10^{-4}	71.43	0.83	18.06	8.3×10^{-5}	99.98
Load = 2.5	CUMUL	0.98	2.4×10^{-3}	0.72	36.24	1.8×10^{-3}	38.08	0.88	11.37	1.8×10^{-3}	38.08
	k-Fingerprinting	0.96	5.0×10^{-4}	0.74	30.05	9.2×10^{-4}	45.46	0.79	22.87	6.7×10^{-4}	25.01
	Deep Fingerprinting	0.98	3.2×10^{-3}	0.93	6.09	2.8×10^{-3}	14.70	0.95	3.30	2.4×10^{-3}	34.47
	Tik-Tok	0.98	1.7×10^{-3}	0.92	6.51	1.7×10^{-3}	4.99	0.94	4.24	1.6×10^{-3}	10.52

Table 18: Binary classification open world performance results when varying test network seed with classifiers that were trained with network seed = 2.

		Baseline (Test Seed = 2)		Test Seed = 1				Test Seed = 3			
		TPR	FPR	TPR	% Err	FPR	% Err	TPR	% Err	FPR	% Err
Load = 1.5	CUMUL	0.99	1.4×10^{-3}	0.98	1.36	1.0×10^{-3}	41.67	0.96	3.83	9.2×10^{-4}	54.55
	k-Fingerprinting	0.98	3.3×10^{-4}	0.88	10.94	4.2×10^{-4}	20.00	0.82	20.00	5.8×10^{-4}	42.86
	Deep Fingerprinting	0.99	7.5×10^{-4}	0.98	0.68	7.5×10^{-4}	0.0	0.91	8.59	5.0×10^{-4}	50.00
	Tik-Tok	0.99	6.7×10^{-4}	0.98	0.85	8.3×10^{-5}	700.00	0.92	7.05	3.3×10^{-4}	100.00
Load = 2.0	CUMUL	0.99	9.2×10^{-4}	0.99	0.53	9.2×10^{-4}	8.3×10^{-3}	0.97	2.23	8.3×10^{-4}	10.00
	k-Fingerprinting	0.99	1.7×10^{-4}	0.93	5.65	4.2×10^{-4}	60.00	0.96	2.25	4.2×10^{-4}	60.00
	Deep Fingerprinting	0.98	1.8×10^{-3}	0.98	0.72	1.2×10^{-3}	40.01	0.98	0.17	1.8×10^{-3}	4.55
	Tik-Tok	0.98	3.3×10^{-4}	0.96	2.14	4.2×10^{-4}	19.99	0.96	2.07	7.5×10^{-4}	55.56
Load = 2.5	CUMUL	0.96	2.9×10^{-3}	0.84	15.41	3.5×10^{-3}	16.66	0.84	15.34	3.4×10^{-3}	14.63
	k-Fingerprinting	0.96	1.1×10^{-3}	0.77	25.40	7.5×10^{-4}	44.46	0.93	3.41	9.2×10^{-4}	18.18
	Deep Fingerprinting	0.99	1.3×10^{-3}	0.89	11.23	1.7×10^{-3}	28.57	0.96	2.60	2.1×10^{-3}	40.00
	Tik-Tok	0.98	2.8×10^{-3}	0.86	15.01	3.0×10^{-3}	5.55	0.96	2.07	2.8×10^{-3}	0.0

Table 19: Binary classification open world performance results when varying test network seed with classifiers that were trained with network seed = 3.

		Baseline (Test Seed = 3)		Test Seed = 1				Test Seed = 2			
		TPR	FPR	TPR	% Err	FPR	% Err	TPR	% Err	FPR	% Err
Load = 1.5	CUMUL	0.99	3.3×10^{-4}	0.97	2.05	7.5×10^{-4}	55.56	0.93	6.80	1.1×10^{-3}	69.23
	k-Fingerprinting	0.98	8.3×10^{-5}	0.94	3.72	1.7×10^{-4}	50.00	0.92	6.74	8.3×10^{-5}	0.0
	Deep Fingerprinting	0.99	1.7×10^{-4}	0.97	1.72	5.8×10^{-4}	71.43	0.94	4.96	0.0	inf
	Tik-Tok	0.99	2.5×10^{-4}	0.99	0.17	4.2×10^{-4}	40.00	0.98	1.36	0.0	inf
Load = 2.0	CUMUL	0.98	9.2×10^{-4}	0.98	0.20	9.2×10^{-4}	8.3×10^{-3}	0.72	36.89	6.7×10^{-4}	37.50
	k-Fingerprinting	0.96	3.3×10^{-4}	0.91	5.83	8.3×10^{-5}	300.03	0.78	23.19	5.8×10^{-4}	42.86
	Deep Fingerprinting	0.99	1.2×10^{-3}	0.97	1.59	1.1×10^{-3}	7.70	0.84	17.19	3.3×10^{-4}	250.00
	Tik-Tok	0.99	7.5×10^{-4}	0.99	0.02	7.5×10^{-4}	8.3×10^{-3}	0.86	14.26	5.0×10^{-4}	50.00
Load = 2.5	CUMUL	0.98	3.7×10^{-3}	0.71	37.15	3.6×10^{-3}	4.66	0.92	6.72	3.2×10^{-3}	15.38
	k-Fingerprinting	0.97	9.2×10^{-4}	0.69	40.99	6.7×10^{-4}	37.51	0.94	3.92	1.0×10^{-3}	8.33
	Deep Fingerprinting	0.98	2.7×10^{-3}	0.82	19.91	2.6×10^{-3}	3.23	0.98	0.0	2.6×10^{-3}	3.23
	Tik-Tok	0.98	2.5×10^{-3}	0.84	15.83	3.2×10^{-3}	21.05	0.98	0.17	3.4×10^{-3}	26.83