Evaluating Adaptive Padding Against Machine Learning-Based Traffic Fingerprinting in Tor

Yimin Gao
*y*g9bq@virginia.edu

Zhenghong Chen
*q*us9bh@virginia.edu

***Abstract*—Website fingerprinting attacks can compromise user anonymity by analyzing traffic patterns, even in encrypted environments like Tor. In this project, we evaluate the effectiveness of adaptive padding defenses, particularly WTF-PAD, against several machine learning-based fingerprinting models. We explore new padding configurations and analyze the privacy-utility trade-offs. Our experiments show that adaptive padding significantly reduces classification accuracy across models, especially under low-data scenarios. However, Random Forest remains highly effective, suggesting the need for stronger defenses and more realistic datasets.**

I. Introduction

In today's digital age, the Internet has become an indispensable infrastructure for global society that enables information exchange, business transactions, social networking, and access to critical services. At the same time, ensuring the security and privacy of network communications has become increasingly important. Although encryption technologies such as HTTPS and Tor are widely deployed, user privacy remains vulnerable to traffic analysis attacks. Website fingerprinting is one particularly dangerous form of such attacks: adversaries do not need to decrypt traffic content but can infer which websites users visit by analyzing packet size, direction, and timing patterns. These attacks threaten user anonymity and can expose individuals to surveillance, censorship, or even legal consequences, especially in high-risk environments.

To counter these attacks, researchers have proposed various defenses. Website traffic fingerprinting - adaptive defense (WTF-PAD) [1] is one of these mechanisms developed for Tor. It adds padding to traffic traces to obscure features relied on by fingerprinting models, while maintaining a balance between privacy protection, latency, and bandwidth overhead. WTF-PAD represents a significant step towards practical and adaptive fingerprinting defenses.

In this project, we extend the WTF-PAD framework by exploring the design space of padding configurations and evaluating their impact on various machine learning models. We introduce new padding variants, such as *light_padding* and *heavy_padding*, to assess trade-offs between accuracy and bandwidth overhead. We also implement and test multiple classifiers, including Logistic Regression, Support Vector Machine, and Random Forest, on both unprotected and padded datasets. Our analysis highlights both the effectiveness and limitations of adaptive padding under different threat models and learning settings.

II. Background

A. Website Fingerprinting

Website fingerprinting is an attack method based on traffic analysis. The attacker observes the pattern of network traffic and infers which website the user visited. Even if the traffic is encrypted, the attacker can still match the target website visited by the user through metadata characteristics such as packet size, request/response time interval, number of packets, traffic direction.

As an attacker, you usually collect traffic data from many websites and train the model using machine learning classifiers. Finally, you monitor user traffic in real time and extract features to infer the website using the model. On the other hand, as the defender fills the traffic, the traffic delay changes the transmission characteristics. However, it is difficult to defend completely because there is a trade-off between defense and performance.

B. The Onion Router

Tor (The Onion Router) is a network system that achieves anonymous communication through multiple layers of encryption and relay nodes. It protects the user's privacy, hides the user's IP address, and hides the user's visited websites.

When you use Tor Browser to visit a website, your traffic is encrypted multiple times, and your data is transmitted hop by hop through 3 randomly selected Tor nodes. Each layer of nodes can only decrypt one layer of data, and does not know the full path. The website only sees the exit node IP, and the exit node does not know who is visiting, and the entry node does not know which website is being visited, which makes it impossible to directly associate visitors with the websites they visit.

Tor hides the visitor's IP and target website, but traffic characteristics can still be observed externally, such as total traffic and packet patterns. Therefore, website fingerprinting attacks are still a threat to Tor.

III. Related Work

As people pay more and more attention to Website Fingerprinting, this technology has gradually developed different paths in the game between attackers and defenders. From the attacker's perspective, the research focus is on improving the accuracy, robustness and adversarial ability of fingerprint recognition, so as to more accurately identify the websites visited by users in an anonymous communication environment;

from the defender's perspective, the focus is on reducing the differentiability of traffic features through traffic obfuscation, traffic padding, randomization and other technologies, thereby weakening the effectiveness of fingerprint recognition.

A. Website Fingerprinting Process

Website Fingerprinting Process is the process by which attackers collect target website traffic features, train classifiers, and use these models to match and identify target user traffic.

- Data collection: The attacker first collects traffic from known websites, simulates visits to these websites, and records the size, direction, number, time series and other characteristics of each website's data packets when transmitted on the network to obtain a set of "fingerprint libraries".
- Feature extraction: Extract key features from the collected traffic as input to the machine learning model.
- Model training: Use machine learning algorithms to train on the training data set to obtain a classifier that can predict the user's visit to a specific website based on traffic characteristics.
- Traffic monitoring: The attacker monitors the target user's network traffic at the network exit
- Feature matching: Extract similar features from the captured traffic data, input these features into the trained classifier, and determine which known website's fingerprint is closest to it.

B. k-Nearest Neighbors Algorithms

k-Nearest Neighbors (KNN) is a very classic and intuitive machine learning algorithm that is often used for classification problems. KNN does not require an overly complex training process. When new data needs to be classified, if its sample features are found to be very similar to those of its k neighbors, then it belongs to the category that appears most frequently among the neighbors.

Under the premise that there are already a lot of website traffic feature data with labeled categories, we can use KNN to determine which website a new traffic belongs to. This is divided into three steps.

- Calculate similarity: Calculate the "distance" between the new sample and each sample in the training set. Usually use Euclidean distance, Manhattan distance or cosine distance.
- Find the nearest k neighbors: Sort the distances from small to large and select the k samples with the closest distance.
- Voting: Check which category each of these k neighbors belongs to, and the new sample will be classified as the class with the most occurrences.

C. Convolutional Neural Network Algorithms

Convolutional Neural Network (CNN) is a deep learning model that was originally designed to process image data and later applied to time series data and network traffic analysis. A typical CNN consists of convolutional layers, activation functions, pooling layers, and a fully connected layer. Its main operation is divided into the following steps.

- Input data: The input is a matrix consisting of traffic features.
- Convolution operation: Use convolution kernels to slide on the input matrix, generate a "feature map" through weighted summation and bias. Then, perform nonlinear transformation on the convolution result.
- Pooling operation: Take the maximum value in every area to reduce the size of the "image" while retaining key information.
- Repeat convolution and pooling operations multiple times. The more advanced the model learns, the higher the system's judgment ability.
- Output prediction. In the fully connected layer, the extracted features are "flattened" into a one-dimensional vector and sent to the neural network for classification. Finally, a classification probability is output.

D. Padding

Padding is a defense technology that aims to mask the characteristics of real traffic by adding additional packets or bytes to network traffic, making it more difficult for attackers to identify visited websites through traffic patterns. For example, if you originally only transmitted 100 packets, and now send 20 more fake packets, the attacker will see 120 packets, but these extra packets are meaningless and are only used to interfere with the attacker's analysis. There are many ways to pad, but generally they are as follows:

- Fixed-length Padding: Each transmission is padded to the same data size. Its advantages are simplicity and strong defense effect, but it will waste a lot of bandwidth, because even accessing a small web page will be padded to a larger data packet.
- Random Padding: Each transmission is padded to a random length. When the attacker sees the traffic, it is different each time and there is no statistical pattern. Its advantage is that it saves more traffic and is more difficult to speculate than fixed padding. However, it is still limited to some statistical attacks.
- Traffic Morphing: The packet sequence is adjusted through a complex algorithm to make the running packets look like traffic from another website. Due to the need to implement camouflage, the algorithm is complex and difficult to implement, but on the other hand, it can deceive specific attacks compared to other defense solutions.
- Padding during burst: Insert fake burst traffic during the silent period of traffic to blur the burst characteristics in the traffic.

Although padding can defend against fingerprinting, it will increase bandwidth overhead and increase latency.

E. Website Traffic Fingerprinting Adaptive Defense

Website Traffic Fingerprinting Adaptive Defense (WTF-PAD) [1] is an important achievement in the field of Website Fingerprinting defense research. It is an adaptive padding method specifically proposed for Website Fingerprinting attacks in the Tor network. WTF-PAD finds a balance point that can effectively reduce the accuracy of attacks without introducing too much additional traffic and significantly increasing latency. It is mainly divided into the following parts:

- Traffic segmentation: Network traffic itself is often not sent continuously, but in bursts. WTF-PAD first divides the traffic into such bursts.
- Add padding during non-burst periods: There will be a silent period after the burst ends and before the next burst begins. WTF-PAD will randomly add some fake traffic during these silent periods, which can disrupt the attacker's ability to identify websites through burst timing. On the other hand, it will not pad in the middle of the burst, but only during the silent period to avoid increasing transmission delay.
- End condition: If there is no real data to be sent at a certain time, the padding will stop, which can prevent infinite padding. In this process, statistical distribution is used to control the length and frequency of padding.
- Training phase: The defender will collect some target traffic features in advance and then generate a transition probability model. Finally, at runtime, it dynamically decides whether to pad based on the model.

IV. Design Space Exploration of Padded Packet Distributions

To explore the trade-offs between privacy and utility in website fingerprinting defenses, we evaluate several padding configurations that shape the packet timing distributions during encrypted traffic transmission. These configurations are used to adjust how padding is applied across different traffic conditions, directly influencing both bandwidth overhead and resistance to traffic analysis attacks.

The original work [1] introduces several padding configurations, including a *normal* configuration that serves as the baseline for their proposed adaptive padding scheme. WTF-PAD operates by modeling two traffic modes: the *burst* state and the *gap* state, which are controlled by a state machine. This state machine monitors the actual inter-packet delays—if the delay is shorter than a defined threshold, it is considered a burst; otherwise, it transitions to a gap state, as shown in Fig. 1.

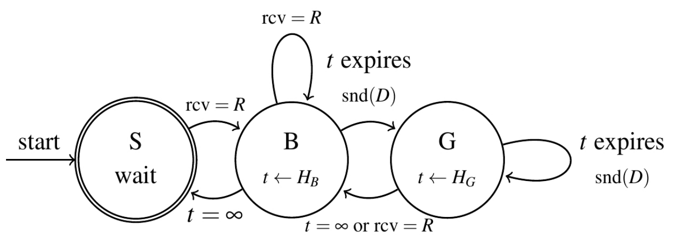
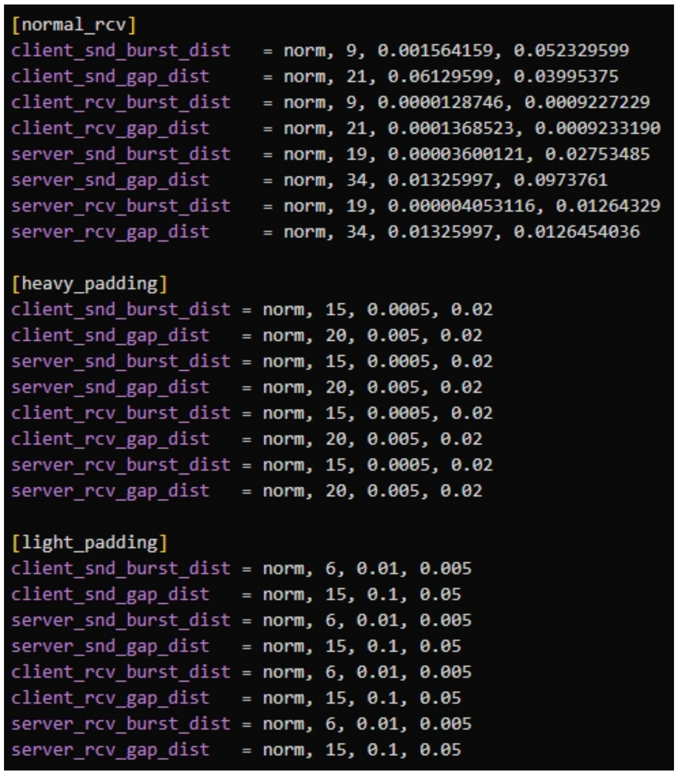


Fig. 1. The state machine algorithm defined in WTF-PAD [1]

Padding behavior is adapted accordingly: in the burst state, padding is inserted less frequently to minimize overhead, while in the gap state, longer or more frequent padding is added to obscure idle periods and prevent leakage of timing information. Each configuration defines a distribution for these states using four parameters: the type of distribution (e.g., normal), the number of packets observed before evaluating a state transition, the mean delay between inserted padding packets, and the standard deviation of the delay distribution.



```
[normal_rcv]
client_snd_burst_dist   = norm, 9, 0.001564159, 0.052329599
client_snd_gap_dist     = norm, 21, 0.06129599, 0.03995375
client_rcv_burst_dist   = norm, 9, 0.0000128746, 0.0009227229
client_rcv_gap_dist     = norm, 21, 0.0001368523, 0.0009233190
server_snd_burst_dist   = norm, 19, 0.00003600121, 0.02753485
server_snd_gap_dist     = norm, 34, 0.01325997, 0.0973761
server_rcv_burst_dist   = norm, 19, 0.000004053116, 0.01264329
server_rcv_gap_dist     = norm, 34, 0.01325997, 0.0126454036

[heavy_padding]
client_snd_burst_dist = norm, 15, 0.0005, 0.02
client_snd_gap_dist   = norm, 20, 0.005, 0.02
server_snd_burst_dist = norm, 15, 0.0005, 0.02
server_snd_gap_dist   = norm, 20, 0.005, 0.02
client_rcv_burst_dist = norm, 15, 0.0005, 0.02
client_rcv_gap_dist   = norm, 20, 0.005, 0.02
server_rcv_burst_dist = norm, 15, 0.0005, 0.02
server_rcv_gap_dist   = norm, 20, 0.005, 0.02

[light_padding]
client_snd_burst_dist = norm, 6, 0.01, 0.005
client_snd_gap_dist   = norm, 15, 0.1, 0.05
server_snd_burst_dist = norm, 6, 0.01, 0.005
server_snd_gap_dist   = norm, 15, 0.1, 0.05
client_rcv_burst_dist = norm, 6, 0.01, 0.005
client_rcv_gap_dist   = norm, 15, 0.1, 0.05
server_rcv_burst_dist = norm, 6, 0.01, 0.005
server_rcv_gap_dist   = norm, 15, 0.1, 0.05
```

Fig. 2. Padding configuration parameters for different modes (burst/gap) across client/server send/receive directions.

In our exploration, we introduce two new configurations as shown in Figure 2: *light_padding* and *heavy_padding*, both derived by tuning the distribution parameters used in the original *normal* configuration. In *light_padding*, we increase the mean and standard deviation of the delay distribution, resulting in less frequent and more varied padding to reduce bandwidth usage while maintaining some protection. Conversely, *heavy_padding* aggressively lowers both the mean and variance, generating more frequent and consistent padding to offer stronger obfuscation, albeit with higher overhead.

These configurations modify the padding distributions for all four communication directions (client/server send/receive) and both states (burst and gap), as detailed in Figure 4. The figure captures the specific parameter values—number of packets before state evaluation, mean delay, and standard deviation—for each direction and mode across the three configurations.

Figure 4 also illustrates the trade-off between utility and privacy: as the padding becomes heavier, classification accuracy (under kNN attacks) decreases, while bandwidth overhead increases. For example, *No padding* yields a kNN accuracy of 90% with no bandwidth overhead, whereas *Heavy padding* reduces the accuracy to 11% but nearly doubles the bandwidth usage.

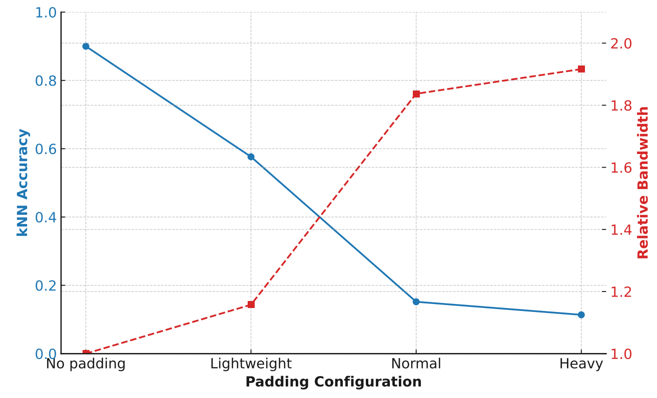


Fig. 3. Trade-off between accuracy and bandwidth across different padding configurations. As padding increases, kNN accuracy decreases while bandwidth overhead increases, highlighting the privacy-utility trade-off.

V. ML-Based Evaluation of Adaptive Padding Defenses

In this section, we evaluate the effectiveness of adaptive padding mechanisms such as WTF-PAD in defending against a range of machine learning-based website fingerprinting (WF) attacks. We replicate and extend the methodology of prior work by extracting the same high-dimensional feature set introduced in [1], and training additional classifiers beyond the original k-Nearest Neighbors (kNN) model. Specifically, we apply Logistic Regression (LR), Support Vector Machine (SVM), and Random Forest (RF) classifiers to examine how different padding configurations affect classification accuracy.

The original implementation from [1] uses a closed-world dataset consisting of traffic traces from 100 monitored websites. Each website was visited 40 times, yielding a total of 4,000 instances. These traces capture packet size and timing information over Tor connections and serve as a controlled benchmark for evaluating website fingerprinting defenses. Despite being synthetic, this dataset is widely adopted due to its reproducibility and well-defined class structure.

A. Linear Models: Logistic Regression and Support Vector Machine

As shown in Fig. 4, both LR and SVM follow the expected downward trend in classification accuracy as padding becomes more aggressive. Under the *No padding* configuration, Logistic Regression achieves approximately 51.82% accuracy, while SVM reaches around 20.27

However, it is also evident that both LR and SVM perform significantly worse than the original kNN model on the unprotected dataset, where kNN achieves 90.27% accuracy. This disparity suggests that the closed-world dataset, though controlled and widely adopted, may not be sufficient to effectively train general-purpose classifiers like LR and SVM. These models typically require larger and more diverse datasets to generalize well, especially in the presence of high-dimensional feature spaces.

In principle, linear models should be more robust to noise introduced by padding, as they are less likely to overfit to low-level timing artifacts compared to instance-based methods like kNN. It can be expected such models to maintain moderate accuracy even in protected settings, particularly if they can leverage residual statistical signals across features. However, our results do not show this behavior, possibly due to limitations of the dataset rather than the models themselves. These findings point to the need for more realistic, diverse, and large-scale datasets, ideally collected from real environments with organic traffic patterns to better evaluate the resilience of modern machine learning models against defenses like adaptive padding. Future work should explore how model capacity, training data volume, and feature representation affect the robustness of classification under protected scenarios.

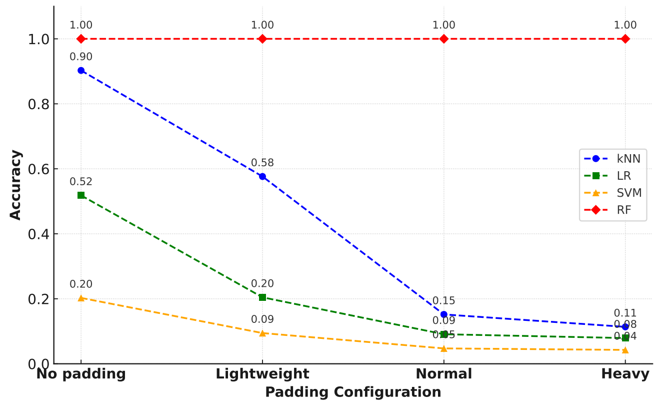


Fig. 4. Classification accuracy under different padding configurations. Heavier padding generally reduces model accuracy, reflecting stronger resistance to fingerprinting attacks.

B. Random Forest Observations and Limitations

TABLE I
Random Forest Testing Accuracy on Unprotected and Heavy-Padding Data

Training Size	Testing Size	Unprotected (%)	Heavy Padding (%)
10	23	80.52	50.70
14	19	93.05	77.89
15	18	100.00	86.50
16	17	100.00	95.12
17	16	100.00	100.00

The Random Forest (RF) classifier demonstrates consistently high performance in our experiments, particularly on the unprotected dataset. When trained with more than 14 instances per class (out of 33), RF achieves 100% testing accuracy, outperforming the baseline kNN model which tops out at 90.27%. This indicates that RF is highly effective at modeling complex patterns in traffic features, even with a modest amount of training data.

While the Random Forest is still able to provide 100% accuracy on the protected data using heavy padding configurations,

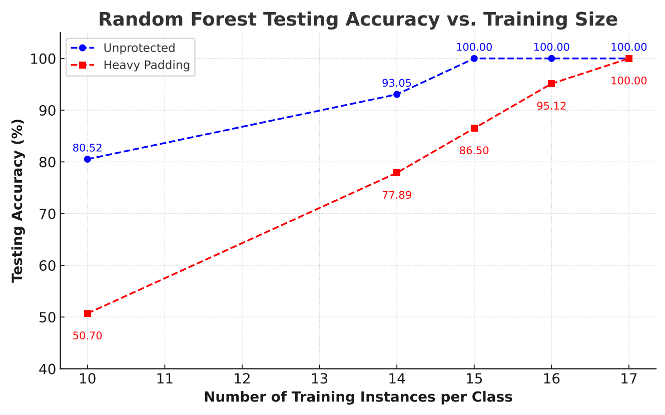


Fig. 5. Random Forest testing accuracy across different training sizes for unprotected and heavy-padded datasets. Padding significantly reduces generalization performance, especially in low-data settings.

we can still observe a clear decline in RF's testing accuracy as the training set shrinks. This shows that adaptive padding makes it harder for the Random Forest model to find useful patterns, especially when there is less training data. Despite this, RF still generalizes well when sufficient training data is available (training size ¿ = 17 out of 33), even in the presence of heavy adaptive padding.

These results reinforce two important insights: first, Random Forest remains a strong baseline for website fingerprinting attacks due to its robustness and ability to exploit subtle feature interactions. Second, adaptive padding is especially effective at lowering model accuracy in low-data settings which shows the need to consider how much data an attacker has when testing defenses.

VI. Reproducibility: Running Baselines and Protected Model Evaluations

Our code is publicly available at *https://github.com/YiminGao0113/wtfpad*, which is a fork of the original WTF-PAD repository (*https://github.com/wtfpad/wtfpad*). We extend the original implementation by adding a dataset preparation script (`dataset.py`) and training scripts for additional machine learning models, including `train_lr.py` for Logistic Regression, `train_svm.py` for Support Vector Machine, and `train_rf.py` for Random Forest. We also fix several typos and bugs present in the original repository, which can be tracked in the latest commits of our fork.

The following outlines the steps required to reproduce the results presented in this work. To reproduce our experiments, follow the steps below for both the baseline (unprotected) and adaptive padding (protected) datasets.

1. Follow the instructions in the repository's `README` file to set up the environment. Then, navigate to the `data/` directory and refer to the `data/README` for guidance on downloading the closed-world dataset required for this project.
2. **Baseline (Unprotected Dataset)**
   First, run the kNN attack script which includes the feature extraction step to generate batch data (i.e., processed features):
   ```
   ./src/knn/run_attack.sh
   data/closed-world-original/
   ```
   This will produce the input features used for model training under **src/knn/batch**. Then, run the training script for the desired model (e.g., Logistic Regression):
   ```
   python3 train_lr.py
   ```
   This command outputs the classification accuracy on the baseline dataset.
3. **Protected Dataset (with Adaptive Padding)**
   To evaluate adaptive padding under different configurations (e.g., `normal_rcv`, `heavy_padding`), first generate padded traces using:
   ```
   python src/main.py -c normal_rcv
   data/closed-world-original
   ```
   This creates padded network traces in the `results/` directory. Next, apply feature extraction as before:
   ```
   ./src/knn/run_attack.sh
   results/normal_rcv_*
   ```
   (or `heavy_padding_*` depending on the configuration used)
   Then, evaluate the trained models again on the protected traces:
   ```
   python3 train_lr.py
   ```
4. **Customizing Training/Test Splits for Random Forest**
   The file `train_rf.py` allows manual configuration of the number of training and testing instances per class. The dataset includes 1,000 sites, each with 33 instances. You can define how many instances are used for training and how many for testing. After adjusting the split, follow the same steps described in Step 2 to evaluate the effectiveness of adaptive padding using the Random Forest model.

VII. Conclusion and Future Work

In this study, we explored the effectiveness of adaptive padding as a defense against website fingerprinting attacks by designing and evaluating multiple padding configurations. By introducing new configurations such as *light_padding* and *heavy_padding*, we analyzed the trade-off between privacy and bandwidth overhead. Our results show that stronger padding significantly reduces model accuracy, confirming that adaptive padding is effective in disrupting traffic analysis.

We also extended the original WTF-PAD framework by incorporating additional machine learning classifiers, including Logistic Regression, Support Vector Machine, and Random Forest. While the performance of linear models degrades as padding increases, Random Forest remains surprisingly strong—achieving 100% accuracy even with heavy padding when trained with enough data. This suggests that the closed-world dataset used in this study may be too limited to challenge more powerful models like Random Forest.

In future work, we plan to evaluate adaptive padding strategies using larger and more realistic open-world datasets that better capture real-world variability. We also aim to investigate why ensemble models like Random Forest perform so well, and whether adding more noise or using different obfuscation methods can reduce their effectiveness. Finally, exploring defenses under more practical conditions, such as limited training data on the attacker's side, may offer further insights into the true robustness of adaptive padding.

VIII. Division of Work:

Yimin prepared the dataset and implemented the training scripts for the machine learning models. Zhenghong was responsible for the design space exploration of the padding configurations. Both Yimin and Zhenghong ran the training experiments, collected results, and collaborated on the final report and presentation.

References

[1] M. Juárez, M. Imani, M. Perry, C. Dıaz, and M. Wright, "Wtf-pad: toward an efficient website fingerprinting defense for tor," *CoRR, abs/1512.00524*, 2015.