DeepCoFFEA: Improved Flow Correlation Attacks on Tor via Metric Learning and Amplification

Se Eun Oh[1] Taiji Yang[2] Nate Mathews[3] James K Holland[2] Mohammad Saidur Rahman[3]
Nicholas Hopper[2] Matthew Wright[3]
[1]Ewha Womans University [2]University of Minnesota, [3]Rochester Institute of Technology
seoh@ewha.ac.kr, {yang6682, holla556, hoppernj}@umn.edu, {njm3308, mr6564, matthew.wright}@rit.edu

***Abstract*—End-to-end flow correlation attacks are among the oldest known attacks on low-latency anonymity networks, and are treated as a core primitive for traffic analysis of Tor. However, despite recent work showing that individual flows can be correlated with high accuracy, the impact of even these state-of-the-art attacks is questionable due to a central drawback: their pairwise nature, requiring comparison between N^2 pairs of flows to deanonymize N users. This results in a combinatorial explosion in computational requirements and an asymptotically declining base rate, leading to either high numbers of false positives or vanishingly small rates of successful correlation. In this paper, we introduce a novel flow correlation attack, DeepCoFFEA, that combines two ideas to overcome these drawbacks. First, DeepCoFFEA uses deep learning to train a pair of feature embedding networks that respectively map Tor and exit flows into a single low-dimensional space where correlated flows are similar; pairs of embedded flows can be compared at lower cost than pairs of full traces. Second, DeepCoFFEA uses *amplification*, dividing flows into short windows and using voting across these windows to significantly reduce false positives; the same embedding networks can be used with an increasing number of windows to independently lower the false positive rate. We conduct a comprehensive experimental analysis showing that DeepCoFFEA significantly outperforms state-of-the-art flow correlation attacks on Tor, e.g. 93% true positive rate versus at most 13% when tuned for high precision, with two orders of magnitude speedup over prior work. We also consider the effects of several potential countermeasures on DeepCoFFEA, finding that existing lightweight defenses are not sufficient to secure anonymity networks from this threat.**

I. Introduction

Tor is perhaps the most well-known anonymous network, used by millions of people each day [1] to hide their sensitive internet activities from servers, ISPs, and potentially, nation-state adversaries. Tor provides low-latency anonymity by routing traffic through a series of relays using layered encryption to prevent any single entity from learning the source and destination of a connection through its content alone.

Nevertheless, it is well known that in low-latency anonymity networks, the timing and volume of traffic sent between the network and end systems (clients and servers) can be used for traffic analysis. For example, recent work applying traffic analysis to Tor has focused on website fingerprinting [2]–[9], identifying which website a client has downloaded based on the traffic between the client and the entry relay.

Perhaps the most fundamental traffic analysis attack on a low-latency anonymity system is the end-to-end flow correlation attack: an adversary observes traffic flows entering the network and leaving the network and attempts to correlate these flows, thereby pairing each user with a likely destination. Such attacks were known and discussed in the context of system designs that predate Tor, such as the Onion Routing network [10] and the Freedom network [11].

The Tor design [12] explicitly acknowledges that such attacks can be effective and concentrates on a more limited threat model. In turn, many published analyses of the security of Tor treat flow correlation as a core primitive and simply account for the fraction of flows that can be observed by an adversary [13]–[24]. These works typically describe methods to increase or limit the fraction of flows that an adversary can observe through some combination of internal manipulation of Tor protocols, manipulation of the Internet routing infrastructure, or network positioning and resources, and assume that flow correlation will work on these observations.

Despite these assumptions, the extent to which end-to-end flow correlation attacks are a realistic threat against the Tor network remains unclear. A problem with directly applying these attacks to Tor traffic is that traffic between the client and entry relay is not identical to traffic between the exit relay and the destination server, due to a variety of factors: multiplexing of encrypted traffic; the use of fixed-size cells to carry data between Tor nodes; and delays caused by buffering, congestion, interaction between circuits, and Tor's flow control mechanisms. For example, when Sun et al. [17] applied Spearman's rank correlation to a small set of entry flows (called *Tor flows* since they are wrapped in the Tor cell protocol), they found that nearly 100MB of traffic per flow was needed to get adequate performance. Nasr, Bahramali, and Houmansadr [25] addressed this limitation by using deep neural networks (DNNs) to learn a more effective Tor-specific flow correlation metric, DeepCorr, that classifies pairs of flows as correlated or uncorrelated with very high accuracy using much less traffic.

Another fundamental limitation, however, on the end-to-end correlation of Tor flows is the *pairwise* nature of the attack. To decide if a flow entering the Tor network and one leaving it are on the same Tor connection, these attacks compute the correlation between the two flow vectors (consisting of packet times and sizes); to deanonymize a set of flows, the attacks must compute the correlation between all possible incoming and outgoing flows. Thus, to deanonymize N Tor connections, they will perform N^2 comparisons.

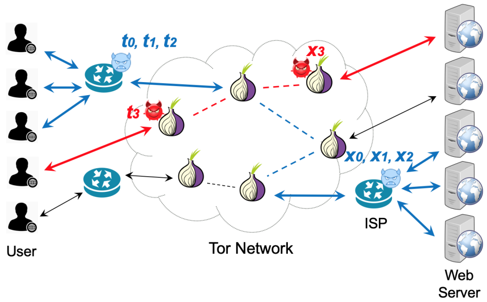


Fig. 1. Threat model of DeepCoFFEA: We have two types of attackers – one controls ISPs (blue) and the other runs their own relays (red).

Naturally, this is computationally expensive when the number of flows at a given moment can be in the tens of thousands. Perhaps more critically, the pairwise comparison results in poor Bayesian Detection Rates (BDRs), since the *base rate* – the probability that both ends of a Tor connection are correlated – is very low (i.e., $\frac{1}{N}$). Any correlation metric with a fixed False Positive Rate (FPR) of ρ will incorrectly classify ρN exit flows as being correlated with each entry flow, so that the probability that a given pair is *actually correlated* given that the metric classifies them that way is $\frac{1}{1+\rho N}$, which approaches 0 as N increases.

This paper presents a novel approach that (i) significantly reduces the cost for each of these comparisons by using a modified *triplet network approach* [26] that first embeds both Tor flows and exit flows into a low-dimensional space, and then uses much more efficient comparisons between these embeddings; and (ii) uses *amplification* [27] to aggregate multiple comparisons for each pair of flows, allowing the FPR to be significantly reduced without the need to learn a new embedding or metric.

To better see the advantage of the triplet network approach, note that DeepCorr learns a pairwise combination of Tor and exit flow features and thus must evaluate a costly DNN on every pair of flows (cost of N^2 DNN runs). In contrast, we develop feature embedding networks to learn two types of embeddings, namely Tor and exit flow embeddings, extending ideas from prior work in website fingerprinting attacks [28]. This architecture learns a pair of functions to generate entry flow and exit flow embeddings separately, and both functions are applied to each flow, which costs just two DNN evaluations per flow (cost of $2N$ DNN runs). The pairwise comparisons can then be done much less expensively using simple distance functions in the embedding space.

The key advantage of amplification, an idea borrowed from randomized algorithms [27], comes from the ability to conduct multiple, partially independent tests of correlation on each pair of flows. To do this, we divide flows into a sequence of k short time segments, or *windows*. We then extract k embeddings per flow (modestly raising our computational cost by a factor of k), and conduct step-by-step pairwise comparisons of Tor and exit flows. Correctly matched flows should be correlated in nearly all k windows, while mismatched flows are likely to only be correlated for at most some of the windows. This windowing approach thus amplifies the difference between true positives (TPs) and potential false positives (FPs).

We call the resulting attack DeepCoFFEA, for Deep Correlated Flow Feature Extraction and Amplification.[1] To sketch our attack at a high level, the network-level adversary monitors Tor flows t_i, between clients and entry guards, and exit flows x_j, between exit relays and destination servers, by running its own relays or controlling autonomous systems (ASes) as shown in Figure 1. After extracting packet timing and size information, the adversary jointly trains two DNN models (G and H) in which Tor flows and exit flows are used as inputs, respectively; G and H should have the property that if Tor flow t and exit flow x are correlated, then $d(G(t), H(x)) \geq \tau$ for some correlation metric d and threshold τ, but if they are not correlated, then $d(G(t), H(x)) < \tau$. Then, the adversary applies G and H to k consecutive flow windows to extract *feature embedding* vectors and computes the pairwise correlation matrix. Finally, if two flows are seen as correlated by d in at least $k - \ell$ windows (for a small threshold ℓ), the adversary determines that the flows are correlated, and otherwise they are not; this exponentially amplifies the difference in TPs and FPs.

Even though both DeepCorr and DeepCoFFEA both perform pairwise correlation, DeepCoFFEA is much more efficient, since G and H generate a total of $2kN$ embedding vectors, while DeepCorr deals with N^2 pairs. This is because DeepCoFFEA compares the embedding vectors for each pair of flows using inexpensive distance computations, while DeepCorr performs expensive DNN evaluations based on pairwise information about the Tor and exit flows together.

This improved strategy enables DeepCoFFEA to simultaneously decrease the required computing resources while increasing the BDR compared to DeepCorr. For instance, when performing a correlation analysis among 10,000 flows, DeepCoFFEA detects an order of magnitude more true correlated pairs (85% vs 8% TPR) for a given FPR (10^{-4}), while decreasing the compute time by *two orders of magnitude*. We also show that the effectiveness of DeepCoFFEA can be transferable across destinations, circuits, and time; in particular, DeepCoFFEA was able to effectively correlate flows on an evaluation dataset that was collected 14 months apart from its training dataset. We also show that, due to amplification, DeepCoFFEA can learn enough useful features to correlate flows that have been protected by unknown padding defenses.

Combined with the DeepCorr results and recent advances in website fingerprinting [5], these results show an urgent need to develop and deploy traffic analysis countermeasures to protect the users of Tor.

II. Background

A. Flow Correlation Attacks

We first review past end-to-end flow correlation studies in Appendix A. Then, we discuss more recent flow correlation techniques, RAPTOR and DeepCorr, which we call state-of-the-art attacks in this paper and compare to DeepCoFFEA.

[1] We pronounce it “Deep Coffee.”

State-of-the-Art Attacks. In more recent work focusing on passive correlation attacks on Tor, Sun et al. [17] presented the RAPTOR attack, which exploited asymmetric traffic analysis to monitor both ends of a higher fraction of Tor connections. They measured the correlation between 50 pairs of flows, each consisting of 300 seconds of traffic, using Spearman's rank correlation algorithm. They found that such previously-studied statistical correlation metrics suffered from a scalability issue, because a long sample of flow data was necessary to yield acceptable results.

Further investigating this issue, Nasr, Bahramali and Houmansadr [25] later described a new flow correlation metric, DeepCorr, which was trained specifically for Tor flows using deep learning. To evaluate DeepCorr, they also collected the largest and most comprehensive correlated flow dataset appearing in the literature, using live Tor network traffic with varying circuits and in different time periods. Although DeepCorr achieved much higher TPR and lower FPR compared to RAPTOR using only 100 packets of flow data, their models yielded a low BDR. For example, based on our experiments using 2,093 testing flow pairs, DeepCorr achieved 81.7% TPR and 0.16% FPR; however, the BDR was 19.8%.

The DeepCorr approach also has significant limitations that might lead skeptics to discount the results of the paper. First, the computational complexity of conducting a correlation attack on a popular network like Tor using DeepCorr is high, since the DeepCorr network operates on pairs of flows to estimate their correlation, so it must be evaluated on $N \times N$ pairs of flows. Additionally, because the metric learns which features are best to predict correlated flows under the current network conditions, it must be re-trained every 3-4 weeks.

B. *Embeddings and Triplet Networks*

Schroff et al. [26] proposed a face recognition system, called FaceNet, based on CNNs. Rather than directly comparing high-dimensional images of faces, FaceNet maps an image to a lower-dimensional vector (which we call a *feature embedding*) so that two images of the same person have very similar embeddings, while images of different people have embeddings with lower similarity. More specifically, FaceNet attempts to train a CNN with a "triplet" loss function that should minimize the Euclidean distance between embeddings of images of the same person and maximize the distance between the embedding of images of different people. Thus, we call this DNN training model a triplet network.

FaceNet is trained by having three copies of the embedding network – called the Anchor (A), Positive (P), and Negative (N) CNNs – with jointly trained and shared weights, namely a unified embedding. Each input to this network consists of three images: A, P, and N triplets. A and P are always selected from the same person class (i.e., positive pair) while A and N are chosen from different people classes (i.e., negative pair). To select the optimal triplets for training, researchers used several *triplet mining* methods: random, hard-negative, and semi-hard-negative. Hard negative samples mean N triplets that are more likely to be close to A triplets.

The objective function in their work aims to ensure a minimum margin or gap α between the L2 distance between A and P embeddings and the L2 distance between A and N embeddings. That is, $\|f(A) - f(P)\|_2^2 + \alpha < \|f(A) - f(N)\|_2^2$ in which f is the embedding and α is the margin being enforced between positive (i.e., (A,P)) and negative (i.e., (A,N)) pairs.

Recently, Sirinam et al. [28] studied the use of triplet networks for website fingerprinting, which they called "Triplet Fingerprinting" (TF). They used cosine similarity scores between embeddings of website traces to compute the triplet loss and adapted the DF [5] architecture to serve as the sub-networks of the triplet network. In the "attack phase" of TF, rather than directly computing the cosine similarity scores of embedding pairs, they trained and tested a k-NN classifier using website trace embeddings with the concept of N-shot learning, which is a technique to use very few training samples to train a classifier. They demonstrated that using embedding networks trained on one set of website traces, they were able to learn a classifier for an independent set of traces with very few samples per class, and achieved comparable or higher accuracy than DF using a large training set.

The embedding portion of DeepCoFFEA is inspired by these two previous works. The main difference between ours and theirs is that DeepCoFFEA has to learn Tor and exit flow embeddings while TF and FaceNet learned one unified embedding (website trace embedding and and face image embedding, respectively). This is because the flows between a client and the Tor entry guard have been heavily modified from the flow between the server and the Tor exit relay, thus our training phase consists of two separate networks: an Anchor network that always operates on packet flows between a client and guard, and Positive and Negative networks (the P/N network) that share weights and always operate on packet flows between an exit relay and server. For distinction, we call a modified triplet network Feature Embedding Networks (FENs). In addition, further modifications were needed for flow correlation rather than website fingerprinting or face recognition. We discuss the Convolutional Neural Networks (CNNs) which we adopted for the architecture of feature embedding networks in Appendix B.

Similar to TF, we use the cosine similarity in the triplet loss, but rather than "N-shot" learning we directly compute the cosine similarity of all flow pair embeddings to classify whether a pair of flows is correlated or not. Further details about our training strategy and the network architecture of DeepCoFFEA appear in Section IV and in particular, we discuss how we extended FaceNet and TF training to DeepCoFFEA in Section IV-A

III. Motivation

In this section, we discuss why and how DeepCoFFEA improves the state-of-the-art correlation metrics.

Problems in State-of-the-Art Attacks. In previous work on flow correlation attacks [17], [25], the adversary targets a correlation metric $d(t, x)$ and computes $d(t_i, x_j)$ for every pair of traces observed in a given time window; if $d(t_i, x_j) \geq \tau$

for some threshold τ, the adversary outputs that t_i and x_j are correlated flows. This approach requires adversaries to collect long flow sequences in the case of RAPTOR [17] or to evaluate an expensive DNN model on all n^2 flow pairs of n connections to use the DNN classifier [25].

For example, in evaluating the RAPTOR attack [17], Sun et al. computed Spearman's rank correlation coefficients for every pair of 50 Tor connections, where each connection captured *300 seconds* of traffic, and selected the exit trace with the highest coefficient as the best match for the Tor trace. DeepCorr [25] trained CNN models to learn a metric $d(t, x)$, and then, in the testing phase, computed this metric using all pairings of 5,000 input and output flows, where the number of associated pairs was 5,000 and non-associated pairs was $5,000 \times 4,999$. In particular, they built feature vectors based on two-dimensional arrays, $F_{i,j}$=[t_i;x_j], where $i, j \in \{1, ..., 5,000\}$ and trained models to minimize the cross-entropy loss between the model output and each pair's label (0 for uncorrelated, 1 for correlated). With this style of training, DeepCorr requires the pairwise combination of each Tor flow and exit flow's features, leading to high space and time complexity. We find, e.g., that DeepCorr requires 5.5 days for testing 10,000 flows on a Tesla P100 GPU.

Furthermore, this high time complexity limits the ability to maximize the data for significant reductions in FPR, which is critical for large-scale traffic analysis. Our amplification technique uses 11 DNN passes through each Tor flow and each exit flow, which remains manageable. Attempting to apply the technique to every pairwise combination in DeepCorr, however, would further exacerbate the computational cost.

A. Our Approach

We introduce a new type of correlation attack, which addresses both the computational complexity and the low BDRs of previous approaches to flow correlation in Tor using a combination of two techniques: Feature Embedding Networks (FENs) and Amplification.

FENs with $2n$ Pairs. Creating feature vectors based on all pairings of input and output flows adds substantial costs to both training and testing in DeepCorr. Each comparison $d(t_i, x_j)$ must evaluate the DeepCorr DNN. In contrast, DeepCoFFEA learns a pair of functions for Tor flows (G) and exit flows (H). To correlate a set of n flows, DeepCoFFEA computes n low-dimensional Tor embeddings $\mathbf{t}_i = G(t_i)$, then computes n low-dimensional exit embeddings $\mathbf{x}_j = H(x_j)$, for a total of $2n$ DNN evaluations, and then uses just these embedding vectors to perform n^2 low-cost cosine similarity computations $cos(\mathbf{t}_i \cdot \mathbf{x}_j)$.

More specifically, DeepCoFFEA jointly trains three networks: *anchor* (A), *positive* (P), and *negative* (N), in which P and N share weights. A learns function G using Tor traces, while P and N learn function H based on exit traces. If we have two flow pairs, (t_1, x_1) and (t_2, x_2), the possible A, P, and N triplets, (a, p, n), will be (t_1, x_1, x_2) or (t_2, x_2, x_1). As shown in this example, x_1 and x_2 appear in both P and N networks, and this negatively affects training. Thus, we implement a modified triplet generator, detailed in Section IV-A. Eventually, using the triplet loss, DeepCoFFEA trains these embedding networks towards maximizing the similarity between a and p while minimizing the similarity (i.e. maximizing the difference) between a and n.

TABLE I
SPACE AND TIME COMPLEXITY COMPARISON OF DIFFERENT FLOW CORRELATION ALGORITHMS WHEN EVALUATING N FLOWS AND EACH FLOW CONTAINS L PACKETS (NOTE THAT $R = \frac{L}{F}$, $R > 1$ AND $q < 1$).

Algorithm	Space	Time
CTA	$O(NL/R)$	$O(N^2L/R)$
CTA+LSH	$O(NL/R)$	$O(N^{(1+q)}L/R)$
DeepCorr	$O(N^2L)$	$O(N^2L)$
DeepCoFFEA	$O(NL/R)$	$O(N^2L/R)$

Thus, even though DeepCoFFEA computes the pairwise cosine similarity of n^2 embeddings, it only needs to evaluate $2n$ FENs. This significantly reduces the complexity of correlation attacks, because the low-dimensional cosine similarity computation between embedding pairs is much less expensive than the deep CNN computation on full flow pairs.

Comparison to CTA. The use of dimensionality reduction to improve the efficiency of flow correlation was also a central idea in Compressive Traffic Analysis (CTA) [29]. To efficiently correlate flows that had been perturbed by network noise, rather than routing through Tor, Nasr, Houmansadr, and Mazumdar applied a Gaussian random projection algorithm to compress IPD feature vectors into fixed-dimension cosine-similar embeddings and then used Locality Sensitive Hashing (LSH) to further reduce the pairwise correlation cost. For N flows with L packets, the sensing basis matrix, $\Phi_{F \times L}$ in which $F < L$ led to F-dimensional feature vectors, which reduced the both time and space complexity by a factor of $\frac{L}{F}$. Furthermore, they extended CTA with LSH to avoid comparing some pairs of embeddings, decreasing the comparison cost from $O(N^2F)$ to $O(N^{(1+q)}F)$, where $1 + q < 2$.

FENs also generate lower-dimensional feature embedding vectors, which reduces the complexity by a factor of $\frac{L}{F}$. Note that we empirically chose F in Section V-C. We note that it might be possible to extend DeepCoFFEA to apply an adapted version of LSH to avoid some comparisons between pairs of embeddings, but we leave this for future work.

We summarize the complexity comparison of CTA, DeepCorr, and DeepCoFFEA in Table I. In Section VI-C, we also evaluate the accuracy of CTA (without LSH, so we maximize the TPR for a given FPR) using the implementation by Nasr, Houmansadr, and Mazumdar [29] and show that when applied to Tor/Exit flow pairs, CTA does not produce usefully correlated results.

Amplification. To reduce the number of FPs, we adapt the concept of amplification from randomized algorithms [27], in which a randomized decision procedure that has any significant gap between acceptance probabilities for positive and negative cases can be repeated multiple times to create a decision procedure with exponentially small false positive and false negative rates. In the context of DeepCoFFEA, we apply amplification by *window partitioning*, in which we divide the flow into several smaller partially-overlapping subflows (windows). Then, we evaluate each window separately and aggregate the results using voting in an ensemble fashion.

By dividing t and x into k discrete windows, and computing the similarity between G and H on the subflows in each window, we end up with k-dimensional vectors comprised of `1`s if the subflows are correlated and `0`s otherwise. These act as *votes* from k windows, which we then aggregate to determine the final decision: if at least $k - \ell$ votes are positive, the flows are correlated, and otherwise they are uncorrelated. For example, if $k = 5$ and $\ell = 1$, and the votes for a given flow pair are `[1,1,1,0,1]`, the pair is predicted as *correlated*. By adjusting k and ℓ for a given anticipated flow set size n, we can push the false positive rate below the base rate of $1/n$, giving a BDR that does not trend asymptotically to 0.

Compared to DeepCorr [25], instead of learning a comparison metric $d(t, x)$, the adversary computes $G(t)$ for every Tor trace and $H(x)$ for every exit trace in k successive windows. We expect only one exit trace, x_j, to line up with the same Tor trace, t_i, with at least $k - \ell$ `1` votes. We show in Section VI-C that amplification can significantly reduce the number of FPs against a pairwise cosine similarity computation, and consequently, DeepCoFFEA becomes more effective with a much lower FPR and higher BDR than the state-of-the-art correlation techniques.

Furthermore, our models do not learn based on labels; rather, they learn statistical differences between correlated and uncorrelated flow pairs, leading to more effective feature extraction as well as a more generalized model. Notably, our empirical study in Section VI-B shows the robustness of DeepCoFFEA against padding-based defenses.

IV. DeepCoFFEA Attacks

In this section, first, we detail how we extended previous work [26], [28] to train the DeepCoFFEA FENs to generate Tor and exit flow embeddings. Next, we describe several methods to compute whether two embedded vectors are correlated and discuss the architecture of the DeepCoFFEA FENs and the hyperparameters involved.

A. Feature Embedding Networks for Correlation Study

To develop FENs for use in the DeepCoFFEA attack, we started with the TF network architecture [28] and adapted it for the flow correlation attack model. As a preliminary step, we tried using their networks directly (that is, having the A,P, and N networks all share weights) when trained with Tor flow, exit flow, and exit flow triplets, but as expected we found that triplet loss did not decrease with training. We then made the following key changes to improve this initial result.

Two Different Networks. Using the TF architecture, the triplet loss did not converge, due to two factors. First, Tor and exit flows are *different* traffic collected at different points in that the Tor trace is collected between the client and guard node and the exit traffic is collected between the exit relay and the web server. Second, Tor traces contained a relatively lower number of packets per window than exit traces, requiring different input dimensions for the two networks. Thus, in contrast to FaceNet [26] and TF, we adopted two separate models: one for the A network and another common model to the P and N networks. This approach led to reduction of the initial loss value, and further, helped achieve decreasing loss curves with training. However, we still ended up with an overall small drop in the training loss. For further improvement, we modified the triplet generator, as described next.

Triplet Epoch Generator. The TF implementation chose triplets from positive and negative examples without regard to whether a particular negative had already been used in a previous input, which led to many exit flows being selected both as a positive (p) and as a negative (n). This quickly froze the triplet loss at some value because some flow pairs were used interchangeably to both maximize and minimize the correlation in the triplet loss function. To resolve this issue, we divided the exit flows into two sets and implemented the triplet generator to choose p from one set and n from the other set. In this way, we guaranteed that p and n were always different within a batch. However, we found that we could obtain a better loss curve when that guarantee was extended to an epoch. Note that we set 128 batches for an epoch in all experiments in the paper. Thus, we kept shuffling the exit trace set and dividing it into two separate pools for every epoch. With this epoch generator, we were able to reach a training loss value closer to zero.

Loss Function. The DeepCoFFEA FENs were trained to minimize the following triplet loss function:

$$max(0, corr(G(a), H(n)) \; - \; corr(G(a), H(p)) \; + \; \alpha)$$

In other words, the goal of FENs is to learn embeddings G and H that satisfy $corr(G(a), H(n)) + \alpha < corr(G(a), H(p))$. The "negative" triplet n plays an important role to train FENs since "easy" negatives in which under the current network parameters we already have $corr(G(a), H(n)) + \alpha < corr(G(a), H(p))$ will not contribute to the loss while "hard" negatives in which $corr(G(a), H(p)) < corr(G(a), H(n))$ will always lead to positive loss. Thus, we used "semi-hard negatives" in which $corr(G(a), H(n)) < corr(G(a), H(p)) + \alpha$. That is, semi-hard negatives are "hard enough" that they contribute to the loss but "easy enough" that adjusting the parameters can push the loss to 0. As such, by tuning α, we could adjust the margin that was enforced between positive and negative pairs. In the following section we describe how we chose the cosine similarity for the correlation metric (i.e., $corr$); we chose α using hyper-parameter tuning, as described in Section V-C.

B. Correlation Methodology

Based on G and H, we extract the Tor and exit flow embeddings, $G(t_{i,w})$ and $H(x_{i,w})$ in which t_i are the Tor flows, and x_i are exit flows, $f_{i,w}$ is the w-th window of flow f_i, and 0$\leq i < n$, and $1 \leq w \leq k$ for n flow pairs and k windows. Then, we compute the correlation values, $d(G(t_{i,w}), H(x_{j,w}))$

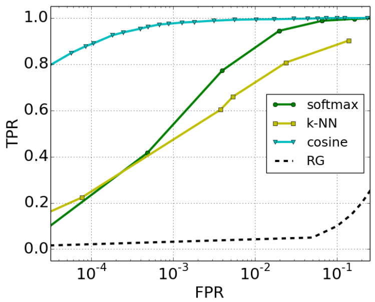


Fig. 2. ROC with different evaluation methods (Note that x-axis is log scale and RG is Random Guess.

for each $0 \leq i, j < n$, and each window $w \in 1, \ldots, k$, for each of the n^2 potential flow pairings. Third, based on these correlation values, we record a 1 vote if $d(G(t_{i,w}), H(x_{j,w}))$ is high enough and 0 otherwise. Finally, we decide that t and x are correlated if they received at least $k - \ell$ 1 votes. We considered three different approaches to computing these similarity "votes": the softmax function, k-NN classifiers, and cosine similarity. In this section, we describe these options.

Softmax. We applied the softmax function, which normalizes the embedding into a vector following a probability distribution with a sum of 1, to the feature embeddings. Based on this probability vector, we could determine a predicted "label" by taking the logit with the highest probability. To investigate the Tor and exit flow pair (t, x), we computed the top-h labels for t and x based on argsort(soft($G(t)$)) and argsort(soft($H(x)$)) in which soft(x_i)=$\frac{e^{x_i}}{\Sigma_{j=1}^{n} e^{x_j}}$. If the matched flow appeared in the top labels, we assume that it is possibly correlated flow. By varying h from 2 to 31, we plot TPRs against FPRs.

k-NN with Clustering. We trained k-NN classifiers using the Tor flows and tested them using the exit flows. We further trained the classifiers using the exit flows and tested them using the Tor flows to find the best setting. For either direction, we first had to label the flows in the training set before training. We explored k-means [30] and Spectral clustering [31] to label training flows. For both clusters, we varied k from 2 to 295, resulting in 2-295 labels to be trained. After training k-NN models using Tor (or exit) flows labeled by clustering, we evaluated the models using exit (or Tor) flows. We then conducted the pair-wise comparison over the predicted labels of exit flows and labels of Tor flows decided by the clustering algorithm. If the labels are the same in the correlated flows, they are TPs. If they are the same in the uncorrelated flows, they are FPs. Based on the preliminary experiments, we decided to train k-NN models using Tor flows and further test them using exit flows. We also empirically chose Spectral clustering with the cosine similarity to label the Tor flows and computed TPRs and FPRs at various k from 2 to 295.

Cosine Similarity. The cosine similarity measures the cosine of the angle between two vectors projected in a multi-dimensional space. It captures the angle, not magnitude, such as the Euclidean distance. Since the FENs were trained using triplet loss based on the cosine similarity (cos), we naturally studied this similarity score as the similarity metric for DeepCoFFEA. That is, we computed the similarity scores for each window w of all Tor and exit embedding pairs, $(t_{i,w}, x_{j,w})$, in which $0 \leq i, j < n$ for n testing flow pairs. For each pair, if $cos(G(t_{i,w}), H(x_{j,w})) \geq \tau$ for some threshold τ, we recorded a vote of 1 and 0 otherwise. By varying τ, we drew the ROC plot. We present how we chose τ in Section V-E.

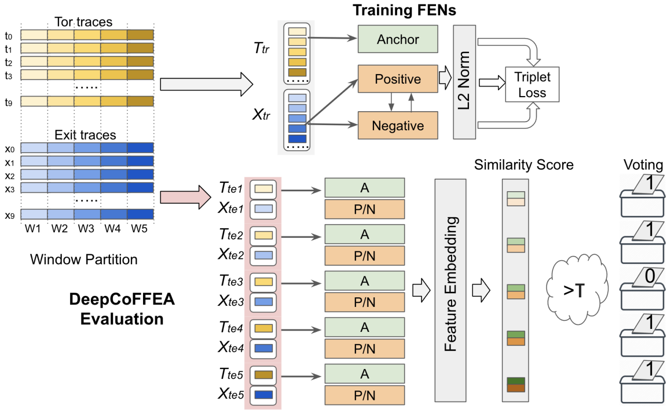


Fig. 3. Example DeepCoFFEA Scenario: In this example, we had ten (t_i, x_i) flow pairs and five windows (W1,...,W5). First, we performed the non-overlapping window partition to generate two training sets, T_{tr} and X_{tr}, and ten testing sets, T_{te1},..,T_{te5},X_{te1},..X_{te5}. Then, we trained the DeepCoFFEA feature embedding network (FEN) with T_{tr} and X_{tr} and generated the feature embedding vectors using A and P/N models for each testing set, (T_{te_w},X_{te_w}) where w=1,...,5. We then computed the pairwise cosine similarity scores for each testing window and voted with 1 if the score was greater than τ or 0 otherwise. Finally, we aggregated those results and determined that the flow pair was correlated if it had at least four 1 votes.

As shown in Figure 2, the cosine similarity approach outperformed other methodologies. The cosine similarity was clearly more effective to distinguish the correlated flows from uncorrelated flows than the softmax-based distinction and the clustering did not extract effective correlation labels. We adopted the cosine similarity when evaluating DeepCoFFEA throughout the remainder of the paper.

C. FEN Architecture

As shown in previous work [5], [6], [25], [25], CNNs typically learn more useful features for analysis of Tor traffic. Thus, we further explored two different architectures, one based on 1D convolutional (Conv) layers and the other based on 2D Conv layers. We adopted the DF [5] and DeepCorr [25] architectures for these two architectures since they have been effective in generating website fingerprints based on traces between the client and the guard node and correlational flow features based on inflow and outflow to the Tor network. We empirically concluded that the 1D CNN-based DF architecture performed better; more specifically, we were unable to reduce the triplet loss below 0.01 with the DeepCorr architecture.

As shown in Figure 3, the two FEN models are learned, and in the testing phase, the A network maps inputs from Tor flows to feature embedding vectors, while the P/N network maps inputs from exit flows to feature embedding vectors. Each FEN consists of four 1D Conv blocks, including two 1D Conv layers, followed by one max pooling layer. After that, there was a fully connected output layer, which generated the feature embedding. We chose the input and output dimensions, optimizer, and learning rate based on parameter optimization as shown in Table II.

Finally, we outline an example DeepCoFFEA evaluation scenario in Figure 3.

V. DeepCoFFEA Experiment Details

In this section, we detail the experimental settings including the dataset, features, window partition, hyper-parameter optimization, and metrics to evaluate DeepCoFFEA in Section VI.

A. Input Preprocessing

Data Collection. To the best of our knowledge, Nasr, Bahramali, and Houmansadr [25] collected the most comprehensive flow correlation dataset collected on the Tor network and reflects the effect of different circuit usage and time gaps between training and testing data. After reparsing the raw captures of DeepCorr set instead of using the preprocessed data in which the outgoing and incoming packets were separated, we selectively chose the flow pairs to ensure that all connection destinations are unique, resulting in 12,503 flow pairs. We call this set the DeepCorr dataset.

However, we were interested in evaluating additional scenarios that could not be addressed with this dataset alone, including a training/testing gap longer than three months, training and testing sets with no mutual circuits, and the ability to test with a large non-training set. Thus, we collected our own dataset (the DCF set) mimicking the collection methodology devised by DeepCorr, with some small differences.

First, we used physical machines to run both Tor clients and the SOCKS proxy servers collecting exit flows while the original method ran clients inside VMs. Second, we crawled 60,084 Alexa websites using 1,051 batches in which the circuits were rebuilt every batch. We captured for a full 60 seconds for every sample to ensure that any dynamic content loaded after the initial page load was included in the capture.

Third, we captured packets at the Ethernet layer while DeepCorr collected it at the IP layer. This change aimed to remove over-MTU-sized packets that appeared in the DeepCorr set and better resemble traffic as it would be seen on the wire. To provide transferability between datasets we combined adjacent packets with IPD values of 0 (e.g., the packets arrived/sent at the same moment) when training our model. Lastly, we filtered out flows whose packet counts were less than 70, which was shorter than DeepCorr which applied 300, 400, and 500 when evaluating the model against traces with 300, 400, and 500 packets, respectively. In this way, the DCF set caused more flows to be padded or truncated than the DeepCorr set. As shown in Figure 11 in Appendix D, we explored various filters to remove shorter traffic whose length was lower than 70, 300, and 500, and the DeepCoFFEA performance was not significantly affected by the size of the filter.

Among 60,084 flow pairs, we selected pairs whose per-window packet count was greater than 10, which led to 42,489 pairs in total. Note that to show the model transferability across different circuits and sites, all 42,489 connections involved a unique destination. This ensures that there should be no overlapping destinations and circuit usage between training and testing sets.

In addition, we also collected 13,000 flows using the obfs4 Pluggable Transport for the evaluation against defended flows. Furthermore, we captured some older flows crawled in April 2020 for the experimental scenario in which there is a long-time gap between training and testing data.

To further evaluate DeepCoFFEA against light-weight website fingerprinting defenses including WTF-PAD [32] and FRONT [33], we also simulated defended flows by using official implementations shared by researchers [34], [35].

Window Pooling. In contrast to DeepCorr, in which n correlated flow pairs could be used to create up to n^2 input pairs, in DeepCoFFEA each correlated flow pair can only produce at most one triplet, creating many fewer training examples for the FEN models. Instead of feeding all possible pairs to FENs, for each anchor point, we selected *one* semi-hard negative that was more useful through triplet mining. For example, if we have three flow pairs, (t_1,x_1), (t_2,x_2), and (t_3,x_3), our approach results in three input pairs such as [(t_1,x_1,x_2), (t_2,x_2,x_1), (t_3,x_3,x_2)]. Based on our epoch generator, only the first and third triplets can appear in the same epoch.

By partitioning the flows into k windows, we were able to pool the correlated pairs across windows, increasing the FEN training set size by a factor of k. More specifically, we divided each flow based on a predefined time interval chosen during the tuning (Section V-C), and constructed the training set. Based on 10,000 pairs of training flows, we built a training set using all k flow windows, resulting in $k \cdot 10,000$ correlated flow pairs. In contrast, we constructed k testing sets separately for each window in which there are n pairs ($2094 \leq n \leq 10,000$). Thus, this setup resulted in testing DeepCoFFEA using n Tor and exit flows across k windows. We detail the window partitioning in Section V-B.

Features. As in DeepCorr [25], we used inter-packet delay (IPD) and packet size information to construct the feature vectors from the flows. Since we chose 1D CNN models for DeepCoFFEA, we constructed one-dimensional arrays, $v_i = [I_i||S_i]$ for the bi-directional Tor and exit flows by concatenating the vector of IPDs and packet sizes. Here, the vector I_i consists of upstream IPDs (I^u) and downstream IPDs (-I^d) and the vector S_i is comprised of upstream packet sizes (S^u) and downstream packet sizes (-S^d).

We also evaluated DeepCoFFEA based on other combinations, such as $v_i = [I_i^u||S_i^u||I_i^d||S_i^d]$ or $[I_i^u||I_i^d||S_i^u||S_i^d]$ and these feature vectors led to much worse performance, perhaps because the local interleaving of upstream and downstream traffic allowed the (local) CNN filters to extract more relevant features. Similarly, we tried training FENs based on feature sets using only the IPD vectors or the packet size vectors and found that these were less effective as well.

Lastly, we evaluated DeepCoFFEA using features based only on packet sizes. The cosine similarity scores between

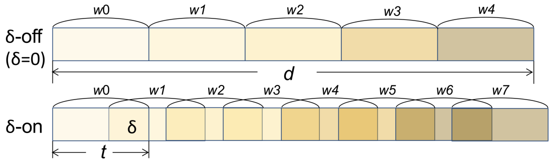
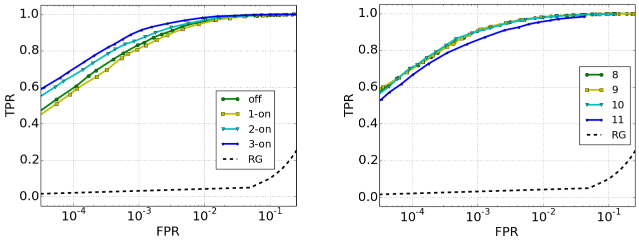


Fig. 4. Overlapping window partition.



(a) ROC for different δ applied to the window partition when loss ≈ 0.006.

(b) ROC for various voting thresholds when loss ≈ 0.006.

Fig. 5. DeepCoFFEA parameter tuning (Note that x-axis is log scale and RG: Random guessing).

correlated flow pairs were higher when considering both packet timing and size information, indicating that the IPD sequence delivered informative features for correlated flows.

B. Window Partitioning

We tested two partition strategies, one based on time intervals and the other based on the number of packets, and then decided to use time intervals as the window *interval*, which yielded a better triplet loss curve. This is because Tor flows generally had fewer packets per time period than exit flows, so that when using windows based on a fixed number of packets, the flows for the second and subsequent windows no longer corresponded to the same traffic. We empirically determined the number of windows and the interval in Section V-C.

We further explored overlapping window partitioning to create overlapping windows with some interval overlap (δ) between subsequent windows, which we refer to as δ*-on* partitioning. As shown in Figure 4, when the window interval length is t and total flow duration is d, then δ*-off* partitioning leads to $\left\lceil\frac{d-t}{t}\right\rceil+1$ intervals, where window w is the interval $[t\times w, t\times w+t)$. In contrast, δ-on results in $\left\lceil\frac{d-t}{t-\delta}\right\rceil+1$ intervals, where window w is the interval $[(t-\delta)\times w, (t-\delta)\times w+t)$. For example, when $t = 5$, $d = 25$, and $\delta = 3$, the 5 δ-off windows are the intervals [0,5), [5,10), ..., [20,25), while the 11 δ-on windows are the intervals [0,5), [2,7), ..., [20,25). As such, with δ-on, we create more windows, leading to more training flow pairs and boosting difference in TPs and FPs by aggregating results from more windows. In other words, δ-on increases the amplification of DeepCoFFEA, improving the performance dramatically, as demonstrated in Section VI-B.

C. Hyperparameter Optimization

The choice of hyperparameters is crucial to improve the DeepCoFFEA performance, and particularly the behavior of the FENs, to result in lower triplet loss. Thus, we explored the parameter search spaces shown in Table II using one Nvidia RTX 2080 and one Tesla P100 GPU. Although we used the DCF set to tune these parameters, we noticed that the DeepCoFFEA performance is not very sensitive to the dataset when choosing these parameters. One exception is the input dimension; the traffic rate per window can change substantially and thus, while other parameters in this section could be used without further search, the input dimension should be adjusted for new network conditions. Here we give more details for how we selected the parameter ranges.

Window Setting Parameters. The length of flow input to FENs should be selected to optimize the model performance within an acceptable range of training costs.

We first investigated the minimum duration of the first window interval whose median packet count was 100 packets. Note that 100 packets were chosen in DeepCorr as the minimum flow length they explored [25]. After computing the median packet counts for intervals between 2 and 5 seconds in Table VI of Appendix C, we determined that five seconds would give the best performance.

Furthermore, we set the search space for the total flow duration up to 35 seconds since the packet count for windows after 35 seconds became lower than 100 packets as shown in Table VII of Appendix C. Second, with $\delta = 0$, we explored various total flow durations since the DCF performance started degrading after 25 seconds. Third, based on Table VIII of Appendix C, we used the minimum and maximum flow lengths in 11 windows as the search space for Tor flow length and exit flow length, which were [106,501] for Tor flows and [244,855] for exit flows.

Window Partition Parameter. First, we investigated the impact of δ-on/off settings for varying δ. As discussed in Section V-B, δ-on creates $\left\lceil\frac{25-5}{5-\delta}\right\rceil+1$ windows, that is, it creates 6 windows for $\delta = 1$, 8 windows for $\delta = 2$, and 11 windows for $\delta = 3$. Note that we omitted $\delta = 4$ since with 21 windows the resulting cosine similarity matrix for 218,610 training flow pairs was too large to compute using our resources, so we could not select semi-hard negatives.

We reported results of $\ell = 2$ for the 3-on and the 2-on and $\ell = 1$ for the 1-on setting in Figure 5a. Even though $\delta = 1$ did not improve the DeepCoFFEA performance over δ-off settings, by increasing δ, DeepCoFFEA was benefited by the enhanced amplification capability with more training flow pairs as well as more voting results. Figure 5a shows that a larger δ typically led to better ROC curves with higher TPRs using more votes, indicating that more resourceful adversaries could further improve the 3-on results by using the 4-on setting. Based on the results shown in Figure 5a, we evaluated DeepCoFFEA in the 3-on setting throughout Section VI.

Model Parameters. As the triplet loss aims to separate the positive pair from the negative by a distance margin, α, we first tuned α to maximize the distinction between the cosine similarity scores of the correlated pairs and those of the uncorrelated pairs. With α=0.1, the FENs attained the lowest loss and we noticed that TF [28] also used the same value.

Second, as discussed in Section IV-A, we implemented our own triplet epoch generator which selected triplets for the positive and negative networks from separate pools. We

TABLE II

CHOSEN HYPER-PARAMETERS AND SEARCH SPACES USED IN THE HYPER-PARAMETER OPTIMIZATION.

Param	Chosen Param	Search Space
Tor flow size	500	$\{106, \ldots, 501\}$
Exit flow size	800	$\{244, \ldots, 855\}$
total flow duration	25	$\{20, \ldots, 35\}$
δ	3	$\{0, 1, 2, 3, 4\}$
α	0.1	$\{5 \cdot 10^{-2}, \ldots, 5 \cdot 10^{-1}\}$
Epoch generator	1	$\{1, \ldots, 10\}$
Output node	64	$\{10, \ldots, 100\}$
Optimizer	SGD	SGD, Adam
Learning rate	10^{-3}	$\{10^{-3}, \ldots, 10^{-4}\}$
Correlation metric	cosine	cosine, euclidean
Vote	9	$\{8, 9, 10, 11\}$

further tuned the number for how frequently those separate pools needed to be updated (i.e., shuffled and divided into two pools). Eventually, the FEN performance improved when updating the pools more frequently. Thus, we recommend updating them every epoch rather than every 2-10 epochs.

Finally, we further tuned the learning rate of SGD optimization and the number of output nodes, which is the dimension of the feature embeddings generated by the trained FENs.

Correlation Parameters. We had to decide a correlation metric to be computed in the triplet loss function, so we explored both cosine similarity and Euclidean distance metrics, as proposed by previous research [26]. Interestingly, with the Euclidean distance function, the loss never decreased.

Lastly, after computing the cosine similarity scores for all testing pairs for each window, we found that DeepCoFFEA performance was comparable with nine and 10 votes across 11 windows while the performance somewhat dropped with 8 and 11 votes (Figure 5b). So, we decided that the flow pair would be correlated if it had at least nine 1 votes across 11 windows since the DeepCoFFEA achieved slightly more TPs against low FPRs less than 10^{-4}.

D. Metrics

In this section, we introduce the definitions of the TPR, FPR and BDR metrics used in Section VI.

- TPR: The true positive rate is the fraction of correlated flow pairs that are classified as "correlated".
- FPR: The false positive rate is the fraction of uncorrelated flow pairs that are classified as "correlated".
- BDR: The Bayesian detection rate in the flow correlation is the probability that a correlated pair is actually "correlated" given that the correlation function detected it as "correlated" and it can be computed as:

$$P(P|C) = \frac{P(C|P)P(P)}{P(C|P)P(P) + P(C|N)P(N)} \ ,$$

where $P(P)$ is the probability that the pair is correlated, $P(C)$ is the probability that the pair will be decided by the flow correlation as correlated, and $P(C|P)$ and $P(C|N)$ are replaced with TPR and FPR, respectively. We note that precision is often used instead of BDR. We chose BDR in this paper since the impact of the base rate is shown more clearly in the computation.

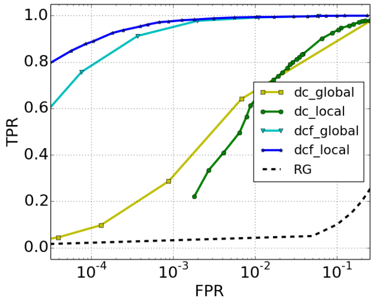


Fig. 6. DeepCoFFEA (loss ≈ 0.0018) and DeepCorr performance with local and global thresholds.

In addition, we measured the performance of state-of-the-art attacks and DeepCoFFEA using ROC curves by varying the correlation threshold parameter.

E. Similarity Thresholds

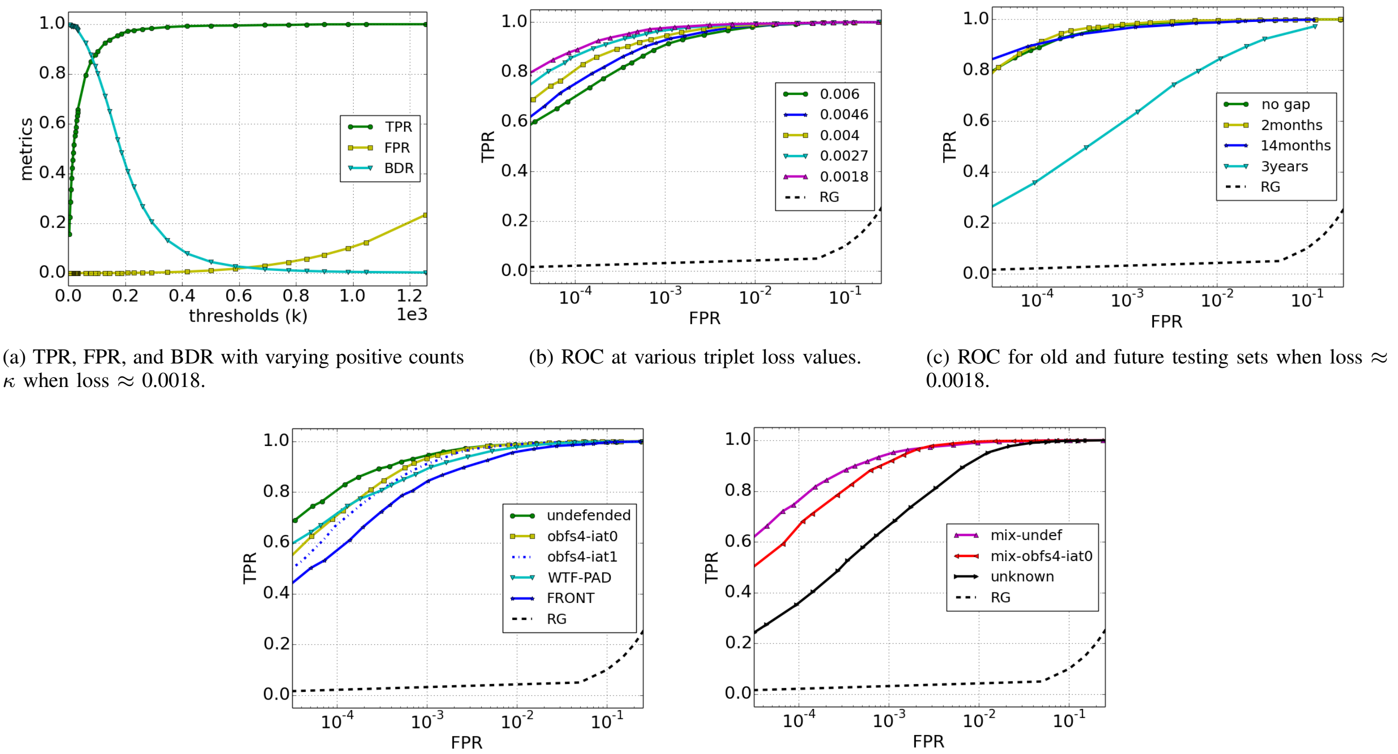
In DeepCoFFEA, the embedded feature correlation threshold τ acts to control the number of exit traces that are classified as "possibly correlated" with each Tor trace in a given time window. We can either control this number *indirectly* by setting a *global* threshold τ that is applied to all cosine similarities (so that, generally, as τ increases, fewer pairs will be classified as possibly correlated); or we can control this number *directly* by classifying only the closest κ exit traces to a given Tor trace window t_i (as measured by $cos(G(t_i), H(x_j))$) as possibly correlated. This latter choice corresponds to computing a *local threshold* for each t_i by sorting the list $d_{i,1}, \ldots d_{i,n}$, where $d_{i,j} = cos(G(t_i), H(x_j))$, and selecting the κ^{th} element as the threshold for t_i.

We explored both approaches and found that, as shown in Figure 6, the number of TPs decreases more rapidly with higher τ (those that yield FPR less than 10^{-3}) than with higher κ, most likely due to some windows in which many flows have similar embeddings. By selecting the thresholds *locally* (i.e., based on the pairwise score distribution of each Tor flow), DeepCoFFEA is able to detect more correlated flows while controlling the number of FPs.

We also evaluated DeepCorr using local thresholds even though they adopted global thresholds in their paper. As shown in Figure 6, the use of κ did not improve the performance of DeepCorr and FPRs did not decrease below 10^{-3}, indicating that most correlation scores for uncorrelated pairs were similar to scores of correlated pairs and decreasing κ did not help reduce the false positives. Thus, we used τ for DeepCorr and adopted κ for DeepCoFFEA as the curve parameter to generate the ROC curves of DeepCoFFEA in Section VI.

VI. EXPERIMENT RESULTS

In this section, we explore the effect of various configurations of DeepCoFFEA on its performance and compare the effectiveness and efficiency of DeepCoFFEA to state-of-the-art attacks.



(a) TPR, FPR, and BDR with varying positive counts κ when loss ≈ 0.0018.

(b) ROC at various triplet loss values.

(c) ROC for old and future testing sets when loss ≈ 0.0018.

(d) ROC for various defenses when loss ≈ 0.004.

(e) ROC for various adversary models against defenses when loss ≈ 0.004.

Fig. 7. Performance of DeepCoFFEA across various settings (Note that RG is Random Guess and all x-axes except Figure 7a are in log scale).

A. *Experimental Settings*

We implemented FEN models using Keras [36] with Tensorflow [37] backend and used one Tesla P100 GPU with 16GB memory to train and test DeepCoFFEA [2]. We selectively chose 12,094 flow pairs from the DCF set, from which we use 10,000 pairs for training data and the other 2,094 pairs for testing data, with no overlapping circuit usage between the two sets. In addition, all 12,094 connections went to unique sites (i.e., 12,094 destinations). With this setting, we aim to demonstrate that DeepCoFFEA successfully detects correlated flows that were collected using *arbitrary* circuits or sites. While we used 2,094 test connections for most of the experiments in this section, DeepCoFFEA performs well even with 10,000 flows, as shown in Figure 8c.

It is likely in practice that users visit sites that are in the training set, since samples from the most popular sites are used for this purpose. Thus, DeepCoFFEA performance shown in all experiments in this section may be better in practice. Moreover, as Nasr, Bahramali and Houmansadr [25] point out, collecting exit flows using a SOCKS proxy may add some latency, meaning that exit traces could be more distinct from each other in practice, leading to further improvement.

B. *DeepCoFFEA Effectiveness*

We evaluate the impact of the different settings and ideas behind DeepCoFFEA on the overall effectiveness of the attack.

TABLE III
TRAINING TIME TO REACH SPECIFIED TRIPLET LOSS VALUES.

triplet loss	0.006	0.004	0.003	0.002
training time (days)	0.96	3	5	14

Amplification. One of the key advantages of DeepCoFFEA is amplification to reduce the number of FPs. This study sheds light on the impact of amplification by evaluating DeepCoFFEA on a per-window basis. Table IX of Appendix D shows TPRs and FPRs when evaluating DeepCoFFEA using each of 11 windows; both metrics were consistent across windows with TPR at 97-98% for 12% FPR. Then we applied the voting strategy to aggregate all 11 results, and DeepCoFFEA yielded 97.99% TPR and 0.13% FPR. Such a significant drop in FPR indicates that only a few positive predictions on unmatched pairs were aligned across all 11 windows, which was exploited by the amplification technique to reduce the number of FPs.

Threshold Parameter. We evaluated the effect of the positive correlation parameter κ, described in Section V-E. We computed the number of TPs and FPs at for κ ranging from 4 to 1,255. Figure 7a shows that both the TPR and FPR increased by increasing the threshold, while the BDR decreased.

Triplet Loss. When training FENs, the triplet loss decreased monotonically with training time; our experiments halted training when the loss hit various loss values from 0.006 to 0.0018. In this study, we investigate the performance of DeepCoFFEA when training stops at different loss values; this experiment gives insight on choosing a stopping point for training FENs.

[2] The source code and datasets are available on https://github.com/traffic-analysis/deepcoffea

Figure 7b shows that DeepCoFFEA performance continually improved as the triplet loss decreased. We stopped the training at 0.0018, since the loss decreased very slowly after 0.003. In Table III, we also report the training time required to achieve different loss values. There is a clear trade-off between model quality and training time.

Time-Separated Testing Sets. We explored how a separation between the time (and thus, incidentally, Tor software versions[3])

of training set collection and test set collection impacted the correlation ability of DeepCoFFEA. For this study, we trained FENs on data from June 2021, while constructing three testing sets: one from August 2021 (*2-month-newer set*), one from April 2020 (*14-month-old set*), and the DeepCorr set collected in January-April 2018 (*3-year-old set*).

As shown in Figure 7c, DeepCoFFEA performs approximately the same for the 2-month-newer set and the 14-month-old set as a baseline test with no time gap. The performance was significantly worse using the 3-year-old set. This dramatic shift may be explained at least in part by the differences in the collection process between the DeepCorr set and the other two datasets. We also expect that Tor flows simply changed more significantly during this three-year window. Nevertheless, DeepCoFFEA still detected some correlated flows correctly with 92% TPR and 3% FPR.

These findings indicate that even though the packet size and timing features could change as network conditions evolve over the time [38], the statistical difference between correlated and uncorrelated features could be similar even after 14 months. As FENs are trained to maximize the difference between correlated and uncorrelated flows, the ability to generate the highly correlated flow features can be persistent even after more than a year. Thus, even though the training cost required to ensure better model quality was considerable (Table III), this may be amortized over long periods, as DeepCoFFEA does not require frequent retraining.

Defended Traces. Nasr, Bahramali and Houmansadr [25] evaluated DeepCorr against traces protected by the obfs4 pluggable transport (PT), the PT recommended by the Tor Project for censorship evasion [39]. obfs4 encrypts and transforms the traffic between the client and the guard node to avoid potential traffic-analysis-based censorship. In particular, it obfuscates packet sizes by appending random padding. obfs4 also provides an IAT (Inter-Arrival Timing) mode that randomizes inter-arrival times. We investigated obfs4 with both IAT mode on (*obfs4-iat1*) and off (*obfs4-iat0*).

Security researchers have also investigated several website fingerprinting defense mechanisms designed to mask traffic patterns in Tor [33], [35], [40]. As the adversary conducts traffic analysis on the Tor flow between the client and entry guard, these defenses hide the total packet statistics by adding padding packets according to various schemes. For example, WTF-PAD [32] seeks to hide statistically unlikely (and thus distinguishing) IPDs between packets by strategically adding padding, while FRONT [33] adds more randomness to the amount of padding and the location where it is injected.

Since these WF defenses reduce the similarity between Tor flows, we might expect that they also make correlated flow features less effective. Note that padding packets are only seen on the Tor flows, but not the exit flows, making the matched flows look less alike than in undefended Tor. Therefore, in this study, we evaluated the effectiveness of DeepCoFFEA against the obfs4 PT, WTF-PAD and FRONT. To the best of our knowledge, this is the first investigation of the effectiveness of WF defenses against end-to-end flow correlation attacks.

Using the same features discussed in Section V-A, we trained three different DeepCoFFEA models using 10,000 defended traces collected with each of obfs4-iat0, obfs4-iat1, WTF-PAD[4], and FRONT[5] defenses. As shown in Figure 7d, the DeepCoFFEA TPR decreased across all defenses, while still maintaining very low FPRs. DeepCoFFEA achieved TPRs above 50% for FPR of 10^{-5} for all defenses besides FRONT. FRONT had the most success defeating DeepCoFFEA, since the obfuscation level in each window is random, making the correlation pattern across windows less consistent.

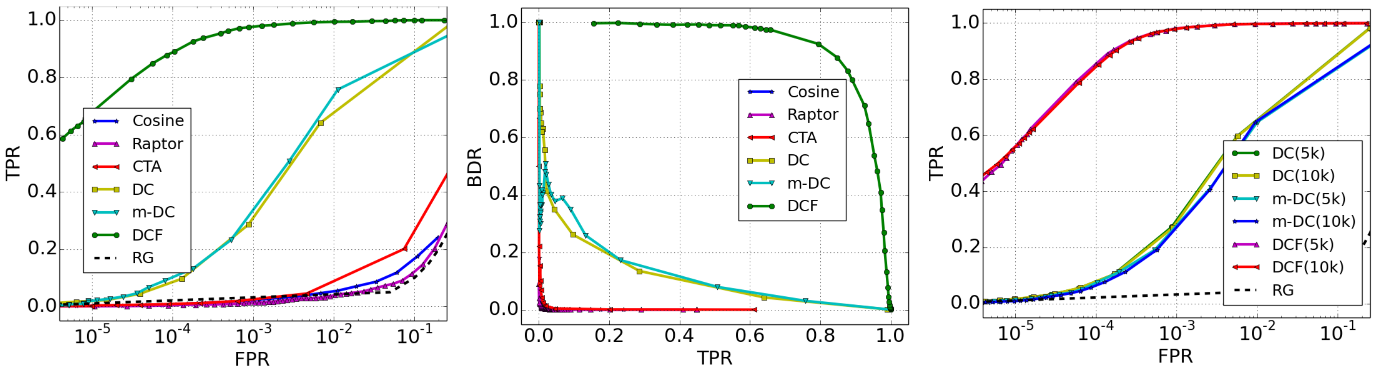
An adversary may face the possibility of some users of interest who apply a defense while most other users take a default setting of not applying the defense. This scenario allows us to examine whether FENs can be extended to detect different types of flows. We trained the model using a mix of defended (i.e., obfs4) and undefended flows with a ratio of 1:4 and then tested the model using two different testing sets, one for each type of flow. As shown in Figure 7e, even though the model trained using a mix had somewhat worse performance against undefended traces (the mix-undef curve in Figure 7e), FENs were still capable of generating effective embedding vectors for both undefended and defended traces. This result indicates that the inclusion of defended traces marginally impacted the correlation capability and shows the potential of unified FENs to detect both types of flows successfully.

Lastly, we explored a more difficult setting for DeepCoFFEA, in which we trained FENs using undefended traces and then detected correlated flows of *unknown* defenses. We note that this is a rather artificial attack scenario, since it is contrary to Kerckhoff's principle. Figure 7e shows that DeepCoFFEA still achieved a TPR above 20% for defended flows with FPR of 10^{-5}, even though it was not trained on any defended traces. Correlations thus appear to remain between the Tor flow and the exit flow for DeepCoFFEA to find. We leave for future work the question of whether the inclusion of flows from different defenses in training data could improve this result.

[3] Client versions in the 0.2.x, 0.3.x and 0.4.x series were used for the 14-month-old set, the three-year-old set, and both the two-month-newer and training sets, respectively.

[4] We used `normal_rcv` as the distribution parameter (bandwidth overhead: 27.54%).

[5] We used the default setting of FRONT with the padding budget as $N_s = 1700$ (proxy side) and $N_c = 1700$ (client side), and the padding window with $W_{min} = 1$ and $W_{max} = 14$ (bandwidth overhead: 33.26%).



(a) ROC

(b) BDR against TPR

(c) ROC using 5,000 and 10,000 testing flow pairs.

Fig. 8. Comparison of state-of-the-art and DeepCoFFEA attacks (Note that x-axes except Figure 8b are in log scale, CTA: Compressive Traffic Analysis, DC: DeepCorr, m-DC: multi-DeepCorr, DCF: DeepCoFFEA (loss ≈ 0.0018), and RG: Random guessing).

C. Comparison to the State-of-the-Art

In this section, we compare the performance of DeepCoFFEA to several previous flow correlation algorithms, including Cosine similarity, RAPTOR, CTA, DeepCorr, and m-DeepCorr, a multi-stage variant of DeepCorr. This variant of DeepCorr was suggested, but not evaluated, by Nasr, Bahramali and Houmansadr [25] as a way of applying DeepCorr in stages to decrease both time complexity and false positives. In this variant, the attacker trains both a lower-dimensional version of DeepCorr that uses only the first p packets to find correlated flows and a full network that uses longer flows of length $p+l$. When searching for correlated pairs, all pairs are evaluated using the less-expensive network; only pairs that are considered to be correlated by this network are then evaluated by the full network, and only flows flagged as matches by both networks are considered to be correlated.

Tuning DeepCorr and m-DeepCorr. For a fair comparison, we tuned DeepCorr and m-DeepCorr to obtain the best performance on the DCF set. First, for DeepCorr, we found the best feature dimension (number of packets) to use and the best number of training flow pairs. The dimension is important, because the attack requires both Tor and Exit flow feature vectors to have the same length. Using longer vectors will induce padding that decreases accuracy, while using shorter vectors might truncate useful information. As detailed in Appendix E, we empirically chose 700 packets as the flow length and 5,000 flow pairs as the training set size.

To determine the combination of input lengths that had the best performance for m-DeepCorr, we used multiple DeepCorr models trained using p packets (namely DC_p), and explored a variety of multi-stage settings to choose the best configuration to maximize performance and minimize time complexity. We found that a 2-stage attack using DC_{100} as the first stage and DC_{700} as the second stage yielded better performance than other settings. The full details and results of this tuning process appear in Appendix F.

Performance Comparison. The results of our comparison are shown in Figures 8a and 8b, in which we also evaluate CTA, RAPTOR, and Cosine similarity. After hyperparameter tuning, we adopted 2,200 and 2,000 packets as the effective flow lengths for RAPTOR and cosine similarity, respectively.

TABLE IV
SPACE AND TIME COMPLEXITIES OF DEEPCORR (DC), m-DEEPCORR (M-DC) AND DEEPCOFFEA (DCF) FOR VARYING NUMBER OF TESTING FLOW PAIRS. WE REPORT BOTH MAIN (MM) AND GPU (GM) MEMORY CONSUMPTION (I.E., MM(GM)) IN SPACE COMPLEXITY.

	Time (seconds)			Space (gigabytes)		
	2,094	5,000	10,000	2,094	5,000	10,000
DC	21,128	118,532	478,667	43(15)	43(15)	43(15)
m-DC	8,041	43,641	174,067	43(15)	43(15)	43(15)
DCF	435	663	1,496	3(7)	5(7)	6(7)

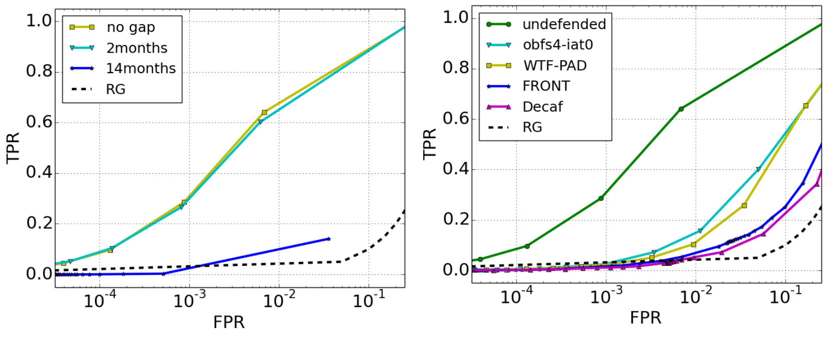
TABLE V
SPACE AND TIME COMPLEXITIES OF TRAINING DEEPCORR AND DEEPCOFFEA (LOSS≈0.004). WE REPORT BOTH MAIN (MM) AND GPU(GM) MEMORY (I.E., MM(GM)) IN SPACE COMPLEXITY.

	Time (days)	Space (gigabytes)
DC	2.5	134(16)
DCF	3	133(6)

DeepCoFFEA outperformed all other attacks when correlating 2,094 flow pairs, reaching a much higher TPR for any given FPR. This substantial improvement correspondingly led to a higher BDR as shown in Figure 8b. CTA failed to detect the correlation between Tor and exit flows effectively, suggesting that the transformations induced by the Tor network are more extreme than the perturbations considered by Nasr, Houmansadr, and Mazumdar [29]. Both DeepCorr and m-DeepCorr performed considerably worse for FPRs closer to 0. In contrast, DeepCoFFEA detected more than half of associated pairs correctly. DeepCoFFEA outperformed m-DeepCorr by significant margins, e.g., 89% vs 13% TPR at 10^{-4} FPR.

Even when increasing the testing dataset size up to 10,000 (non-training) flow pairs, DeepCoFFEA detected the correlated flows more effectively than both DeepCorr attacks with 85% vs 7.6% TPR at 10^{-4} FPR as shown in Figure 8c.

Space and Time Complexity. We compare the runtime of DeepCorr, m-DeepCorr and DeepCoFFEA in Table IV for varying testing set sizes (t_n). We computed the total time to complete the full $t_n \times t_n$ flow attack, including data loading and the correlation metric computation. For DeepCoFFEA, we computed the total time for loading testing flows, generating feature embeddings, computing cosine similarity scores, and aggregating the resulting votes across 11 windows. For m-DeepCorr, we summed the time elapsed for the first stage



(a) ROC for old and future testing sets

(b) ROC for various defenses

Fig. 9. DeepCorr performance against various testing sets and defenses (Note that x-axes are in log scale and RG: Random guessing).

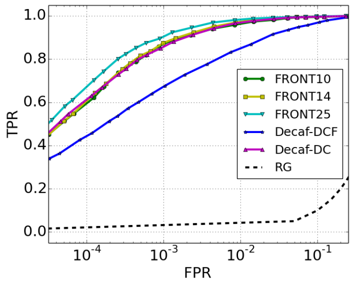


Fig. 10. DeepCoFFEA performance against FRONTn by varying the padding window length ($n = W_{max}$) and the Decaf defense (Note that x-axis is in log scale, RG: Random Guessing, and DeepCoFFEA loss ≈ 0.006).

with DC_{100} and the second stage with DC_{700}. In addition, we measured the main and GPU memory consumption while testing all three attacks.

Compared to DeepCorr, DeepCoFFEA had much lower time and memory requirements. Even though the multi-stage setting reduced the overhead of DeepCorr for longer flows, for the correlation analysis using 10,000 testing flow pairs, DeepCoFFEA still performed faster than m-DeepCorr by *two orders of magnitude* (116:1) while using less memory. We also note that the cost gap between m-DeepCorr and DeepCoFFEA further increased as the number of flow pairs increased. The discrepancy between both time and space costs of DeepCoFFEA and DeepCorr clearly showed the benefit of needing to apply FENs only once per flow window versus applying DC_{100} to every pair of flows. More specifically, we only needed to evaluate the Tor FEN $11 \times t_n$ times and the Exit FEN $11 \times t_n$ times to generate all feature embeddings, rather than evaluating t_n^2 instances of the DeepCorr CNN.

The combination of feature embedding and amplification helps improve the state-of-the-art performance while reducing the complexity significantly, to the point that a DeepCoFFEA-based end-to-end correlation attack may be feasible to deploy at Tor scale.

Time Gap between Training and Testing Sets. Nasr, Bahramali and Houmansadr [25] investigated DeepCorr using a testing set collected three months later than the training set. To see the effects of further separation, we evaluated DeepCorr using DCF testing sets separated by 14 months. Figure 9a shows that DeepCorr performance degraded significantly with the 14-month-old testing set, while it performed comparably on the 2-month-newer set. Based on the observation that DeepCoFFEA effectively detected the correlated flows even after a 14-month gap, this result clearly indicates that DeepCoFFEA requires *much less frequent re-training* than DeepCorr, offsetting the slightly greater training time complexity for DeepCoFFEA compared to DeepCorr (Table III and V).

Robustness against Defenses. We further evaluated DeepCorr against the obfs4, WTF-PAD, and FRONT defenses. Figure 9b shows that DeepCorr significantly degraded against all defenses with 2.7% (obfs4), 2.5% (WTF-PAD), and 1.8% (FRONT) TPRs at 10^{-3} FPR, whereas DeepCoFFEA detects more than 80% of correlated pairs at the same FPR. This result indicates the importance of amplification in DeepCoFFEA since many positive predictions on unmatched flow windows failed to get enough votes to become FPs. The relative effectiveness of FRONT against both attacks suggests that WF defenses may be effective against flow correlation, if the defense can also bypass the amplification effect of DeepCoFFEA.

D. Summary of Contributions

We summarize the key findings as follows:

- Compared to state-of-the-art attacks, DeepCoFFEA achieved a lower computational cost (Table IV) since FENs are used to extract only $O(n)$ feature embedding vectors. Even though m-DeepCorr improved the complexity of vanilla DeepCorr, DeepCoFFEA is more efficient than m-DeepCorr by *two orders of magnitude*.
- DeepCoFFEA significantly lowers the number of FPs (Figure 8a and 8b) due to the use of amplification. This leads it to have substantial performance improvements over DeepCorr and m-DeepCorr, e.g., by 80% in TPR (i.e., 93% vs 13%) at $2 \cdot 10^{-4}$ FPR.
- DeepCoFFEA is *transferable* across circuits, sites, and time scales of up to 14 months (Figure 7c), suggesting that annual retraining would suffice to maintain performance. In contrast, DeepCorr was considerably worse with a 14-month gap between training and testing (Figure 9a).
- DeepCoFFEA correctly detected many of the protected flows by the obfs4 PT and WF defenses with 95% (obfs4), 90% (WTF-PAD), and 84% (FRONT) for 10^{-3} FPR (Figure 7d). This is in contrast to DeepCorr degrading significantly against all defenses (Figure 9b). Furthermore, DeepCoFFEA can conduct flow correlation analysis for both undefended and defended traces without training separate models (Figure 7e).

E. Countermeasures

In this section, we discuss potential countermeasures to thwart DeepCoFFEA-style attacks. DeepCoFFEA learns the difference between correlated and uncorrelated flows rather than directly learning correlated features based on predefined labels. This enables DeepCoFFEA to better understand the distinction between obfuscated, correlated and uncorrelated flows, even if the Tor flows are further perturbed. Figure 7d demonstrates that FRONT was able to hinder DeepCoFFEA to some degree, with a 15% TPR drop at 10^{-3} FPR, The amount of the drop remains insufficient, however, such that further ideas for defenses are needed.

Given the performance gap between DeepCoFFEA and DeepCorr against all the defenses tested, we speculate that the key property that makes DeepCoFFEA difficult for defenses is its use of amplification, which can filter out incorrect correlations from a subset of windows. Random noise generally does not create enough false correlations across enough windows among mismatched flow pairs to confuse the attacker. Thus, we explored further settings in FRONT to undermine DeepCoFFEA more effectively. First, we increased the padding window with $W_{max} = 25$ to increase the chances to pad the entire 25-second flow. Second, we decreased the padding window with $W_{max} = 10$ to increase the chances to inject dummy packets in earlier windows. We also raised the padding data budget for both of these tests ($N_s, N_c = 2,500$) compared to Figure 7d. As shown in Figure 10, the number of true positives decreased when the obfuscation was forced into the first several windows. This indicates that window-level obfuscation could be more effective to weaken the amplification ability.

Thus, we further explored a new defense strategy called *Decaf* to more effectively disrupt the window pattern. For each Tor flow, t_i, we randomly selected a *peer* flow from the DCF set. After dividing the peer flow into ω-second non-overlapping windows, we randomly picked v windows among the total k windows. Note that $\frac{v}{k}$ is a tunable parameter. Then, we extracted the timestamps, T_{pad}, in the chosen v windows and for each t_i, we injected dummy packets at T_{pad}. In this way, we could force the defender to ruin the window pattern, while making each Tor flow less distinct by adding timing information from peer flows. Figure 10 shows the result of applying this approach (Decaf-DCF) with $\omega = 5$ seconds, $\frac{v}{k} = 0.5$. The result had a bandwidth overhead of 49.6% and performed much more effectively than any setting of FRONT. This suggests that a defense specifically focused on perturbing a significant fraction of windows could overcome the amplification feature of DeepCoFFEA.

We made two assumptions in this evaluation. First, we assume that the defender has access to the DCF set, which makes Decaf less reliable. Thus, we conducted an additional evaluation in which the defender picked peer flows from the DeepCorr set ($\omega = 5$ and $\frac{v}{k} = 0.5$). In this way, we can ensure no exposure on the original flows. As shown in Figure 10, this new setting (Decaf-DC) achieved comparable performance to FRONT14 with 4% lower bandwidth cost.

Second, we assume that the defender knows the approximate window duration used by the attacker. While the attacker is not likely to disclose the parameters of the attack, our exploration of window length found that a relatively small range of durations were useful. Very short windows did not have enough information to learn good embeddings, and longer windows resulted in fewer windows overall, reducing the opportunity for amplification. We leave further investigation of the best method of regularizing windows and the consequences of mismatched window durations for future work.

VII. Conclusion & Future Work

End-to-end correlation can break the unlinkability property of an anonymity system, enabling an attacker to match users with the servers they connect with. In this work, we illustrated the practicality of such an attack by introducing DeepCoFFEA, which is much more scalable and practically effective than state-of-the-art attacks. DeepCoFFEA adapts the triplet network architecture as a feature extractor to enable full pairwise comparisons at a cost that is linear rather than quadratic with the number of flows. Further, by splitting flows into a small number of windows and extracting features for each window, DeepCoFFEA creates multiple semi-independent correlation tests that can be combined to amplify differences between matched pairs of flows and unmatched pairs and thereby lower the false positive rate.

By evaluating DeepCoFFEA in various experimental settings, we demonstrated that this new architecture and attack paradigm greatly improves state-of-the-art flow correlation attacks while reducing time complexity by two orders of magnitude.

Our work suggests several important directions for future research. For example, it is possible that more realistic Tor traffic is somehow different from our datasets, and realistic flows lead to less correlated features. Another possible question is whether more sophisticated DNN architectures (e.g., Var-CNN [8]) can be used to decrease the size of the training set while yielding comparable performance to DeepCoFFEA. Further investigation of the DeepCoFFEA architecture for stepping-stone detection and correlation of VPN or HTTPS proxy services might yield interesting results as well. However, the most important next step is to devise a defense that can be effectively deployed against DNN-based traffic analysis attacks, since our empirical study demonstrates the weakness of existing defenses.

Acknowledgments.

We thank our anonymous reviewers for helpful suggestions and comments regarding the presentation and evaluation of DeepCoFFEA. We also thank Milad Nasr for sharing the code of Compressive Traffic Analysis and the DeepCorr set, and discussion about the data collection. We extend our appreciation to Erik Lindeman for the help with building the data collection method. This work was funded by the National Science Foundation under Grants nos. 1816851, 1433736, and 1815757, and the Ewha Womans University Research Grant of 2022.

References

[1] A. Mani, T. Wilson-Brown, R. Jansen, A. Johnson, and M. Sherr, "Understanding tor usage with privacy-preserving measurement," in *ACM Internet Measurement Conference (IMC)*, 2018, pp. 175–187.

[2] T. Wang and I. Goldberg, "Improved website fingerprinting on tor," in *ACM workshop on Workshop on privacy in the electronic society (WPES)*, 2013, pp. 201–212.

[3] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website fingerprinting at internet scale." in *Network & Distributed System Security Symposium (NDSS)*, 2016.

[4] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *USENIX Security Symposium*, 2016, pp. 1187–1203.

[5] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep Fingerprinting: Undermining website fingerprinting defenses with deep learning," in *ACM Conference on Computer and Communications Security (CCS)*, 2018, pp. 1928–1943.

[6] S. E. Oh, S. Sunkam, and N. Hopper, "p1-FP: Extraction, classification, and prediction of website fingerprints with deep learning," *Proceedings on Privacy Enhancing Technologies (PETS)*, vol. 2019, no. 3, pp. 191–209, 2019.

[7] M. S. Rahman, P. Sirinam, N. Mathews, K. G. Gangadhara, and M. Wright, "Tik-Tok: The utility of packet timing in website fingerprinting attacks," *Proceedings on Privacy Enhancing Technologies (PETS)*, vol. 2020, no. 3, pp. 5–24, 2020.

[8] S. Bhat, D. Lu, A. Kwon, and S. Devadas, "Var-CNN: A data-efficient website fingerprinting attack based on deep learning," *Proceedings on Privacy Enhancing Technologies (PETS)*, vol. 2019, no. 4, pp. 292–310, 2019.

[9] S. E. Oh, N. Mathews, M. S. Rahman, M. Wright, and N. Hopper, "GANDaLF: GAN for data-limited fingerprinting." *Proceedings on Privacy Enhancing Technologies (PETS)*, vol. 2021, no. 2, pp. 305–322, 2021.

[10] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr, "Towards an Analysis of Onion Routing Security," in *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, H. Federrath, Ed. Springer-Verlag, LNCS 2009, July 2000, pp. 96–114.

[11] P. Boucher, A. Shostack, and I. Goldberg, "Freedom Systems 2.0 Architecture," Zero Knowledge Systems, Inc., White Paper, December 2000.

[12] P. Syverson, R. Dingledine, and N. Mathewson, "Tor: The second generation onion router," in *USENIX Security Symposium*, 2004.

[13] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, "Users get routed: Traffic correlation on tor by realistic adversaries," in *ACM Conference on Computer and Communications Security (CCS)*, 2013, pp. 337–348.

[14] R. Nithyanand, O. Starov, A. Zair, P. Gill, and M. Schapira, "Measuring and mitigating as-level adversaries against tor," *arXiv preprint arXiv:1505.05173*, 2015.

[15] M. Edman and P. Syverson, "As-awareness in tor path selection," in *ACM Conference on Computer and Communications Security (CCS)*, 2009, pp. 380–389.

[16] A. Houmansadr, N. Kiyavash, and N. Borisov, "Non-blind watermarking of network flows," *IEEE/ACM Transactions on Networking*, vol. 22, no. 4, pp. 1232–1244, 2013.

[17] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal, "RAPTOR: Routing attacks on privacy in tor," in *USENIX Security Symposium*, 2015, pp. 271–286.

[18] M. Akhoondi, C. Yu, and H. V. Madhyastha, "Lastor: A low-latency as-aware tor client," in *IEEE Symposium on Security and Privacy (S&P)*, 2012, pp. 476–490.

[19] N. Feamster and R. Dingledine, "Location diversity in anonymity networks," in *ACM workshop on Privacy in the electronic society (WPES)*, 2004, pp. 66–76.

[20] J. Juen, A. Johnson, A. Das, N. Borisov, and M. Caesar, "Defending tor from network adversaries: A case study of network path prediction," *Proceedings on Privacy Enhancing Technologies (PETS)*, vol. 2015, no. 2, pp. 171–187, 2015.

[21] H. Tan, M. Sherr, and W. Zhou, "Data-plane defenses against routing attacks on tor," *Proceedings on Privacy Enhancing Technologies (PETS)*, vol. 2016, no. 4, pp. 276–293, 2016.

[22] G. Wan, A. Johnson, R. Wails, S. Wagh, and P. Mittal, "Guard placement attacks on path selection algorithms for tor," *Proceedings on Privacy Enhancing Technologies (PETS)*, vol. 2019, no. 4, pp. 272–291, 2019.

[23] A. Arnbak and S. Goldberg, "Loopholes for circumventing the constitution: Warrantless bulk surveillance on americans by collecting network traffic abroad," 2014.

[24] A. Barton and M. Wright, "Denasa: Destination-naive as-awareness in anonymous communications," *Proceedings on Privacy Enhancing Technologies (PETS)*, vol. 2016, no. 4, pp. 356–372, 2016.

[25] M. Nasr, A. Bahramali, and A. Houmansadr, "Deepcorr: strong flow correlation attacks on tor using deep learning," in *ACM Conference on Computer and Communications Security (CCS)*, 2018, pp. 1962–1976.

[26] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 815–823.

[27] J. Kleinberg and E. Tardos, *Algorithm design*. Pearson Education India, 2006.

[28] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright, "Triplet Fingerprinting: More practical and portable website fingerprinting with n-shot learning," in *ACM Conference on Computer and Communications Security (CCS)*, 2019, pp. 1131–1148.

[29] M. Nasr, A. Houmansadr, and A. Mazumdar, "Compressive traffic analysis: A new paradigm for scalable traffic analysis," in *ACM Conference on Computer and Communications Security (CCS)*, 2017, pp. 2053–2069.

[30] K. Krishna and M. N. Murty, "Genetic k-means algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 29, no. 3, pp. 433–439, 1999.

[31] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2002, pp. 849–856.

[32] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, "Toward an efficient website fingerprinting defense," in *European Symposium on Research in Computer Security (ESORICS)*. Springer, 2016, pp. 27–46.

[33] J. Gong and T. Wang, "Zero-delay lightweight defenses against website fingerprinting," in *USENIX Security Symposium*, 2020, pp. 717–734.

[34] *FRONT Implementation*. [Online]. Available: https://github.com/websitefingerprinting/WebsiteFingerprinting/

[35] *WTF-PAD Implementation*. [Online]. Available: https://github.com/wtfpad/wtfpad/

[36] *Keras: The Python Deep Learning library*. [Online]. Available: https://keras.io/

[37] *Tensorflow*. [Online]. Available: https://www.tensorflow.org/

[38] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt, "A critical evaluation of website fingerprinting attacks," in *ACM Conference on Computer and Communications Security (CCS)*, 2014, pp. 263–274.

[39] "Tor project," https://www.torproject.org/.

[40] T. Wang and I. Goldberg, "Walkie-Talkie: An efficient defense against passive website fingerprinting attacks," in *USENIX Security Symposium*, 2017, pp. 1375–1390.

[41] J.-F. Raymond, "Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems," in *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, H. Federrath, Ed. Springer-Verlag, LNCS 2009, July 2000, pp. 10–29.

[42] X. Wang, D. S. Reeves, and S. F. Wu, "Inter-packet delay based correlation for tracing encrypted connections through stepping stones," in *Proceedings of ESORICS 2002*, October 2002, pp. 244–263.

[43] B. N. Levine, M. K. Reiter, C. Wang, and M. K. Wright, "Timing attacks in low-latency mix-based systems," in *Proceedings of Financial Cryptography (FC '04)*, A. Juels, Ed. Springer-Verlag, LNCS 3110, February 2004, pp. 251–265.

[44] Y. Zhu, X. Fu, B. Graham, R. Bettati, and W. Zhao, "On flow correlation attacks and countermeasures in mix networks," in *Proceedings of Privacy Enhancing Technologies workshop*, ser. LNCS, vol. 3424, May 2004, pp. 207–225.

[45] V. Shmatikov and M.-H. Wang, "Timing analysis in low-latency mix networks: Attacks and defenses," in *European Symposium on Research in Computer Security (ESORICS)*. Springer, 2006, pp. 18–33.

[46] X. Wang, S. Chen, and S. Jajodia, "Network flow watermarking attack on low-latency anonymous communication systems," in *IEEE Symposium on Security and Privacy (S&P)*, 2007, pp. 116–130.

[47] A. Houmansadr, N. Kiyavash, and N. Borisov, "Rainbow: A robust and invisible non-blind watermark for network flows." in *Network & Distributed System Security Symposium (NDSS)*, 2009.

[48] S. J. Murdoch and P. Zieliński, "Sampled traffic analysis by internet-exchange-level adversaries," in *International workshop on privacy enhancing technologies*. Springer, 2007, pp. 167–183.

[49] R. Wails, Y. Sun, A. Johnson, M. Chiang, and P. Mittal, "Tempest: Temporal dynamics in anonymity systems," *Proceedings on Privacy Enhancing Technologies (PETS)*, vol. 2018, no. 3, June 2018.

[50] Y. Sun, A. Edmundson, N. Feamster, M. Chiang, and P. Mittal, "Counter-raptor: Safeguarding tor against active routing attacks," in *IEEE Symposium on Security and Privacy (S&P)*, 2017, pp. 977–992.

[51] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 98–113, 1997.

TABLE VI
MEAN TOTAL PACKET COUNT PER WINDOW DURATION.

interval	2	3	4	5
Tor flow	16	31	52	106
Exit flow	24	44	101	244

APPENDIX A
END-TO-END FLOW CORRELATION ATTACKS

End-to-end flow correlation attacks are mentioned in some of the earliest work on low-latency anonymous communications, typically being referred to as Last/First attacks or Packet Counting attacks [10], [11], [41]. Since designs like the Freedom Network and Tor introduce some basic amount of padding that defeats simple packet counting, later works on passive end-to-end attacks used statistical measures of correlation (e.g., normalized distance metrics, Pearson and cosine correlation, empirical mutual information) between flows entering and exiting the network [42]–[45]. A separate line of work has pursued active flow correlation attacks that insert "watermarks" into network flows – by delaying or dropping packets – that can survive the transformations introduced by various network conditions [16], [46], [47].

While Tor does not attempt to defend against *global* passive adversaries, Feamster and Dingledine [19] introduced the idea of AS-level adversaries and showed that such adversaries could potentially observe the entry and exit flows of a significant fraction of the Tor network. Following this work, many researchers investigated how routing dynamics and potential manipulation of the routing infrastructure could position an adversary to observe a larger fraction of traffic flows into and out of the Tor network [13]–[15], [17], [18], [22], [23], [48], [49], and introduced systems intended to reduce the fraction of potentially observed flows [20], [21], [24], [50].

APPENDIX B
CONVOLUTIONAL NEURAL NETWORKS

CNNs [51] are DNN architectures that learn local patterns in input data by employing local filters. More specifically, filters are vectors of weights that are convolved with input feature vectors by "sliding" along the positions of the input vector and computing local dot products to produce feature maps. These feature maps are then "pooled" to reduce their dimensions before further processing is applied to these maps. CNNs are composed of multiple convolutional blocks where each block consists of a convolutional layer, followed by a pooling layer. These blocks are followed by fully connected layers that each compute an output vector by applying an element-wise non-linear activation function to multiple linear combinations of the previous layer.

CNNs have recently been applied as classifiers and feature extractors for website fingerprinting and Tor ingress and egress flow correlation analysis [5], [6], [25]. In this paper, we used the architecture of Deep Fingerprinting (DF) [5] as a starting point for our feature extractors based on the preliminary experimental results presented in Section IV-C.

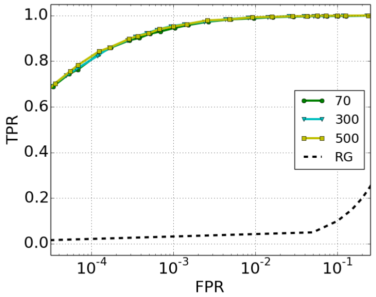


Fig. 11. ROC for different filter applied to the preprocessing when loss ≈ 0.006.

APPENDIX C
WINDOW SETTING INVESTIGATION

Packet Count per Interval. We reported the mean values of the total number of packets in various intervals (seconds) in Table VI. This preliminary study helped determine window intervals in which each window carries enough packets to make correlational traffic analysis possible. We utilized this result to determine the window interval search space as shown in Table II.

Total Flow Duration. We presented the packet counts of 5-second (non-overlapping) consequent subflows for 60 seconds. Table VII shows that the packet count becomes less than 100 after 35 seconds. Based on this observation, we configured the search space for the total flow duration up to 35 seconds in Section V-C and finally chose 25 seconds after hyperparameter tuning.

Window Length. We presented the packet counts of 11 windows and those are counts after padding or truncating flows until the same packet counts remain in each window in Table VIII. Since we used one Tor FEN and one Exit FEN, the input dimensions for all windows have to be equal, in our training, we used 500 packets for Tor traces and 800 packets for exit traces in all windows.

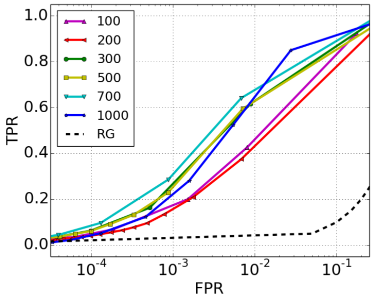


Fig. 12. ROC of DeepCorr (DC) by varying the flow length (i.e., the number of packets) (Note that x-axis is log scale and DC(w) is when evaluating DeepCorr in the window setting).

TABLE VII
THE MEDIAN NUMBER OF PACKETS (PKT) FOR EACH INTERVAL (TOTAL FLOW DURATION: 60SECONDS).

window	$[0, 5)$	$[5, 10)$	$[10, 15)$	$[15, 20)$	$[20, 25)$	$[25, 30)$	$[30, 35)$	$[35, 40)$	$[40, 45)$	$[45, 50)$	$[50, 55)$	$[55, 60)$
Tor	106	458	497	421	290	169	104	65	48	37	27	29
Exit	244	836	797	624	391	202	114	66	48	37	27	26

TABLE VIII
THE MEDIAN NUMBER OF PACKETS (PKT) FOR EACH INTERVAL (TOTAL FLOW DURATION: 25SECONDS AND TOTAL: THE TOTAL NUMBER OF PACKETS OF 25 SECOND FLOW).

window	0	1	2	3	4	5	6	7	8	9	10	total
Tor	106	271	415	481	501	497	479	439	399	347	299	2003
Exit	244	583	797	855	846	797	727	658	586	496	391	3441

TABLE IX
DEEPCOFFEA PERFORMANCE (%) PER WINDOW WHEN $\kappa = 261$.

w	0	1	2	3	4	5	6	7	8	9	10
T	97.2	98.8	97.9	98.2	97.3	96.9	97.4	97.3	97.8	97.6	97.5
F	12.5	12.5	12.5	12.5	12.5	12.5	12.5	12.5	12.5	12.5	12.5

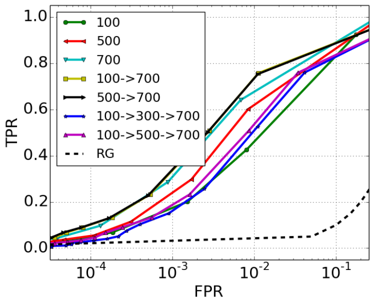


Fig. 13. ROC of m-DeepCorr (m-DC) by varying the multi-staged settings .

APPENDIX D
DEEPCOFFEA IN VARIOUS EXPERIMENTAL SCENARIOS.

Impact of Padding Amount. The connections to different websites should carry a variety of packet counts since the corpus of site sizes is tremendous. The DCF set was collected using all unique 60,084 sites which led to a fair number of flows to be padded or truncated. In this experiment, we investigated the impact of the connections to smaller sites on the DeepCoFFEA performance. We applied different filters when preprocessing the training and testing sets to filter out the connections whose trace packet counts were less than h. As shown in Figure 11, we adopted h from 70, 300, and 500 and the DeepCoFFEA performance was hardly affected by the inclusion of smaller sites, in other words, the increased amount of padding. Thus, we selected 70 for h to evaluate the transferability of the DeepCoFFEA across numerous sites in a more realistic setting by adding more diversity to the site load sizes.

Per-Window Performance. To quantify the amplification capability to reduce the false positive count, we first evaluated DeepCoFFEA within each window and then aggregated the results with voting for comparison. As shown in Table IX, the DeepCoFFEA behavior was consistent across all windows. More interestingly, its performance was significantly benefited by the amplification since FPR decreased from 12.5% to 0.13%. As such, the amplification plays a key role in boosting the performance of DeepCoFFEA.

APPENDIX E
DEEPCORR WITH VARIOUS FEATURE DIMENSIONS

In Section VI-C, using 5,000 training and 2,093 testing flow pairs, we tuned DeepCorr to maximize its performance for a fair comparison. First, we started training the DeepCorr using 5,000 flow pairs as DeepCorr used this scale in most evaluations in the paper, and kept increasing up to 10,000 unique connections. The performance of DeepCorr did not improve with more training data, rather, it became worse using 10,000 pairs. Thus, we chose 5,000 flow pairs as a more effective number of training flows. After that, we explored various packet counts per training flow up to 2,200 packets. According to Figure 12, we decided to use 700 packets as the flow length since the performance became more effective than other flow lengths. Even though it was unable to train DeepCorr using flows containing more than 1,000 packets due to the limited resource, the performance of DeepCorr unlikely improves using more packets since it degraded against 1,000 packet flows.

APPENDIX F
m-DEEPCORR TUNING

In this section, we investigated m-DeepCorr in a variety of multi-stage settings. To gain the upgraded performance along with the reduced correlation complexity, the latter stage of DeepCorr trained using N packet flows should outperform the former stage of DeepCorr trained using M packets when $M < N$. Based on this insight and Figure 12 of Appendix E, we chose 100, 300, 500, and 700 as the flow lengths since they led to the performance improvement compared to shorter dimensions and set 700 as the longest feature dimension since the performance did not improve after 700 packets. The goal of this analysis was to find the multi-stage setting which achieves the effective performance while yielding the acceptable time complexity.

We trained four DeepCorr models using p packets of training data (namely DC_p in which p=100, 300, 500, and 700) and then tested them in various r-stage settings in which $1 \leq r \leq 4$ using testing traces. For example, in one of 2-stage settings, $100 \rightarrow 700$, we first tested DC_{100} using all testing flow pairs and then, tested DC_{700} using the correlated flows determined by DC_{100}. After investigating all possible seven settings [6], we reported our exploration in Figure 13. We excluded the results for 300→700 and 100→300→500→700 because we did not achieve better performance than 1-stage setting (i.e., 700) and 3-stage settings, respectively. Compared to 100 and 500, subsequent attacks such as 100→700 and 500→700 improved the overall performance. However, compared to 1-stage attack (i.e., 700), multi-stage attacks led to almost comparable (i.e., 100→700 and 500→700) or worse performance (i.e., 100→300→700, 100→500→700). Based on this study, we chose 100→700 and used it in Section VI-C as *m*-DeepCorr since it gave us more effective correlation capability than others at less time complexity than 500→700.

[6] There were 700, 100→700, 300→700, 500→700, 100→300→700, 100→500→700, and 100→300→500→700.