IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 17 October 2023, accepted 1 November 2023, date of publication 6 November 2023, date of current version 9 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3330208

RESEARCH ARTICLE

GPU-Accelerated Deep Learning-Based Correlation Attack on Tor Networks

MUHAMMAD ASFAND HAFEEZ[1], YASIR ALI[1], KYUNG HYUN HAN[2], AND SEONG OUN HWANG[2], (Senior Member, IEEE)
[1]Department of IT Convergence Engineering, Gachon University, Seongnam 13120, South Korea
[2]Department of Computer Engineering, Gachon University, Seongnam 13120, South Korea

Corresponding author: Seong Oun Hwang (sohwang@gachon.ac.kr)

This work was supported by the Agency for Defense Development by the Korean Government under Grant UI220040XD.

ABSTRACT The Tor network, renowned for its provision of online privacy and anonymity, faces the constant threat of correlation attacks that aim to compromise user identities. For almost two decades, these correlation attacks were based on statistical methods. However, in recent years, deep learning-based correlation attacks have been introduced to make them more accurate. Nevertheless, in addition to being accurate, fast correlation attacks on Tor are crucial for assessing the real-world viability of such attacks because reduced correlation time aids in estimating its practical implications. Moreover, a reduction in correlation time also helps improve efficiency and ensures practical relevance of the attack. The existing state-of-the-art implementation of a correlation attack on Tor suffers from slow performance and large memory requirements. For instance, training the model required 133 GB of memory, and correlating 10,000 flows takes about 976 seconds. In this paper, we present a novel GPU-based correlation strategy and a fast traffic flow loading technique to reduce time complexity by 7.12× compared to existing methods. Our computational approach, reliant on PyCUDA, facilitates the parallelization of operations used in the attack, thereby enabling efficient execution through the utilization of GPU architecture. Leveraging these two approaches, we introduced an improved correlation attack, which shows high accuracy and fast performance compared to state-of-the-art methods. Moreover, we address resource limitation issues by reducing memory consumption by 47.37% during the training phase, which allows the model to be trained with much fewer resources.

INDEX TERMS Anonymity network, correlation attack, parallel computing, traffic analysis, Tor network.

I. INTRODUCTION

The internet has emerged as a vast, linked network due to the rapid technological advancement in the past decades. However, this advancement has raised the possibility of digital surveillance, which creates privacy concerns in the community. The necessity for dependable privacy-enhancing solutions is becoming more prevalent than in the past as we evolve into a deeply connected society. This is especially important for people living under totalitarian regimes and for whistle-blowers who want to protect their identities. Furthermore, military and commercial entities rely on anonymous networks to provide secure communication across public networks. As a result, the pursuit of anonymous communication protocols has become critical across a wide range of internet applications. To address these issues, several anonymous communication methods have been proposed. Early systems like JAP [1] and Mixminion [2] suffered from latency concerns, which resulted in limited adoption. As a result, newer low-latency systems have replaced them, providing more efficient and secure communication.

In response to these concerns, the Onion router (Tor) network [3] emerged as a potential solution to online privacy and security concerns. It provides the means for users to protect their online identities and communications by encrypting and relaying network traffic through a series of nodes, obscuring the relationship between the sender and receiver [3], [4]. However, this anonymity also creates

The associate editor coordinating the review of this manuscript and approving it for publication was Thomas Canhao Xu.

opportunities for malicious activities, making it a double-edged sword. As a result, it is critical for cybersecurity and law enforcement agencies to understand the vulnerabilities within anonymous communication protocols. Our research focuses on examining the limitations and potential weaknesses of the Tor network, which is one of the most widely used anonymous communication systems.

Despite its robust protection, Tor is not completely impervious to attacks [5], [6]. To identify Tor users, researchers have employed various methods, such as browser-based [7], timing [8], correlation [9], [10], watermarking [11], and fingerprinting attacks [12], [13]. Of all these methods, the correlation attack is considered the most effective [14] because it meticulously scrutinizes the traffic that flows through the Tor network. Through statistical analysis and traffic monitoring, correlation attacks can establish links between incoming and outgoing packets, ultimately revealing the connection between sender and receiver within the Tor network [15], [16], [17].

The Tor network is designed with a specific threat model in mind, which acknowledges the potential risks of end-to-end flow correlation attacks. Yet, more extensive security studies on Tor, such as [16], treat correlation as a primary concern. These studies often only address a fraction of the flows that potential adversaries might observe. They outline techniques to tweak Tor protocols, manipulate internet routing structures, optimize network positioning, and manage resources to either enhance or restrict flow observation. Consequently, the extent to which end-to-end flow correlation attacks pose a genuine threat to the Tor network remains uncertain.

However, applying these attacks to Tor traffic presents a challenge because the traffic between the client and the entry relay differs considerably from the traffic between the exit relay and the destination server. This is because the nature of Tor traffic undergoes significant transformations at different stages of its journey. When the user's data travels from their device to the entry relay, it is encrypted, mixed with other user's data, and broken into fixed-size cells. These transformations create a complex and randomized traffic pattern that is difficult to correlate directly with the original user. On the other hand, when leaving leaves the exit relay heading to the destination server, the data undergo another set of transformations and encryption layers. These multi-stage changes introduce multiplexing, buffering delays, flow control mechanisms, etc., that obscure the relationship between the client and the final destination. As a result, attempting to link entry relay traffic with exit relay traffic becomes intricate and challenging for attackers launching correlation attacks on Tor traffic. For instance, Sun et al. [18] discovered that using Spearman's rank correlation [19] for a small set of entry flows required almost 100MB of traffic per flow for satisfactory performance. To overcome this obstacle, Nasr et al. [9] developed DeepCorr, a Tor-specific correlation method that leverages deep neural networks to precisely classify pairs of flows as correlated or uncorrelated using significantly less traffic.

Another limitation in the process of correlating Tor flows from end-to-end is the pairwise comparison restriction of the attack [20], [21], [22]. Such attacks analyze the correlation between the packet times and sizes of a flow entering and leaving the Tor network in order to determine whether they belong to the same connection. To reveal the identities of multiple flows, attacks must evaluate the correlation between all conceivable incoming and outgoing connections. Consequently, for N Tor connections, the attacks have to conduct N^2 comparisons, raising a scalability issue. This leads to an increase in computational time, especially if the number of flows is huge (tens of thousands) at a given moment, resulting in a poor Bayesian detection rate (BDR). Note that the BDR is a crucial metric for assessing the efficacy of detection systems or algorithms using Bayesian statistics. It is imperative to evaluate the probability of detection mechanisms identifying or detecting specific patterns, which indicates a correlation between the two ends.

To resolve these problems, Oh et al. [10] presented a modified triplet approach to compare entry and exit flows. Although their results were satisfactory, model real-world implementation was minimal. For example, consider a scenario where we need to correlate 10,000 network traffic flows using the DeepCoFFEA method proposed in [10]. Each flow in the network involves crucial steps for analyzing the correlation between packets. However, this process is non-trivial. It requires significant computation, particularly when dealing with hundreds and thousands of packets. Even in a controlled lab setting, where the conditions are ideal, the analysis might take several hours to complete due to the complex calculations required. In real-world applications, the network data is constantly changing and demands almost real-time analysis. Due to the sheer amount of computations involved, the algorithm becomes impractical for such scenarios. Therefore, there is a critical need for efficient and scalable correlation attacks based on deep learning for real-time analysis. This way, the analysis can be done in almost real-time without compromising the accuracy of the results. Additionally, the training space's intricacy poses a challenge as it requires substantial memory allocation. This necessitates the need for advanced computing techniques to perform the analysis efficiently and accurately.

Network flow analysis involves a crucial step of analyzing the correlation between packets. However, this process is not a simple one. It requires a significant amount of computation, particularly when dealing with multiple packets. Even in a controlled lab setting, where the conditions are ideal, the analysis might take several minutes to several hours to complete due to the complex calculations required.

In real-world applications, the network data is constantly changing and demands real-time or near-real-time analysis. However, the time complexity of the algorithm becomes impractical for such scenarios. Therefore, there is a critical need for efficient and scalable correlation attacks based on deep learning for real-time analysis. This way, the

analysis can be done in real-time or near-real-time without compromising the accuracy of the results.

However, the training space's intricacy poses a challenge as it requires a substantial allocation of memory. This necessitates the need for advanced computing infrastructure to perform the analysis efficiently and accurately.

In summary, slow timing performance and high memory consumption are two primary reasons that limit the study of correlation attacks on the Tor network to being carried out only on small datasets. This seriously limits the possibility of conducting such an experiment on a larger dataset to emulate a practical, real-world situation. In this paper, we address the above-mentioned challenges and significantly improve the performance of deep learning-based correlation attacks. The following are the major contributions of this paper:

1) First, we propose a pioneering approach that harnesses the parallel processing capabilities of GPUs to accelerate the correlation attack process. To the best of our knowledge, this is also the first work to explore utilization of a parallel architecture like the GPU in correlation attacks. The proposed technique efficiently computes correlation similarities between flows by parallelizing calculations using GPU processing power. Our proposed method surpasses the previous conventional CPU implementation in [10], delivering a 13.765× improvement in threshold definition and is 7× faster for the voting mechanism.
2) We develop a highly effective method for loading data, which works in tandem with our GPU implementation, leading to a significant decrease in correlation time. A greater length in the flows may improve performance, although it also increases correlation attack time. However, our technique can speed up the processing of flows and improves performance by 10.86× compared to DeepCoFFEA [10].
3) Finally, our research delves into the significant issue of memory consumption to train deep learning models for correlation attacks. Traditional correlation attacks demand huge amounts of random access memory (RAM) [9], [10], thus preventing scalability and hindering large dataset analysis. We propose a technique to significantly reduce memory consumption by the system while training the deep learning model. The proposed technique achieves a noteworthy reduction of 47.37% in RAM consumption for model training in correlation attacks compared to [10]. This breakthrough paves the way for analyzing more extensive datasets and enhances the overall efficiency of the attack process.
4) We have made the source code for our proposed method publicly available at https://github.com/yasirali0/fast _ correlation_attack_on_tor. We hope that this will allow other researchers to easily reproduce our findings and even inspire further studies and research on GPU acceleration for deep learning approaches.

Our proposed attack involves a network-level adversary that closely monitors Tor flows between clients and entry guards, as well as exit flows between exit relays and destination servers. This is achieved by adversaries running their own relays, as demonstrated in Fig. 1. The adversaries then extract packet timing and size information, which are subsequently utilized to train two deep neural network (DNN) models, A and B. Inputs for A and B are Tor flows and exit flows, respectively. It is a crucial property of A and B that if a correlation exists between Tor flow t and exit flow x, then $d(A(t), B(x)) \geq \tau$ for some correlation metric d and threshold τ. On the other hand, if such a correlation is not present, then $d(A(t), B(x)) < \tau$.

The adversary then applies A and B to k consecutive flow windows, resulting in extraction of feature embedding vectors. The pairwise correlation matrix is subsequently computed, and if two flows are deemed correlated by d in at least k-l windows (for a small threshold), the adversary concludes that the flows are indeed correlated. Otherwise, the flows are considered to be uncorrelated. This approach exponentially amplifies the difference between true positives and false positives.

The remainder of this paper is organized as follows: Section II provides background information and an overview of related work in correlation attacks, highlighting the gaps our research addresses. Section III presents the methodology and technical details of our proposed approach. Section IV presents the experimental setup and showcases the results and performance improvements achieved. Finally, Section V concludes the paper, highlighting the significance of our contributions.

II. BACKGROUND

In this section, we discuss the GPU architecture and the PyCUDA programming model. We also discuss previous work related to correlation attacks and PyCUDA.

A. OVERVIEW OF GPU ARCHITECTURE AND PYCUDA PROGRAMMING

A GPU is composed of several streaming multiprocessors (SMs), each of which contains hundreds of CUDA cores. For instance, the RTX3080 model boasts 128 cores. To simplify GPU programming for general-purpose computing, NVIDIA developed the CUDA Software Development Kit. The CUDA programming model organizes numerous threads into blocks, and multiple blocks form a GPU grid, as illustrated in Fig. 2. Each thread and block has its own indexing for parallel computing. NVIDIA GPUs group 32 threads into one warp to enable efficient instruction scheduling and memory access. It is important to note that warp divergence can occur when threads within a warp execute different paths, resulting in a significant performance penalty.

In this paper, we explore PyCUDA, which provides a Pythonic interface to the CUDA driver API. With PyCUDA, we can allocate and manage GPU memory, transfer data between the CPU and GPU, and execute CUDA kernels from

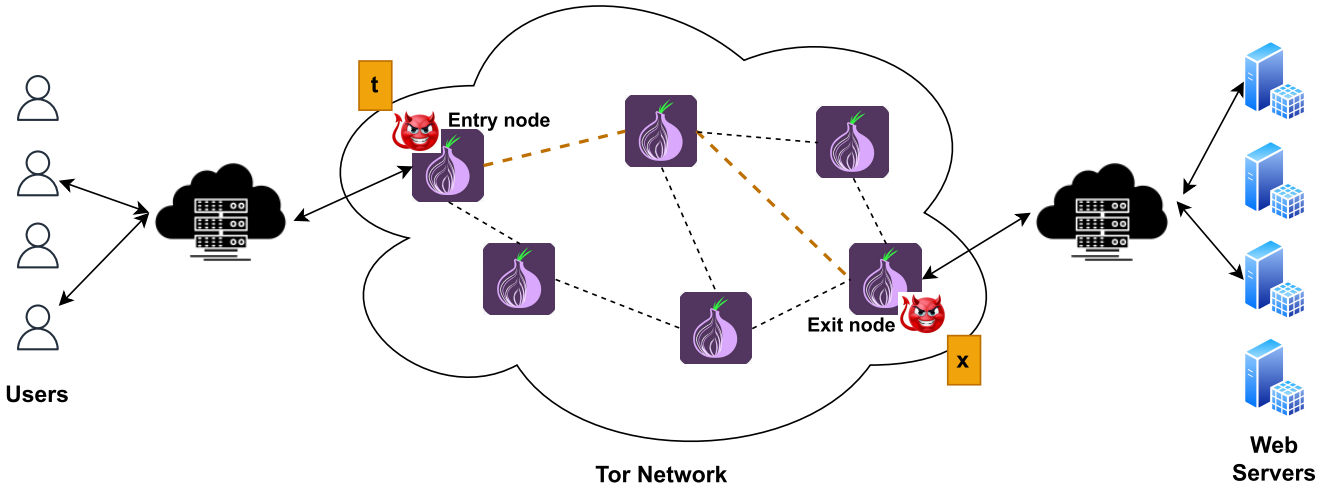

FIGURE 1. Implications of the threat model on Tor networks.

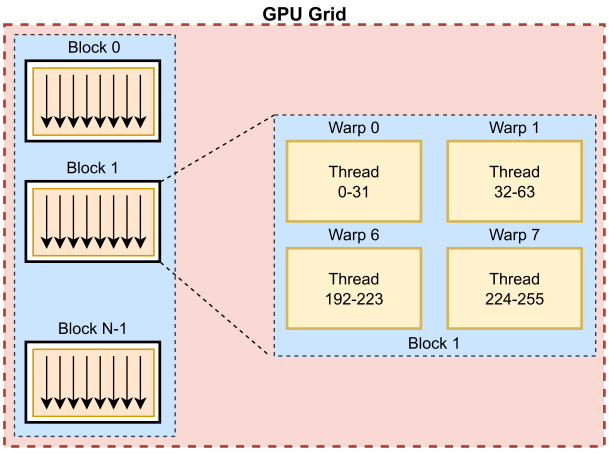

FIGURE 2. The GPU architecture represents the grid, blocks, warps, and threads in CUDA.

Python. Additionally, PyCUDA offers tools for profiling and debugging CUDA applications.

B. RELATED WORK

1) CORRELATION ATTACKS

Previous correlation techniques were used to link network flows, each utilizing various traffic features such as packet timings, packet sizes, and flow rates. However, these methods have their advantages and disadvantages. For example, Chothia and Guha [23] utilized mutual information metrics, which have higher accuracy but require longer feature vectors. On the other hand, Levine et al. [24] explored the Pearson correlation for linearly correlating the packet timing of Tor flows, but this method was limited to shorter flows. Sun et al. [18] focused on passive correlation attacks on Tor and presented the RAPTOR attack using Spearmen's correlation, which monitors both ends of a higher fraction of Tor connections through asymmetric traffic analysis. However, these statistical correlation metrics had solubility issues and required long-flow data samples for acceptable results.

DeepCorr by Nasr et al. [9] employs deep learning to scrutinize Tor flows, yielding consistently accurate outcomes across diverse scenarios compared to previous methods. By using only 100 packets of flow data, it outperformed RAPTOR [18] in terms of true positive rate (TPR) and false positive rate (FPR), although it had a lower BDR. However, DeepCorr has limitations, including computational complexity and the requirement for retraining every three to four weeks to adapt to current Tor network conditions. Oh et al. [10] introduced DeepCoFFEA, which utilizes feature embedding networks and amplification. They employ a triplet loss function to train the model, aiming to perform correlation using cosine similarity. DeepCoFFEA successfully tackles the issue of low BDRs encountered in prior research, delivering 5% higher accuracy than DeepCorr. Nevertheless, despite these advances, the scalability and time complexity of DeepCoFFEA underlined a pressing gap in real-time Tor traffic analysis. Tian et al. [25] developed a perturbation method for correlating traffic. This approach underscores the importance of implementing effective techniques for managing the dynamic landscape of Tor traffic in a timely and efficient manner. Despite notable advancements in correlation techniques, there remain critical research gaps that require attention. It is imperative to develop more efficient, scalable, and data-independent methods that can meet the ever-evolving and real-time demands of Tor traffic analysis.

2) PARALLEL COMPUTATION USING A GPU AND PYCUDA

In the past, various data processing techniques that were commonly used to correlate data fell short in terms of meeting software performance standards. To address this issue, the GPU framework has been utilized to optimize correlation algorithms, resulting in substantial improvements

in performance. For instance, Zhu et al. [26] used the parallel computing power of the GPU to correlate data for radio astronomy. Similarly, Lee [27] explored application of a GPU to accelerate the sequence alignment by Fourier-space cross-correlation. León-Sandoval and Barbosa-Santillán [28] demonstrated the power of PyCUDA to elevate data processing capabilities in machine learning for meticulous analysis of numerical data. Furthermore, Dallas et al. [29] emphasized the efficacy of GPU utilization in enhancing the efficiency in statistical analysis of large data sets.

C. CORRELATION ATTACKS AND DEEPCOFFEA

The goal of a correlation attack on the Tor network is to link a user's entry and exit nodes, thereby de-anonymizing the traffic. The attacker tries to identify patterns in network traffic and timing that could expose the user's identity or activity. A successful attack would compromise a user's privacy by linking their online activities to their real-world identity. The effectiveness of the attack is measured by the ability to consistently and accurately associate a user's entry and exit nodes, thus breaking the anonymity provided by Tor's layered encryption and routing.

Oh et al. [10] DeepCoFFEA is a precise method for conducting correlation attacks on the Tor network with high accuracy. Utilizing deep learning capabilities, Oh et al. [10] successfully improved the accuracy and efficiency of these attacks, potentially facilitating the de-anonymization of Tor users. To address the existing limitations, they suggested the integration of two components, metric learning, and amplification, forming the DeepCoFFEA technique. Metric learning is employed to train a deep neural network that can extract informative features from network traffic data. This feature-extraction process enhances the discriminative power of the attack, allowing a more accurate classification of flows that belong to the same Tor circuit. The authors presented detailed experimental results, demonstrating that DeepCoFFEA achieved significantly higher classification accuracy compared to traditional correlation attacks [10].

In addition, Oh et al. [10] presented a method for enhancing the attack accuracy through an amplification mechanism. This mechanism breaks down the flows into smaller, partially overlapping sub-flows or windows. DeepCoFFEA can evaluate each one separately and aggregate the results using ensemble voting. This amplification mechanism has proven effective in significantly reducing false positives, improving the accuracy of attacks. The correlation attack proposed in DeepCoFFEA is performed in five steps.

1) **Data Loading:** N Tor and exit flows are loaded into memory. This involves retrieving and storing the necessary flow data for further analysis.
2) **Embedding Generation:** The flow data are fed into a CNN that transforms them into a lower-dimensional representation that captures the essential characteristics of the flows.
3) **Cosine Similarity Calculation:** This involves measuring the cosine similarity between each Tor and exit node flow embedding pair to quantify a correlation.
4) **Threshold Definition:** This step sets threshold values to determine the correlation between Tor and exit flows. These threshold values serve as the decision boundaries to evaluate the success of an attack.
5) **Voting Mechanism:** The mechanism determines whether a Tor flow is correlated with an exit flow. This step involves aggregating the results from the similarity calculations using the defined thresholds. The voting-based approach decides on the correlation between each Tor flow and its corresponding exit flow.

Experiments on real-world Tor traffic datasets showed that DeepCoFFEA was highly effective. It outperformed state-of-the-art correlation attacks in terms of both precision and recall. The authors conducted a comprehensive evaluation of the performance under various scenarios, including different network conditions and traffic loads.

III. METHODOLOGY

In this section, we discuss our proposed techniques to address the issues of slow timing performance and high memory consumption in DeepCoFFEA. First, we present our proposed parallelization technique using PyCUDA to reduce threshold and voting time. Then, we discuss our memory offload mechanism and our technique to streamline the memory complexity issue.

A. PARALLELIZATION TECHNIQUES USING PYCUDA

Referring to previous discussions in Section II-C, a correlation attack can be executed in five steps. Table 1 depicts the time required for different steps in DeepCoFFEA for different numbers of flows. The majority of the time is spent on data loading, threshold finding, and by the voting mechanism. Note that threshold definition and the voting mechanism are more time-consuming than data loading, and they are typically hard to scale up. Consider when the number of flows increases from 5,000 to 10,000; data-loading time increases linearly ($\approx 1.99\times$) but the time for threshold finding and using the voting mechanism increases quadratically ($\approx 4.36\times$ and $\approx 4.04\times$). This section describes the proposed parallelization technique to speed up threshold definition and the voting mechanism, which can be implemented through PyCUDA.

1) THE PARALLELIZATION TECHNIQUE FOR THRESHOLDS

The process of threshold finding is a critical component when executing a correlation attack in DeepCoFFEA. However, this process can be quite time-consuming because it involves two distinct steps: sorting lists and performing computations to determine the optimal cut-point. For sorting, implementation of the bubble sort technique on a GPU can lead to a decline in performance due to excessive irregular global memory accesses. This limitation can hamper the efficiency of

TABLE 1. Correlation times (in seconds) for different steps in DeepCoFFEA [10].

Flows	Data Loading	Embedding Generation	Cosine Similarity	Threshold Finding	Voting Mechanism	Total
2094	24.72	11.52	0.217	16.13	19.25	70.07
5000	58.39	19.5	0.927	97.32	104.6	275.48
7500	87.29	26.39	2.401	230.79	222.43	569.43
10,000	116.42	32.71	3.187	423.15	420.07	976.99

Algorithm 1 Determining the Threshold of Flows on a GPU

Input: Input similarity list S_{list}, Threshold similarity list T_{simi}, Tor Entry Flows T_{Et}, Tor Exit Flows T_{Ex}
Output: Single Output list out_s, Multiple Output list out_m

// Total number of threads required
1: $threads\_tot$ = 1000 ▷ For 5K and 10K Flows
2: $threads\_tot$ = 698 ▷ For 2094 Flows
// CPU Part ▷ Sorting the Input Similarity list
3: S_{sorted} =sort.S_{list}
// GPU Part
4: Threshold_GPU(($Thres\_out$), (S_{list}), (S_{sorted}), ($Thres\_value$), block=($threads\_tot$, 1, 1), grid=(1, 1)) ▷ Alg.2

Algorithm 2 Threshold_GPU: Post-Processing of the Threshold to Vote on Tor Flow Similarity

Input: $N \times N$-length input similarity list S_{list} and input similarity sorted list S_{sorted} and Threshold Value T_v
Output: $N \times N$-length single output list $Thres\_out$

1: tid = $thread$ ID
// Launch N threads in parallel
2: num_lines = $num\_flowstot\_threads$
3: **for** i from 0 to num_lines **do**
4: cut_point = (sizeof(S_{list}) − 1)× (T_v100)
5: $Thres\_out$ [$tid\ i \times 1000$] = S_{sorted}[cut_point[tid] $tid \times 5000\ i \times 5000000$]
6: **end for**

GPU-based sorting operations and may require consideration of alternative approaches. To tackle this issue, the CPU handles the first part (list sorting), while the second part, which possesses parallelism used to calculate the cut-point, is executed by the GPU. Algorithm 1 outlines the process of parallel threshold finding.

The algorithm determines the number of threads required for various flows, seen in lines 1 and 2 of Algorithm 1. Once the number of threads is determined, the list is sorted on the CPU in line 3. Finally, the kernel is launched on the GPU in line 4 to execute the cut-point calculation and to determine the threshold of flows based on the threshold value provided. Implementation of this technique allows for an efficient and more accurate finding of thresholds for a correlation attack.

Algorithm 2 (Threshold_GPU) is a post-processing step that assesses the similarities of Tor network flows based on a specified threshold value. The algorithm enhances the efficiency of flow analysis by exploiting GPU parallelism. It operates on input similarity list S_{list} (comprising $N \times N$ entries) and sorted similarity list S_{sorted}. Specified threshold value T_v is provided as input to govern flow similarity determination.

When launching a parallel execution with N threads, each thread is assigned a unique ID (tid). The algorithm calculates the number of lines to be processed concurrently, denoted num_lines. The workload is divided evenly among the threads, allowing for efficient parallel computation. The subsequent loop iterates over a range of indices, starting from 0 up to num_lines. Each iteration of the loop calculates cut_point, which is crucial in identifying a separation point in S_{sorted}. The computation of this cut point is based on threshold value, T_v. It is calculated as a fraction of the size of S_{list} (($sizeof(S_{list}) - 1) \times (T_v 100)$) and establishes a pivotal index within the sorted list that acts as a reference point for subsequent analysis.

In the next step, Algorithm 2 extracts relevant information from S_{sorted} using cut_point. The values are assigned to a specific location on $Thres\_out$ (the output list) based on thread ID, loop index, and offset constants. Each thread contributes to the final output list in a coordinated manner. In essence, Algorithm 2 utilizes parallelism to efficiently compute flow similarity based on the predefined threshold. The algorithm distributes tasks across threads and uses an ingenious indexing strategy to process large volumes of data on the GPU. This technique holds great promise for accelerating flow analysis in Tor networks, advancing network security and performance optimization.

2) PARALLELIZATION TECHNIQUE FOR THE VOTING MECHANISM

Algorithm 3 outlines the parallel voting mechanism. It generates a single list indicating correlated flows, accurately identifying all correlations. Lines 1 and 2 show the number of threads required for different flow sizes, and then line 3 launches the GPU kernel for the voting mechanism to correlate the flows. Finally, the results are stored in out_s, which can be used for final analysis.

Algorithm 4, responsible for determining the output list (denoted out_s), correlates the flows. Line 2 calculates the necessary number of iterations for the loop, and line 4 transfers elements from s_{simi} to t. The appropriate index for the output is then determined in line 5. If S_{simi} is greater than or equal to t, the corresponding line is executed, and the output is stored in out_s according to the calculated index.

Algorithm 3 Voting in Parallel on the GPU

Input: Input similarity list S_{list}, Threshold similarity list T_{simi}
Output: Single Output list out_s

// Total number of threads required
1: $threads\_tot = 1000$ ▷ For 5K and 10K Flows
2: $threads\_tot = 698$ ▷ For 2094 Flows
// GPU Part
3: Voting_GPU((out_s), (T_{simi}), (S_{list}), block=($threads\_tot$, 1, 1), grid=(1, 1)) ▷ Alg.4

Algorithm 4 Voting_GPU: Post-Processing of the Threshold to Vote for Tor Flow Similarity

Input: $N \times N$-length input similarity list S_{list} and N-length similarity threshold T_{simi}
Output: $N \times N$-length single output list out_s

1: tid = *thread* ID
// Launch N threads in parallel
2: $num\_lines = num\_flowstot\_threads$
3: **for** i from 0 to num_lines **do**
4: $t[tid] = S_{simi}[tid \;\; i \times tot\_threads]$
5: $con\_num = i \times num\_lines \;\; tid \times num\_lines \times tot\_threads$
6: **for** j from 0 to num_lines **do**
7: **if** $S_{simi}[i \times tid \;\; j \times tid] \geq t[tid]$ **then**
8: $out_s[con\_num = j \times tid] = out_s[con\_num = j \times tid] \;\; 1$
9: **else**
10: $out_s[con\_num = j \times tid] = out_s[con\_num = j \times tid]$
11: **end if**
12: **end for**
13: **end for**

B. DATA-LOADING MECHANISM

In this section, we first explain why efficient data loading is important, and we then explain our data-loading mechanism in comparison with DeepCoFFEA's [10] data-loading mechanism. In the context of correlation attacks on the Tor network, an efficient data-loading mechanism is crucial to handling the substantial volume of network traffic. The data-loading process plays a pivotal role in preparing flows for subsequent analysis and correlation. Here, we outline our innovative approach to efficiently load Tor and exit flows, further reducing the time complexity of the correlation attack as a whole.

Before explaining our strategy, we briefly explain the storage format and data-loading mechanism of DeepCoFFEA. The storage of Tor and exit flow basically means that the necessary information that defines these flows, such as packet timing, packet size, directions of packets, and so on, are stored in a format suitable to executing the correlation attack. In DeepCoFFEA, the Tor and exit flows are stored as an associative array or dictionary. The dictionary data structure allows the data to be looked up through their associated keys, just like a hash table. The Tor flows have their own keys, and the same is true for exit flows. At data-loading time, DeepCoFFEA redundantly uses the lookup keys to load the corresponding windows of Tor and exit flows.

Although the data-loading mechanism used in DeepCoFFEA is acceptable, there is a basic redundancy in their method. For each window, they need to provide the key, which then points to the stored flow, which is then passed to the deep learning model. This lookup method adds extra delay, which is very important to address. Therefore, we propose using a new data-loading mechanism in which the lookup keys or the pointers for Tor and exit flows are loaded into memory only once, as illustrated in Fig. 3. They are then used in the whole data-loading process to avoid extra lookup time. By employing this strategy, a significant amount of time ($\approx 11\times$) is reduced compared to DeepCoFFEA as shown in Table 3.

C. STREAMLINING MEMORY CONSUMPTION

Here we explain our strategy to reduce memory consumption when training the deep learning model used later for correlating the flows.

The main strategies to reduce memory consumption during the training stage are dynamic memory allocation and garbage collection. Dynamic memory allocation is employed to allocate resources based on the current needs of the training process, and the garbage collection is used at the end of a training iteration to deallocate unnecessary memory, freeing up space for the next training iteration. These adaptive approaches ensure that memory is allocated only when necessary, preventing unnecessary memory consumption and allowing the system to use more data to train a more effective correlation model. The memory consumption of our proposed method is 70GB, which is 47.37% less than the state-of-the-art DeepCoFFEA [10], as shown in Table 2.

While dynamic memory allocation and garbage collection are indeed well-established in mainstream training frameworks, we would like to emphasize a crucial distinction in our approach. In typical training frameworks, garbage collection usually operates when the Python kernel stops execution or in this case, when the whole training process is completed. However, we take a proactive stance by analyzing the training process, identifying memory-consuming variables, deallocating their memory promptly by deleting references as soon as they are no longer needed and invoking garbage collection to free up that memory. This proactive management ensures that memory is freed up in a more granular and immediate fashion, contributing to the efficient use of resources throughout the training iterations. This meticulous analysis and manual invocation of garbage collection allow the memory space to be freed up promptly, optimizing memory consumption in real-time during the training process. This approach enhances the scalability and practicality of our correlation attack model, where large memory consumption can present obstacles for researchers and lead to scalability issues.

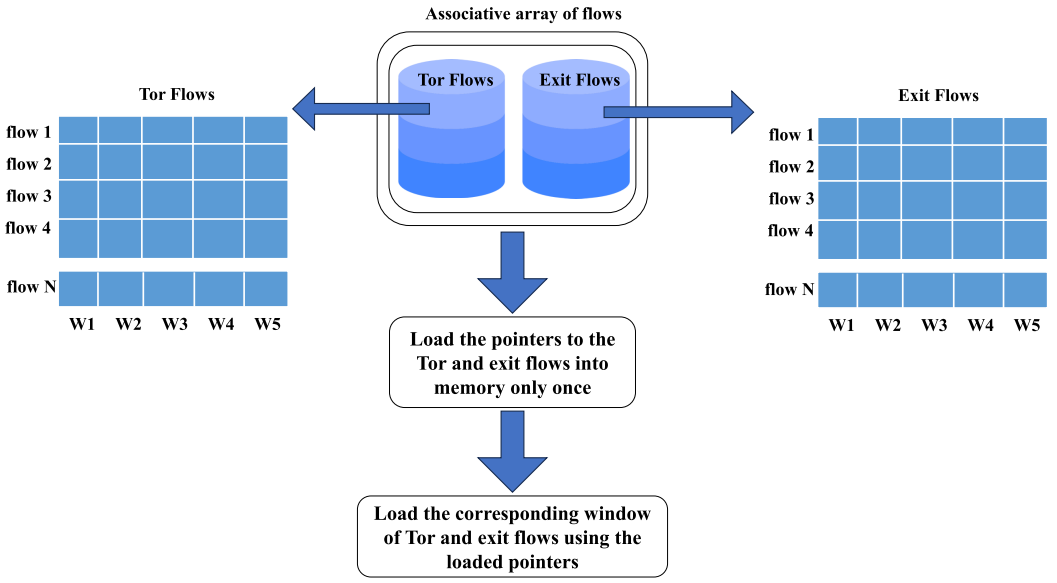

FIGURE 3. Data-loading mechanism. W represents the windows of flow.

TABLE 2. Training space complexity comparison.

Models	Space (gigabytes)
DC [9]	134(16)
DCF [10]	133(6)
Proposed	70(6)

In summary, while dynamic memory allocation and garbage collection are fundamental techniques, introducing a proactive and targeted approach to them allows the reduction of memory consumption in a more immediate and resource-efficient manner than the traditional frameworks.

IV. RESULTS

In this section, we delve into research findings and experimental results. Experiments were carried out on a machine running Ubuntu operating system equipped with a 3.20GHz Intel Xeon Silver 4208 processor with 128GB of RAM. The performance was evaluated using an NVIDIA Ampere A10 GPU. The training and testing datasets used in the computational experiments were obtained from DeepCoFFEA [10] and can be accessed at https://github.com/traffic-analysis/deepcoffea. The training dataset comprises 11 windows-separated Tor and exit traffic flows, with inter-packet delays and packet sizes being the two main features. For computational testing purposes, the 10k testing data was also utilized, which was collected by DeepCoFFEA [10].

A. CORRELATION TIME

To improve the correlation attack time, we worked on three steps: data loading, threshold finding, and voting time. To demonstrate the advantages of our proposed methods, we conducted experiments with varying numbers of flows and compared the performance of the two methods. Table 3 and Fig. 4 depicts the improvement in correlation time from using our proposed methods compared to the DeepCoFFEA [10] (DCF) method. Our proposed method showed significant improvements. For example, when loading data for 2094 flows, the DCF method took 24.62 seconds, whereas our method only took 2.24 seconds (10.99× faster). As the number of flows increased, the advantages of our method became more apparent. For 10,000 flows, our method outperformed the DCF, taking only 10.71 seconds compared to 116.32 seconds.

The experimental evaluation of the proposed method has demonstrated its superior performance in both threshold finding and voting time when compared to the DCF method. When processing 2094 flows, it took merely 1.50 seconds to complete the threshold process and only 2.42 seconds for voting (improvements of 10.66× and 7.18×). In contrast, the DCF method required significant time, taking 15.99 and 17.38 seconds for threshold finding and voting, respectively.

Furthermore, the proposed method continues to exhibit its dominance even when tested with a larger number of flows, namely 10,000. In this case, it delivered 13.77× and 7.0× faster results for threshold finding and voting time, respectively, completing the threshold finding process in only 31.03 seconds and the voting process in 56.63 seconds. These results demonstrate that the proposed method is faster and more efficient than the DCF method.

B. COMPARISON

The efficiency of the DCF method and the proposed method are comprehensively compared in Table 4, taking into account all the time components involved in the process. For a fair comparison, we ran DCF method on our system to record the time for all components involved in the

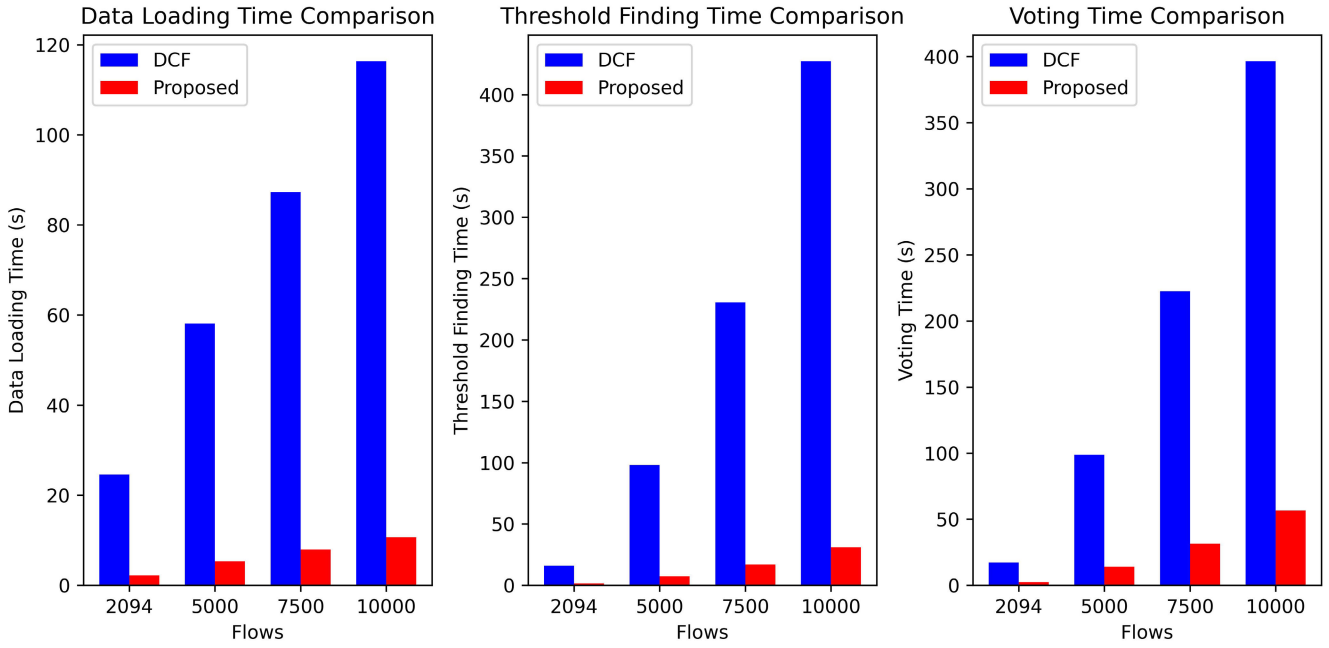

FIGURE 4. Comparision of correlation times for data loading, threshold finding, and voting mechanism with DCF [10] for different flows.

TABLE 3. Correlation times (in seconds) for data loading, threshold finding, and voting mechanism.

Flows	Method	Data Loading	Threshold Finding	Voting Time
2094	DCF	24.62	15.99	17.38
	Proposed	2.24	1.50	2.42
5000	DCF	58.10	98.17	98.72
	Proposed	5.34	7.49	14.14
7500	DCF	87.29	230.79	222.43
	Proposed	7.92	17.03	31.44
10000	DCF	116.32	427.14	396.43
	Proposed	10.71	31.03	56.63

process. The code is available from the GitHub repository https://github.com/traffic-analysis/deepcoffea. The table clearly indicates that the proposed method consistently outperformed the DCF method in terms of total time, regardless of the flow quantity. The proposed method achieved these time savings by significantly reducing data loading, threshold finding, and voting time. Although there may be minor increases in embedding time and cosine similarity calculations, these differences are far outweighed by substantial improvements in other areas.

Our proposed method yielded significant improvements in correlation time. In particular, we observed 3.83× faster processing for 2094 flows, with a processing time of only 18.29 seconds compared to 70.07 seconds under the DCF method. Consequently, our method facilitates nearly real-time performance. This attribute renders our method highly advantageous in time-sensitive scenarios where swift analysis of user traffic patterns and identification of potential vulnerabilities are of paramount importance.

Furthermore, our method demonstrated processing speeds 5.41× and 6.34× faster for 5000 and 7500 flows, respectively. Notably, our proposed method achieved an impressive 7.12× faster correlation time for 10000 flows (correlating the flows in just 137.19 seconds), whereas DCF took 976.99 seconds. We observed that performance gain was more significant if the number of flows increased. This shows that our work can be suitable for analyzing a large-scale correlation attack with a large number of flows, which was not possible for DCF. This advancement in efficiency and scalability highlights the practical utility of our GPU-accelerated deep learning-based approach in real-world scenarios.

In addition to correlation time, reliability is another important aspect of correlation attack. Reliability is measured by TPR and FPR as defined in the DCF framework. The performance of the correlation attack depends on the deep learning model used to correlate Tor and exit flows. Currently, DCF has achieved the best performance with 93% TPR at 10^{-4}% FPR. Our methodology employs the same deep learning model used by DCF, while significantly reducing the correlation time while maintaining the same level of TPR and FPR.

C. DISCUSSION AND LIMITATION

Significant progress has been achieved in the realm of GPU-based correlation attacks. Our approach has significantly improved performance and simultaneously decreased resource consumption. By leveraging the parallel processing abilities of GPUs, we have expedited the correlation attack process. This is discernible from noteworthy enhancements

TABLE 4. Comparison of the proposed correlation attack with DCF [10] for different flows.

Flows	Methods	Data Loading	Embedding Time	Cosine Similarity	Threshold Finding Time	Voting Time	Total Time	Speed up
2094	DCF	24.62	11.90	0.181	15.99	17.38	70.07	3.83
	Proposed	2.24	12.03	0.099	1.50	2.42	18.29	
5000	DCF	58.10	19.40	1.085	98.17	98.72	275.48	5.41
	Proposed	5.34	22.86	1.097	7.49	14.14	50.93	
7500	DCF	87.29	26.39	2.401	230.79	222.43	569.43	6.34
	Proposed	7.92	31.15	2.289	17.03	31.44	89.83	
10000	DCF	116.32	33.28	3.823	427.14	396.43	976.99	7.12
	Proposed	10.71	35.02	3.797	31.03	56.63	137.19	

compared to traditional CPU implementations, as demonstrated in the research presented in DeepCoFFEA. However, our work only examines up to 10,000 flows, but it remains unclear if it can efficiently manage more extensive datasets or complex correlation tasks on a scale of ten or even a hundred times larger. To handle such large datasets, more aggressive techniques have to be developed to reduce memory consumption, training time, and correlation complexity. This is an interesting research direction that we would like to pursue in the near future.

To ensure the reproducibility of our study, it is essential to conduct various environmental tests and allow other researchers to test our methodology under different conditions. This may involve testing under different network conditions, GPU architectures, or datasets. By publicly sharing our codebase and tools, other researchers can replicate our results, verify our claims, and potentially enhance our methodologies.

V. CONCLUSION

The paper presented a comprehensive study on enhancing the efficiency and effectiveness of correlation attacks on the Tor network. By addressing the critical challenges of slow performance, large memory consumption, and resource limitations, we introduced innovative solutions that significantly advance the field of online privacy and anonymity analysis. Our proposed approaches include the GPU-based correlation strategy and efficient data-loading technique, significantly reducing the time complexity associated with correlation attacks compared to existing methodologies. However, the paper does not solely focus on performance improvements. It also acknowledges the constraints posed by resource limitations, particularly memory consumption during the training phase.

From a practical perspective, our novel strategies present a pathway for drastically improved efficiency in conducting correlation attacks on the Tor network. Users and stakeholders in the digital realm can significantly benefit from our proposed methodologies because the correlation attacks that are more complex can now be analyzed in a much shorter time. The integration of GPU technology ensures faster processing times, while optimized data handling and resource-efficient solutions ensure that the correlation attacks maintain high accuracy even with fewer resources. These practical advantages cannot be understated, as they potentially transform how correlation attacks are executed, thus fortifying the very privacy safeguards that users trust.

In essence, this work represents a substantial leap forward in the domain of online privacy and security. By utilizing advanced GPU technology, optimized data handling strategies, and resource-efficient approaches, the paper brings about a new era in correlation attack mitigation within the Tor network. With enhanced accuracy, faster processing times, and reduced resource requirements, the paper's contributions are poised to significantly bolster the defense against correlation attacks and fortify privacy safeguards that users rely upon in the digital realm. Although our work improved the correlation attacks significantly, its performance is limited to a medium-scale (up to 10,000 flows) network only. When a larger and more sophisticated dataset is used, new techniques must be developed to reduce memory consumption, training time, and correlation performance.

For future research directions, there is immense potential for further optimizing our method. As deep learning models rapidly evolve, new techniques can refine our approach, enhancing its efficiency and broadening its applicability. Additionally, exploring the methodology's relevance to other privacy networks beyond Tor presents a promising avenue worth investigating. Lastly, while our research addressed memory constraints during training, developing even more resource-efficient strategies, particularly in regard to computational power and storage, is worth pursuing.

It is worth considering how GPU-accelerated techniques can be fine-tuned for greater efficiency. Furthermore, as the digital landscape continues to evolve, how will correlation attacks adapt, and how must our defenses evolve in response to it? These pivotal issues present noteworthy research challenges for future studies in this domain.

REFERENCES

[1] O. Berthold, H. Federrath, and S. Köpsell, "Web MIXes: A system for anonymous and unobservable internet access," in *Proc. Int. Workshop Design Issues Anonymity Unobservability*, Berkeley, CA, USA: Springer, 2001, pp. 115–129.

[2] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a type III anonymous remailer protocol," in *Proc. Symp. Secur. Privacy*, 2003, pp. 2–15.

[3] R. Dingledine, N. Mathewson, and P. F. Syverson, "Tor: The second-generation onion router," in *Proc. USENIX Secur. Symp.*, vol. 4, 2004, pp. 303–320.

[4] H. Kim, E. Kang, D. Broman, and E. A. Lee, "Resilient authentication and authorization for the Internet of Things (IoT) using edge computing," *ACM Trans. Internet Things*, vol. 1, no. 1, pp. 1–27, Feb. 2020.

[5] I. Karunanayake, N. Ahmed, R. Malaney, R. Islam, and S. K. Jha, "De-anonymisation attacks on tor: A survey," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 4, pp. 2324–2350, 4th Quart., 2021.

[6] A. Lazarenko and S. Avdoshin, "Anonymity of Tor: Myth and reality," in *Proc. 12th Central Eastern Eur. Softw. Eng. Conf. Russia*, Russia, Oct. 2016, pp. 1–5.

[7] S. M. Mohamed, N. Abdelbaki, and A. F. Shosha, "Digital forensic analysis of web-browser based attacks," in *Proc. Int. Conf. Secur. Manag. (SAM)*, 2016, p. 237.

[8] J. Feigenbaum, A. Johnson, and P. Syverson, "Preventing active timing attacks in low-latency anonymous communication," in *Proc. 10th Int. Symp. Privacy Enhancing Technol. Symp. (PETS)*, Berlin, Germany: Springer, Jul. 2010, pp. 166–183.

[9] M. Nasr, A. Bahramali, and A. Houmansadr, "DeepCorr: Strong flow correlation attacks on tor using deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 1962–1976.

[10] S. E. Oh, T. Yang, N. Mathews, J. K. Holland, M. S. Rahman, N. Hopper, and M. Wright, "DeepCoFFEA: Improved flow correlation attacks on tor via metric learning and amplification," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2022, pp. 1915–1932.

[11] A. Iacovazzi, D. Frassinelli, and Y. Elovici, "The DUSTER attack: Tor onion service attribution based on flow watermarking with track hiding," in *Proc. 22nd Int. Symp. Res. Attacks, Intrusions Defenses (RAID)*, 2019, pp. 213–225.

[12] T. Wang and I. Goldberg, "Improved website fingerprinting on Tor," in *Proc. 12th ACM Workshop Workshop Privacy Electron. Soc.*, Nov. 2013, pp. 201–212.

[13] M. Safi, S. Dadkhah, F. Shoeleh, H. Mahdikhani, H. Molyneaux, and A. A. Ghorbani, "A survey on IoT profiling, fingerprinting, and identification," *ACM Trans. Internet Things*, vol. 3, no. 4, pp. 1–39, Nov. 2022.

[14] S. P. DeFabbia-Kane, "Analyzing the effectiveness of passive correlation attacks on the Tor anonymity network," Wesleyan Univ., Middletown, CT, USA, Tech. Rep. ir:1694, 2011.

[15] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, "Users get routed: Traffic correlation on Tor by realistic adversaries," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 337–348.

[16] L. Basyoni, N. Fetais, A. Erbad, A. Mohamed, and M. Guizani, "Traffic analysis attacks on Tor: A survey," in *Proc. IEEE Int. Conf. Informat., IoT, Enabling Technol. (ICIoT)*, Feb. 2020, pp. 183–188.

[17] S. Chakravarty, M. V. Barbera, G. Portokalidis, M. Polychronakis, and A. D. Keromytis, "On the effectiveness of traffic analysis against anonymity networks using flow records," in *Proc. 15th Int. Conf. Passive Act. Meas. (PAM)*, Los Angeles, CA, USA: Springer, Mar. 2014, pp. 247–257.

[18] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal, "RAPTOR: Routing attacks on privacy in Tor," in *Proc. 24th USENIX Secur. Symp. (USENIX Secur.)*, 2015, pp. 271–286.

[19] J. Hauke and T. Kossowski, "Comparison of values of Pearson's and Spearman's correlation coefficients on the same sets of data," *Quaestiones Geographicae*, vol. 30, no. 2, pp. 87–93, Jun. 2011.

[20] J. Hiller, J. Pennekamp, M. Dahlmanns, M. Henze, A. Panchenko, and K. Wehrle, "Tailoring onion routing to the Internet of Things: Security and privacy in untrusted environments," in *Proc. IEEE 27th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2019, pp. 1–12.

[21] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, "Low-resource routing attacks against tor," in *Proc. ACM workshop Privacy Electron. Soc.*, Oct. 2007, pp. 11–20.

[22] K. Müller, "Defending end-to-end confirmation attacks against the Tor network," M.S. thesis, Dept. Comput. Sci. MTDMT Technol., Gjøvik Univ. College, NTNU, Norway, 2015.

[23] T. Chothia and A. Guha, "A statistical test for information leaks using continuous mutual information," in *Proc. IEEE 24th Comput. Secur. Found. Symp.*, Jun. 2011, pp. 177–190.

[24] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright, "Timing attacks in low-latency mix systems," in *Proc. 8th Int. Conf. Financial Cryptogr. (FC)*, Key West, FL, USA: Springer, Feb. 2004, pp. 251–265.

[25] J. Tian, G. Gou, Y. Guan, W. Xia, G. Xiong, and C. Liu, "Universal perturbation for flow correlation attack on tor," in *Proc. IEEE Int. Perform., Comput., Commun. Conf. (IPCCC)*, Oct. 2021, pp. 1–9.

[26] Y. Zhu, J. Hou, Y. Song, Y. Zheng, T. Huang, and H. Wu, "Processing data of correlation on GPU," in *Big Data in Astronomy*. Amsterdam, The Netherlands: Elsevier, 2020, pp. 139–163.

[27] M. A. Lee, "GPU multiple sequence alignment Fourier-space cross-correlation alignment," Johns Hopkins Univ., Baltimore, MD, USA, Tech. Rep. EN.600.639, 2013.

[28] E. León-Sandoval and L. I. Barbosa-Santillán, "Data intensive parallel tree algorithm patterns based on GPUs," in *Proc. Int. Conf. Data Sci. Inf. Technol.*, Jul. 2018, pp. 69–73.

[29] C. Dallas, M. Wu, V. Chou, A. Liberzon, and P. E. Sullivan, "Graphical processing unit-accelerated open-source particle image velocimetry software for high performance computing systems," *J. Fluids Eng.*, vol. 141, no. 11, Nov. 2019, Art. no. 111401.

MUHAMMAD ASFAND HAFEEZ received the B.S. degree in electrical engineering from the University of Management and Technology, in 2021. He is currently pursuing the master's degree in IT convergence engineering with Gachon University, South Korea. His research interests include cryptography, GPU computing, deep learning, and hardware implementations.

YASIR ALI received the B.S. degree in electrical engineering from the University of Engineering and Technology, Peshawar, Pakistan, in 2021. He is currently pursuing the M.S. degree in IT convergence engineering with Gachon University, South Korea. His research interests include machine learning, federated learning, and cyber security.

KYUNG HYUN HAN received the B.Sc. degree in computer engineering, the M.S. degree in electronics and computer engineering, and the Ph.D. degree in electronics and computer engineering from Hongik University, South Korea, in 2015, 2017, and 2023, respectively. He is currently a Researcher with Information Security and Machine Learning Laboratory, Gachon University, South Korea. His research interests include cyber security, GPU computing, machine learning, and blockchain.

SEONG OUN HWANG (Senior Member, IEEE) received the B.S. degree in mathematics from Seoul National University, in 1993, the M.S. degree in information and communications engineering from the Pohang University of Science and Technology, in 1998, and the Ph.D. degree in computer science from the Korea Advanced Institute of Science and Technology, South Korea. He was a Software Engineer with LG CNS Systems Inc., from 1994 to 1996. He was also a Senior Researcher with the Electronics and Telecommunications Research Institute (ETRI), from 1998 to 2007. He was a Professor with the Department of Software and Communications Engineering, Hongik University, from 2008 to 2019. He is currently a Professor with the Department of Computer Engineering, Gachon University. His research interests include cryptography, cybersecurity, and artificial intelligence. He is also an Editor of *ETRI Journal*.