A structure-aware and explainable approach to website fingerprinting using graph neural networks⋆

Zulu Okonkwo[a,*], Ernest Foo[a], Zhe Hou[a], Qinyi Li[a] and Zahra Jadidi[a]

[a]Griffith University, QLD, Australia

ARTICLE INFO

Keywords:
Network Traffic Classification
Web Fingerprinting
Tor
Graph Neural Networks

ABSTRACT

Website fingerprinting (WF) poses a significant threat to anonymity networks such as Tor, allowing adversaries to infer visited sites from encrypted traffic. Packet direction has emerged as a dominant feature in WF, outperforming timing and size-based features, even against Tor defences, particularly when leveraged by deep learning (DL) models, yet its resilience remains poorly understood. To investigate this behaviour, we propose a structure-aware WF method that represents each traffic trace as a graph in which nodes correspond to burst-level temporal bins and edges encode both temporal adjacency and data driven dependencies estimated through mutual information. This graph formulation organises burst-level directional information into a topology that exposes temporal continuity and dependency structure across the trace, enabling a GNN to learn request response sequencing, inter-burst interactions, chronology, and non-local dependencies. Directional features show stronger transferability because they are represented at a coarse-grained burst level. This preserves stable web interaction patterns while suppressing packet-level details that are susceptible to distortion by Tor defences, such as obfuscation, including packet-length obfuscation. In closed-world experiments, the proposed method achieves the highest average defended accuracy of 62.99%, while in open-world settings it yields superior precision–recall performance. The graph representation supports a structured examination of traffic traces, providing insights into the robustness of packet direction across defence scenarios and illustrating how graph-based modelling can strengthen systems for encrypted-traffic analysis.

1. Introduction

Tor is a widely used service that enables anonymous connections and safeguards users' privacy. It protects users by constructing multi-hop circuits of onion routers, negotiating per-hop session keys, and relaying TCP streams in fixed-size cells, so that no single relay can observe both the user and the destination [1]. Currently, more than 3 million active Tor users rely on Tor to protect their privacy [2]. However, this widespread adoption has also made Tor a prime target for adversaries exploiting various network vulnerabilities. One prominent class of attacks targeting Tor is website fingerprinting (WF). Although Tor is designed to conceal the source and destination of communication, WF attacks exploit side-channel information to infer the websites visited by Tor users, thereby undermining anonymity and posing a significant threat to user privacy and Tor's resistance to traffic analysis.

WF is a representative traffic analysis attack that leverages side-channel features, such as packet direction and timing, to disclose the visited websites by a user [3, 4]. DL approaches, especially CNNs, have been widely adopted in traffic analysis, and most WF systems are built on CNN-based architectures. Early work [3, 4] demonstrated that packet direction sequences alone can reveal the websites visited by Tor users. To mitigate such leakage, Tor defences such as WTF-PAD [5] employ adaptive padding to mask traffic patterns. More recent studies [6, 7] have further shown that coarse-grained features, obtained by aggregating traffic over temporal segments, are often more robust and informative than fine-grained packet-level features. Nevertheless, these methods still rely on CNNs primarily for local feature extraction, whereas accurate WF classification also requires capturing dependencies across the full traffic trace.

This leaves important gaps in WF research. Although prior studies have shown that packet directional burst level features remain effective under Tor defences [6, 7], most existing methods still model traffic as sequences and therefore emphasise local temporal patterns, ignoring relationships across the full trace. This leaves two issues insufficiently addressed: why coarse-grained directional features remain robust under defensive perturbation, and whether explicitly modelling non-local dependencies can improve both performance and interpretability. These limitations motivate the need for a framework that can model burst-level traffic as a structured object, capture both temporal and non-local dependencies, and better explain robustness

Graph Neural Networks (GNNs) provide a natural way to address these limitations. Like CNNs, GNNs rely on locality and weight sharing, but they operate on graph-structured, non-Euclidean data rather than fixed grids. This is well suited to packet direction traces, which are temporally ordered but variable in length and context-dependent. CNNs mainly capture local relationships, RNNs often struggle with long-range dependencies, and transformers do not consistently scale well as trace length increases (see Figure 1). In contrast, GNNs can represent traffic traces as graphs in which nodes correspond to packets or burst-level bins and

*Corresponding author
zulu.okonkwo@griffithuni.edu.au (Z. Okonkwo); e.foo@griffith.edu.au (E. Foo); z.hou@griffith.edu.au (Z. Hou); qinyi.li@griffith.edu.au (Q. Li); z.jadidi@griffith.edu.au (Z. Jadidi)
ORCID(s):

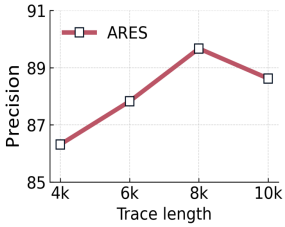

Figure 1: The attention-based ARES [6] model performs best at intermediate trace lengths, then declines.

edges encode temporal transitions or other dependencies. This formulation enables the model to capture both local chronology and non-local interactions, yielding richer traffic representations that may also be more robust to obfuscation such as padding, timing perturbation, and traffic splitting. This work addresses the following research questions:

- **RQ1:** Can graph-based representations improve WF in defended and undefended settings?
- **RQ2:** Do temporal and non-local dependencies add value beyond sequence-based local features?
- **RQ3:** Can graph-based modelling explain the robustness of coarse-grained directional features under Tor defences?

In this work, we present a GNN-based WF attack that addresses these questions by modelling each traffic trace as a connected graph. Our method preserves packet chronology through edges that emphasise both local and non-local relationships between trace segments. The proposed design is intended not only to improve classification effectiveness but also to provide a more principled explanation of the model and its components. Our contributions are as follows:

- We use entropy to quantify the information that selected features reveal under Tor defences. Using the features as node attributes, we construct graph representations of Tor traces that preserve packet chronology through temporal edges. Using mutual information, we identify later nodes that depend on earlier nodes and connect them via causal edges. This representation captures both local packet order and non-local dependencies, producing informative graph structures for learning on Tor traffic.
- To leverage this representation, we develop a GNN-based classifier for trace graphs that aggregates information over nodes, temporal, and causal edges to learn graph level embeddings for WF. By modelling local packet order and dependency links between non-adjacent nodes, the classifier captures structural information that remains informative under Tor defences, improving attack effectiveness on defended traffic.
- We conduct extensive experiments on publicly available datasets to evaluate our GNN-based attack in both open and closed-world scenarios, showing performance comparable to or better than state-of-the-art methods. We further present an ablation study covering GNN variants, explainability via feature analysis and rule extraction, and component analysis via masking and architecture evaluation, to examine their impact on classification performance.

2. Related Work

Similar to traditional encrypted traffic classification, early WF attacks relied on machine learning (ML) with handcrafted statistical features. Recent WF attacks, however, increasingly adopt deep learning (DL), which automates feature extraction and improves performance. Accordingly, our review focuses on DL-based WF methods to reflect current advancements.

2.1. Deep Learning Based WF Attacks

Deep learning has proven effective for complex tasks, with Convolutional Neural Networks (CNNs) becoming the leading approach for local feature extraction in WF. Features used for website fingerprinting tasks are generally classified into two categories: per-packet feature sequences and statistical features.

Per-packet features are sequences that describe each packet in a traffic trace, such as packet direction or timing. These features have been widely used in WF. Rimmer et al. [3] used packet direction to show that DL methods can achieve WF accuracy comparable to that of traditional ML classifiers. They evaluated several DL architectures, including autoencoders, CNNs, and LSTMs, demonstrating the potential of DL for WF. Bhat et al. [8] used packet direction, inter-arrival time (IAT), and seven cumulative statistical features as inputs to a ResNet-18 classifier, achieving a true positive rate of 88.8% against the WTF-PAD defence. Sirinam et al. [4] used packet direction and developed a CNN classifier based on the VGG16 architecture, achieving 90.7% accuracy on WTF-PAD-defended Tor traffic. Rahman et al. [9] combined timing and packet direction features with a classifier similar to that of Sirinam et al. [4], achieving 93.5% accuracy on WTF-PAD-defended traffic, a 2 percentage point improvement over Sirinam et al. [4]. Beyond CNN-based approaches, Guan et al. [10] incorporated an attention mechanism into CNNs to address the challenges of multi-tab WF. Bahramali et al. [11] introduced NetCLR, which uses contrastive learning, while Jin et al. [12] proposed TMWF, a Transformer-based WF classifier built on a CNN backbone.

Recently, statistical features computed over trace segments have gained attention because they remain effective under padding-based defences. Shen et al. [7] segmented traffic traces into time slots and constructed a matrix representation by counting the number of incoming and outgoing packets within each time slot. This approach demonstrated significant robustness to Tor defences, achieving an accuracy of 96.58% on WTF-PAD-defended traffic. Deng et al. [6] also computed features over time segments and developed ARES, a classifier that uses CNNs for local feature extraction and Transformers for website identification. Deng et al. [13] further proposed Holmes for early-stage WF, using aggregated directional features extracted from time segments of a trace. The reviewed studies predominantly relied on either raw packet direction [3, 4, 8] or modified representations of raw packet direction [7, 13] as input features for WF. However, the robustness of packet direction against traffic obfuscation remains underexplored. Understanding

why it persists as an effective feature could inform both stronger WF attacks and improved defences.

2.2. GNN based WF attacks

GNNs are neural networks for graph-structured data that leverage graph topology and node or edge features to learn representations for tasks such as classification and prediction. In traffic analysis, a key challenge is to represent traffic traces as graphs that preserve the underlying relationships, where nodes typically correspond to packets or flows and edges encode their relationships.

Using packets as nodes, Shen et al. [14] introduced Traffic Integration Graphs (TIG), which models traffic flow as a graph of bursts, with packets as nodes and intra and inter-burst edges that capture relationships within and across bursts. Using packet length and direction as node features, their method achieved 89% accuracy in identifying decentralised applications. Other packet-based designs [15, 16] adopted similar graph structures for general traffic classification rather than WF.

Using flow as nodes, Lu et al. [15] proposed a method for constructing traffic graphs in which nodes represent individual network flows. Traffic traces are segmented into flows using five-tuples. For each flow, key features, including packet length, time interval, direction sequence, and the timestamp of the first packet, are extracted and encoded as node features. Edges between nodes are established according to specific threshold criteria. The method was evaluated using a custom dataset, demonstrating superior performance compared to existing approaches. Other flow-based designs [17, 18] follow a similar structure to address traffic classification tasks. Beyond this, several studies have built graphs from modalities such as packet bytes [19] and website resource information [20], further showing that GNNs can capture complex traffic patterns and relationships.

2.2.1. WF Defences

As DL based WF attacks demonstrated high effectiveness, researchers focused on developing countermeasures aimed at disrupting these attacks. Proposed defences primarily pad, delay or split traces. Padding-based defences such as WTF-PAD [5], Walkie-Talkie [21], Front [22], and RegularTor [23], introduce dummy traffic or standardised burst sizes randomly of adaptively to obscure patterns. Alternatively, delay-based approaches such as Tamaraw [24] inject timing perturbations to disrupt temporal consistency. Traffic-splitting defences, such as Trafficsliver [25], fragment traffic across multiple paths or sessions to conceal local dependencies altogether.

3. Our Methodology

This section presents our WF attack methodology, starting with the threat model then the intuition behind our feature selection and extraction inspired by the principles of information leakage [26]. We then detail the design and structure of our proposed graph-based representation.

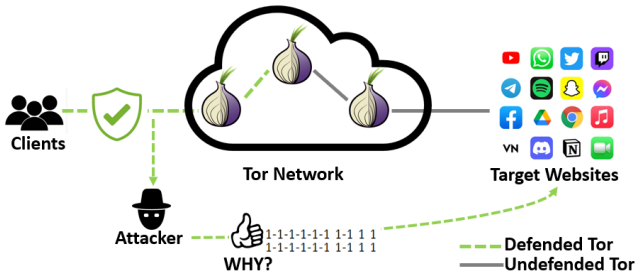

Figure 2: Threat Model.

3.1. Threat Model and Attack Goals

Figure 2 illustrates the threat model underlying our WF attack, detailing its foundational assumptions. The model assumes a passive adversary who solely monitors traffic (defended or undefended) without altering the contents of the trace. The adversary's capability is limited to capturing and recording traffic for subsequent offline analysis. The traffic sniffing tool is strategically positioned between the Tor proxy and the guard node, suggesting that the adversary could be any individual or entity operating quietly within the victim's local network.

After collecting network traces in a passive WF attack, the adversary employs advanced analytical techniques to infer visited websites. In a supervised learning setting, this involves offline training, requiring labelled traffic and data preprocessing. The goal is to extract rich features to accurately classify the visited websites. Given the significant attention WF attacks have received, various defences have been proposed to counteract their success. Consequently, we assume that the client has the capability to deploy certain defences to thwart the adversary's efforts; we refer to this as *defended Tor traffic*. Recent literature [7] suggests that adversaries often possess prior knowledge of deployed defences, allowing them to adapt and evaluate their WF attacks accordingly. Consistent with recent research methodologies in traffic analysis [3, 4, 7, 13], we consider closed and open-world scenarios for our experiments. The closed-world scenario represents a controlled experimental environment where the adversary possesses a priori knowledge of the entire set of websites visited by clients. In this strictly delimited context, the adversary maintains comprehensive access to all network traces, enabling precise analytical characterization. Conversely, the open-world scenario offers a more realistic experimental framework that better reflects real-world network behaviour. In this setting, clients access websites freely, while the adversary's visibility is inherently limited, restricted to a small, known subset of monitored sites. The vast majority of client activity involves unmonitored (often unknown) websites, introducing substantial ambiguity and significantly increasing the difficulty of accurate inference.

3.2. Feature Selection and Leakage Analysis

In WF, a trace refers to the recorded sequence of network packets exchanged during a user's visit to a website. These traces are typically represented as sequences of +1

Table 1
Feature Index.

Index	Feature
0	total_packet_count
1	incoming_packets
2	outgoing_packets
3	incoming_bursts
4	outgoing_bursts
5	avg_incoming_burst_size
6	avg_outgoing_burst_size
7	max_burst_length
8	direction_transitions
9	traffic_intensity
10	mean_relative_position
11–19	percentiles (10th to 90th)

and –1(integers are sometimes replaced with the packet timestamp), indicating outgoing and incoming packets, respectively. This encoding captures the directional flow of traffic and serves as a compact representation of underlying communication patterns.

Shen et al. [7] categorized traffic features into two, namely statistical and per-packet feature sequences. They considered statistical features coarse and per-packet features fine-grained. Because both coarse and fine-grained attributes fluctuate under Tor defences, they argue that neither alone suffices for robust WF. Shen et al. however empirically demonstrated that counting packets and bursts transmitted within carefully chosen time intervals remains informative for countering defences based on dummy packet padding. Building on their findings, we carefully define a set of features based on selected time intervals. Table 1 presents the features along with their corresponding indices. Before we extract the features from each trace we first collapse the original packet sequence into its burst sequence summing consecutive packets in the same direction. The collapsed trace is segmented(binned) as described in figure5, section 3.3 and features are then extracted for every bin. The features can be categorized into the following classes: Packet-based: indices 1–2, Burst-based: indices 3–7, Traffic flow: indices 8–9, Position-based: index 10, Statistical: indices 11–19.

To illustrate the feature extraction process, we sample one time bin from a randomly chosen trace. Let the full trace X be segmented into n equal-length intervals $\{ X_{\text{bin}_1}, \dots, X_{\text{bin}_n} \}$. For each interval X_{bin_i} we compute the 20-dimensional feature vector defined in Table 1. Consider the first bin, whose packet-direction sequence is

$$X_{\text{bin}_1} = \big[+2, -5, +3, -2, +7, -5, +6, -3, +2, -8, +2, -3, +2, -3, +4, -6, +5, -7, +1, -1, +6, -8, +2, -3, +4, -5, +6, -7, +9, -9 \big],$$

where a term "$+k$" (respectively "$-k$") denotes a burst of k outgoing (incoming) packets. In the remainder of this section we use X_{bin_1} to demonstrate, step by step, how each feature is derived before being fed to the classifier.

- **Total packets**: The total count of packets observed within a bin. Represents overall activity in the bin and provides a fundamental measure of traffic volume.

$$\sum_{i=1}^{n} |X_{\text{bin}_1}[i]| = 136$$

- **Incoming Packets**: The count of all the incoming packets in a bin. Distinguishes download-heavy versus upload-heavy traffic.

$$\sum_{i=1}^{n} \Big| X_{\text{bin}_1}[i] \Big| \mathbf{1}\{X_{\text{bin}_1}[i] < 0\} = 75$$

- **Outgoing packets**: The count of all outgoing packets in a bin. Identifies user requests and potential bursts of outgoing activity.

$$\sum_{i=1}^{n} X_{\text{bin}_1}[i] \cdot \mathbf{1}\{X_{\text{bin}_1}[i] > 0\} = 61$$

- **Incoming bursts**: The count of incoming bursts in a bin. A higher count of incoming bursts often signifies fragmented downloads or streaming-like behaviour.

$$\sum_{i=1}^{n} \mathbf{1}\{X_{\text{bin}_1}[i] < 0\} = 15$$

- **Outgoing Bursts**: The count of outgoing bursts in a bin. A higher count of outgoing bursts may indicate user-initiated activities such as form submissions or video uploads.

$$\sum_{i=1}^{n} \mathbf{1}\{X_{\text{bin}_1}[i] > 0\} = 15$$

- **Mean Incoming Bursts**: The mean size of the incoming bursts in a bin. Captures typical download chunk sizes, useful for differentiating streaming, browsing, or file downloads.

$$\frac{\sum_{i=1}^{n} |X_{\text{bin}_1}[i]| \cdot \mathbf{1}\{X_{\text{bin}_1}[i] < 0\}}{\sum_{i=1}^{n} \mathbf{1}\{X_{\text{bin}_1}[i] < 0\}} = 5.000$$

- **Mean Outgoing Bursts**: The mean size of the outgoing bursts in a bin. Identifies common upload behaviours and distinguishes between interactive traffic (e.g., messaging) and bulk uploads.

$$\frac{\sum_{i=1}^{n} X_{\text{bin}_1}[i] \cdot \mathbf{1}\{X_{\text{bin}_1}[i] > 0\}}{\sum_{i=1}^{n} \mathbf{1}\{X_{\text{bin}_1}[i] > 0\}} = 4.067$$

- **Max Burst Length**: The maximum size of bursts (largest single burst, either incoming or outgoing) in a bin. Measures the largest sustained traffic flow, which can indicate bulk transfers or long user sessions.

$$\max_{1 \le i \le n} |X_{\text{bin}_1}[i]| = 9$$

- **Direction Transitions**: The number of times the sign of the burst size changes within a bin (e.g., incoming to outgoing or vice versa). High transition count often corresponds to interactive activities like web browsing or messaging. Low transition count may indicate sustained downloads (e.g., video streaming).

$$\sum_{i=2}^{n} \mathbf{1}\left\{ \text{sign}(X_{\text{bin}_1}[i]) \neq \text{sign}(X_{\text{bin}_1}[i-1]) \right\} = 29$$

- **Traffic Intensity**: The absolute count of bursts observed in a bin. Captures overall traffic volume within a bin. Helps differentiate between lightweight and heavy network activity.

$$\sum_{i=1}^{n} 1 = 30$$

- **Mean Relative Position**: The average normalized time position of bursts within a bin, scaled relative to the time span of the entire sequence. Higher values indicate bursts occurring later in the session, while lower values indicate bursts happening early in the session. This feature helps understand when key activity happens in a trace.

$$\frac{1}{n} \sum_{i=1}^{n} \frac{t_i - t_{\text{min}}}{t_{\text{max}} - t_{\text{min}}} = 0.500$$

- **Percentiles**: To capture variability in burst sizes the $10th$ to $90th$ percentiles of the absolute burst sizes in a bin are calculated. These percentiles summarize the distribution of burst sizes within a bin. Helps identify whether traffic is uniform or skewed. Higher percentiles (e.g., 90th) capture outliers (e.g., large file downloads), while lower percentiles (e.g., 10th) highlight background traffic.

$$P_{10} = 2, \quad P_{20} = 2, \quad P_{30} = 3, \quad P_{40} = 3,$$
$$P_{50} = 4.5, \quad P_{60} = 5, \quad P_{70} = 6,$$
$$P_{80} = 7, \quad P_{90} = 8$$

To evaluate the impact of these features, we employ entropy-based information leakage metrics as in [26] to quantify the extent of information an adversary can infer from the features. If $H(W)$ represents the entropy of the websites W, and $H(X \mid F)$ represents the conditional entropy of the websites W after observing some features F, the information leakage is:

$$\textit{Information Leakage} = H(W) - H(W \mid F) \tag{1}$$

The entropy of websites $H(W)$ is calculated using the *entropy* function, which takes the *labels* (representing the websites visited) as input. The formula for entropy is:

$$H(W) = -\sum_{i=1}^{n} P(w_i) \cdot \log_2(P(w_i)) \tag{2}$$

where w_i is a specific website label, $P(w_i)$ is the probability of visiting website w_i, calculated as the frequency of w_i in the *labels* divided by the total number of *labels*. The entropy of websites $H(W)$ is essentially the uncertainty in predicting the website a user has visited, based on the distribution of website labels. The conditional entropy $H(W|F)$ is calculated for each *feature category*. It measures the uncertainty of the website labels (W) given the values of a specific feature (F). The conditional entropy is given by:

$$H(W|F) = \sum_{f \in F} P(f) \cdot H(W|F = f) \tag{3}$$

where F is the set of feature categories, $P(f)$ is the probability of the feature value being in category f, $H(W|F = f)$ is the entropy of website labels restricted to the subset of traces where the feature value falls in category f. Figure 3 depicts the information leakage for our selected features. To compute the information leakage we randomly select 500 trace graphs (undefended and defended) per website. Then we measure the information leakage in a closed-world setting. Figure 3 shows that the selected features remain robust against the defences and, in some cases, leak more information than undefended traffic. Incoming bursts, outgoing bursts, and total packet leak the most features in the case of the Walkie-Talkie defence. This increased leakage can be attributed to the reliance of Walkie-Talkie on a half-duplex communication model, which inherently generates distinct bursty traffic patterns, making these features highly exploitable for burst-based analysis. While percentiles which show the burst sizes distribution, although exhibiting greater variability, leak the most for WTF-PAD and front.

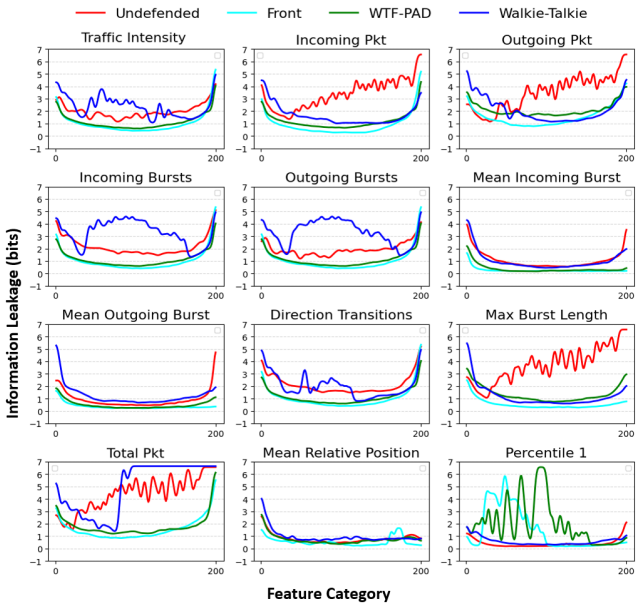

Figure 3: Information leakage for individual features.

3.3. Graph Representation

As stated in Section 2.2, representation for DL-based WF tasks is always a primary concern that must be addressed to ensure proper classification. Figure 4 shows the

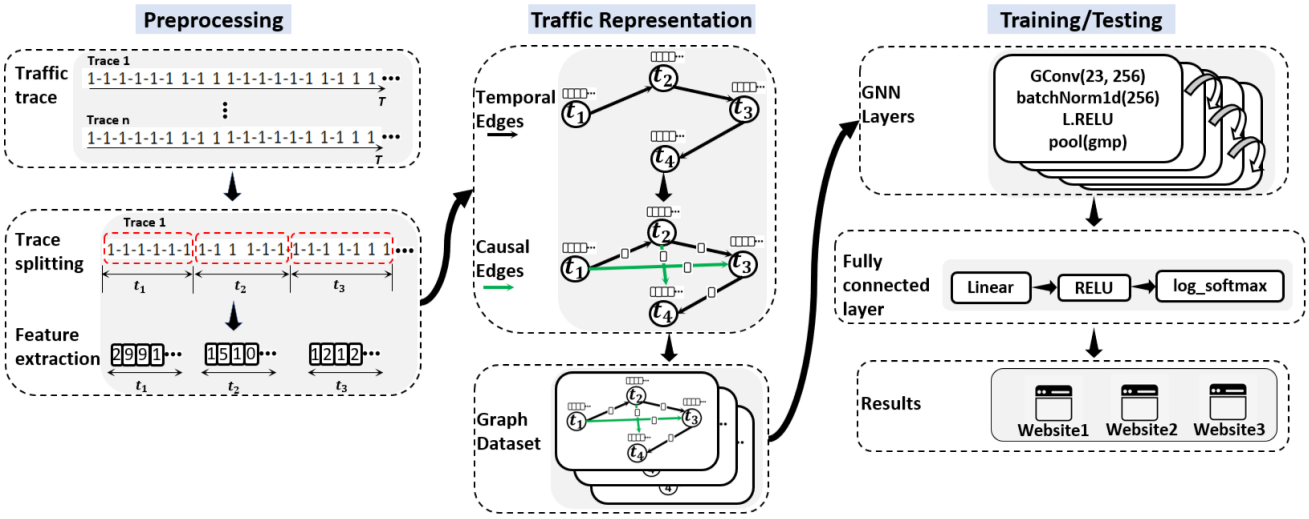

Figure 4: Model Overview: The raw traces are segmented and transformed into sequences of bins, where features are calculated for each bin. These bins are represented as nodes in a graph, with sequential connections forming a labelled graph structure. The dataset is split into three (train/val/test), and the labelled graphs are processed through a hybrid model for training. The trained model predicts the website associated with each traffic trace.

overview of our model, divided into three parts: preprocessing, training and testing. We begin by explaining how the graphs are generated. Existing work [14, 17, 19] used graphs for encrypted traffic classification, leveraging access to raw traffic data to extract richer and more detailed information. However, in the context of WF, the available data is restricted to directional time-series traces of both undefended and defended Tor traffic. This constraint poses challenges as it limits the variety of accessible information, thereby increasing the complexity of the task. Graphs express abstract entities, with nodes and edges representing relationships between predefined entities. In DL-based traffic classification methods, extracted features are often transformed to meet the classifier's input requirements, frequently with limited consideration for the underlying data structure. This transformation leads to misrepresentation and loss of information. While methods utilizing GNNs [14, 19] have demonstrated promising results on specific datasets, they consistently exhibit traces of poor generalization. We observe that the delicate structures of graphs can be either highly beneficial or significantly detrimental for network traffic classification-related tasks for the following reasons: The efficacy of graph-based methods hinges on the integrity of the graph structure and the attributes of its nodes and edges. Minor perturbations in the graph data, such as missing nodes or artificially created edges intended to improve the graph structure, can substantially distort the representation. Such distortions often lead to a misrepresentation of the actual traffic trace, undermining the fidelity of subsequent analyses. While data transformation is an essential step in preparing input for classification models, it is imperative that the transformed data faithfully represents the underlying input without deviating excessively from its original form.

Network traffic data, is sequential and represented as time series data, with events ordered chronologically. A visit to a website results in a trace denoted as $W = (w_1, w_2, w_3, \ldots, w_l)$, where l is the length of the trace. Each w_k (for $1 \leq k \leq l$) is a tuple $w_k = (t_k, d_k)$, where t_k denotes the arrival time and d_k represents the direction of the k-th packet. d_k can either be 1 for an outgoing packet or -1 for an incoming packet. Prior work by Shen et al. [7] and Deng et al. [13] showed that statistical features of packets within a defined time interval are capable of withstanding defences. Motivated by this, we divide each traffic trace into multiple segments representing the graph nodes and introduce two edge types to create connected graphs.

First, a trace is read into the system and broken into smaller parts where each segment is characterized by features described in Section 3.2. For the remainder of this paper, we refer to these segments as bins. The number of bins serves as a critical hyperparameter that must be determined before graph generation. To identify the optimal number of bins, we systematically evaluate the impact of varying the bin count between 10 and 50. This analysis is conducted on both undefended and defended datasets, allowing us to assess the robustness and generalizability of the bin-based feature representation across different traffic scenarios. The results presented in Figure 5 illustrate the impact of varying the number of bins on the model's performance across four key metrics: accuracy, precision, recall, and $f1$-score. The results illustrates the following observations: When the number of bins is low (less than 25), the model's performance is suboptimal across all metrics. This is because the smaller number of bins limits the granularity of feature extraction, which reduces the model's ability to capture meaningful patterns in the traces. Increasing the number of bins (beyond 30) does not result in significant improvements in performance. This plateau suggests that overly fine-grained segmentation of the trace does not provide additional meaningful information and can introduce noise or redundancy. For a

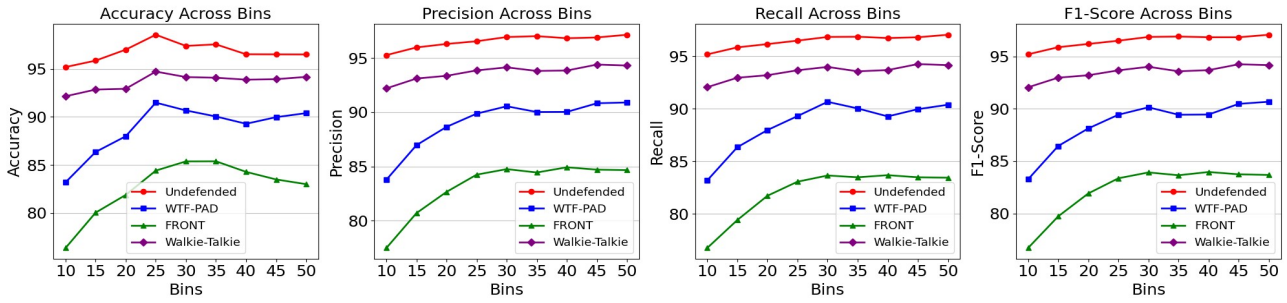

Figure 5: Impact of bin count on model performance across accuracy, precision, recall, and F1-score. Performance is sub-optimal below 25 bins and shows diminishing returns above 30, with optimal results in the 25–35 bin range.

trace length of 5000, the range of 25 to 35 bins achieves the best balance across all metrics, providing the highest performance while avoiding diminishing returns. This range offers sufficient granularity for effective feature extraction without over-segmenting the data. Based on the results of the experiment illustrated in Figure 5, we determine that a trace can be effectively characterized using 30 bins. Consequently, this bin count is adopted for all subsequent experiments. To ensure fair comparisons with prior studies, the length of each traffic trace is standardized to 5000. For each bin, we compute the defined features and append them as node attributes in the resulting graph representation.

Temporal Edges. Since the bins represent sequential segments of the trace, the temporal relationship between them is preserved by adding temporal edges, where a temporal edge is created from one bin to the next in the sequence, forming a relationship between nodes $b_1, b_2, \cdots, b_n$ such that the edges are defined as: $\{(b_1, b_2), (b_2, b_3), \ldots, (b_{n-1}, b_n)\}$. This edge type generates a *network path graph*, where all vertices are arranged along a single straight line. While this form of edge construction does not introduce additional topological complexity to the graph, it ensures that the temporal ordering inherent in the original traffic trace is accurately maintained within the graph representation. Formally, given nodes i and j, the edge relationship is encoded in the adjacency matrix $\mathbf{a}$, where each entry a_{ij} is defined as:

$$a_{ij} = \begin{cases} 1, & \text{if } j = \text{succ}(i), \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Here, succ(i) denotes the *successor* node of i in the sequence. This representation ensures that the time-series structure of the trace is preserved, facilitating accurate graph-based modelling of network traffic. Each edge is initialized with an attribute that represents the difference between the mean arrival times of packets in the connected bins. This attribute captures the temporal gap between consecutive bins of the traffic trace. To enhance message passing among nodes, these edge types are undirected.

Causal Edges. This edge type augments the graph with long range, directed dependencies that cannot be captured by temporal edges. A causal edge connects two non-adjacent bins whose Mutual Information (MI) suggests that earlier traffic in the trace is likely to influence later traffic. This emphasizes directed, informative relationships that enrich the structural and semantic expressiveness of the graph. From Fig. 3, we select the three most informative features: total, incoming, and outgoing packets. Each is discretized into eight bins using quantiles computed over the entire dataset. For every feature, we then count how often a value in bin a (source) co-occurs with a value in bin b (target) across all non-adjacent bin pairs and across all traces, yielding a global 8×8 co-occurrence table. Normalizing these tables to joint and marginal probabilities lets us compute a feature-wise Mutual Information lookup table (LUT). For a given trace, each non-adjacent pair of bins gets three MI scores by indexing the LUTs with their discretized values; the pair's MI is the maximum of the three. We collect all such candidate MI scores in that trace and set the causal threshold to their 90th percentile. A directed causal edge is added only if the target node is forward and non-adjacent to the source, and if the MI of the source to the target node is greater than or equal to the calculated threshold.

Let the time ordered vertex sequence of a trace be $V = [b_1, \ldots, b_n]$ with $t(b_1) < \cdots < t(b_n)$ and a set of causal edges $E_c = \{(b_{p_1}, b_{q_1}), (b_{p_2}, b_{q_2}), \ldots\} \subseteq V \times V$. Also a data-driven threshold of all candidate MI values for the same trace is fixed once per trace. For causal edge creation, the following conditions must be satisfied.

- Forward & non-adjacent
  $(b_i, b_j) \in E_c \implies i < j - 1$. Every causal edge points strictly forward in time and skips at least one intermediate node, avoiding temporal edges.
- Mutual-information threshold
  $(b_i, b_j) \in E_c \implies \text{MI}(b_i, b_j) \geq \tau$. Only pairs whose mutual information exceeds the trace-specific threshold are retained.

Formally, the edge relationship is encoded in the adjacency matrix

$$a^c_{ij} = \begin{cases} 1, & \text{if } (b_i, b_j) \in E_c, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

An entry $a^c_{ij} = 1$ therefore records that "b_i is the causal predecessor of b_j". Figure 6 presents graphs from four randomly

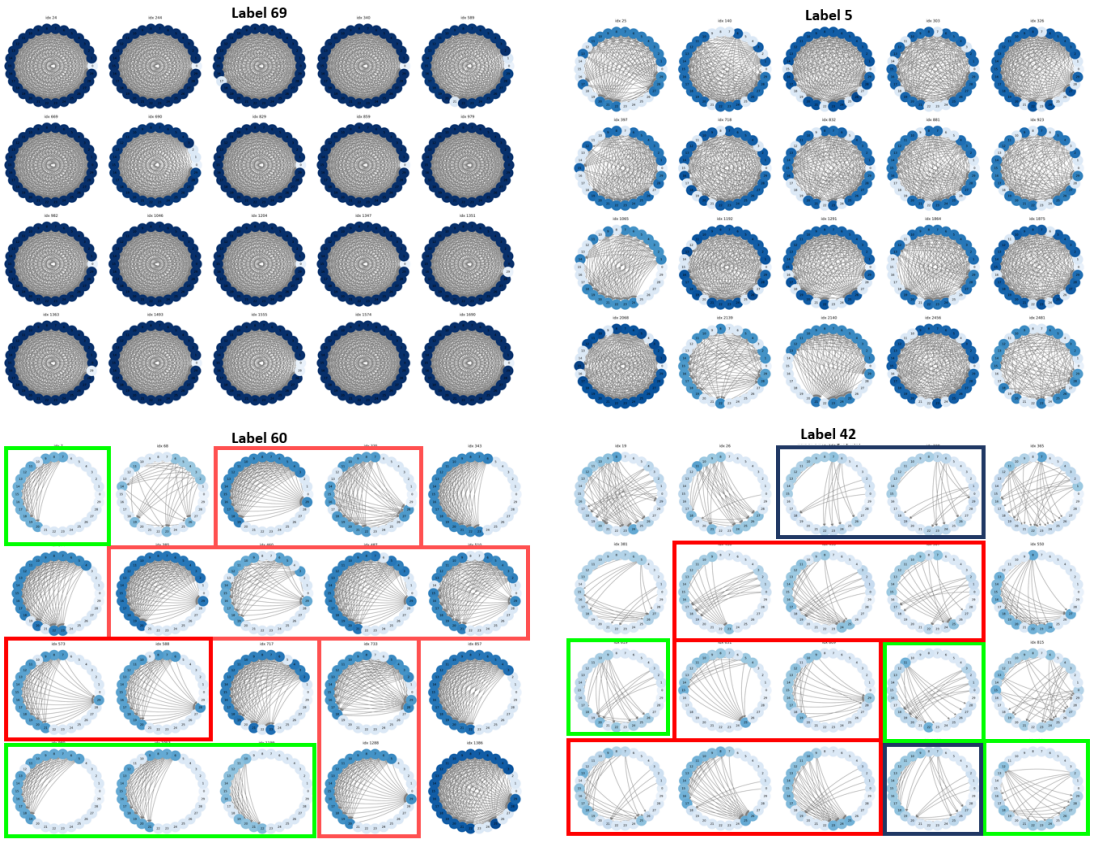

Figure 6: Graphs for four classes in the undefended dataset. Node colour intensity reflects degree centrality (darker=higher connectivity), while coloured outlines highlight recurring intra-class motifs in sparsely connected graphs.

selected website classes in the undefended dataset. The circular layout is used to depict the mandatory temporal edge. Close observation shows consistent patterns amongst graphs of the same class. Label 69 produces a denser pattern and label 42 generates sparse patterns, providing discriminating cues for the GNN to exploit. Figures 8 and 7 show example graphs under the WTF-PAD and TrafficSliver defence.

Structural features. The features introduced in Section 3.2 describe *what* happens within each time bin but not *where* that bin sits in the global temporal–causal structure. To expose this topology we append three graph theoretic descriptors to every node:

- *Out-degree* the number of outgoing edges, capturing how many future bins a node directly influences.
- *Hop index* the zero-based position of the node along the time axis, providing an absolute measure of when a node appears, capturing the distinction between "early" and "late" events in the trace.
- *Normalised centrality* the out-degree scaled by its maximum possible value $2(N-1)$, providing a length-invariant measure of the node's structural prominence within its graph.

The enhancement of the 20 burst-level statistics with degree, hop index, and centrality produces a compact 23-D node descriptor that fuses local traffic dynamics with global graph topology, increasing the classifier's sensitivity to structure driven browsing patterns.

Algorithm 1 outlines the detailed procedure for generating a graph representation of a trace. The algorithm iterates over every trace file and emits one graph per trace. Lines 1–3 clear the dataset container D and start the loop over traces. Lines 4–5 read a trace and condense packets into bursts by merging consecutive packets with the same direction e.g., $\{1,1,1,1,1-1,-1,-1,1,1\}$ becomes $\{+5,-3,+2\}$, preserving order while shortening the sequence. Lines 6–9 compute the number of bins from the trace's time span, create the bin edges, assign bursts to bins, and extract the 20 bin-level statistics that become the initial node features. Lines 10–13 discretize the three highest-leakage features (total, incoming, and outgoing packets) and, using all traces, accumulate co-occurrence counts to build global MI look-up tables (one per feature). Lines 14–16 add temporal edges between consecutive bins (bidirectional). Lines 17–23 evaluate every forward, non-adjacent bin pair, look up its three MI scores, take the maximum, and add a causal edge if it exceeds the trace-specific threshold (90th percentile of that trace's candidate scores). The temporal and causal edges are then merged in line 24. Lines 25–27 append structural node features: degree, hop index, and centrality. Lines 28–32 assemble a 5-D edge attribute for each edge: $[\Delta t_z, \mathrm{MI}_{\mathrm{out}}, \mathrm{MI}_{\mathrm{in}}, \mathrm{MI}_{\mathrm{tot}}, \mathtt{type}]$, where Δt_z is the mean time gap between the two bins, $\mathrm{MI}_{\mathrm{out}}$, $\mathrm{MI}_{\mathrm{in}}$,

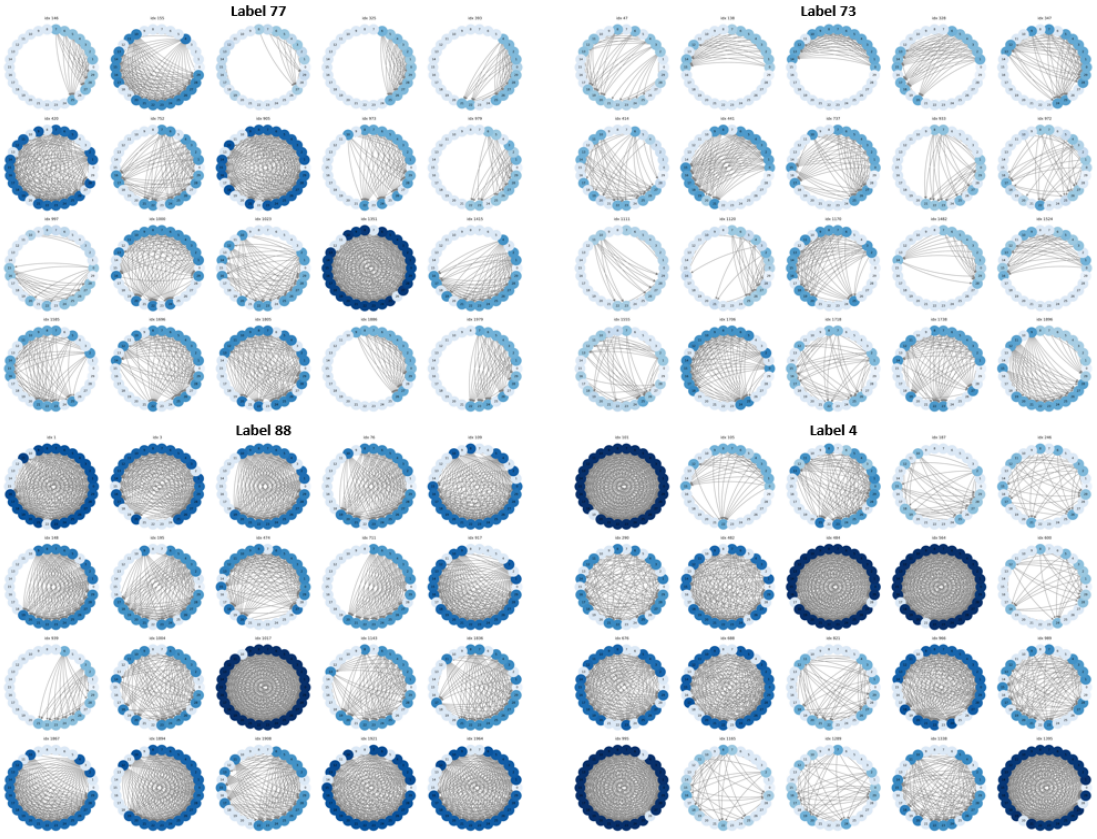

Figure 7: Graphs for four classes under the TrafficSliver defence.

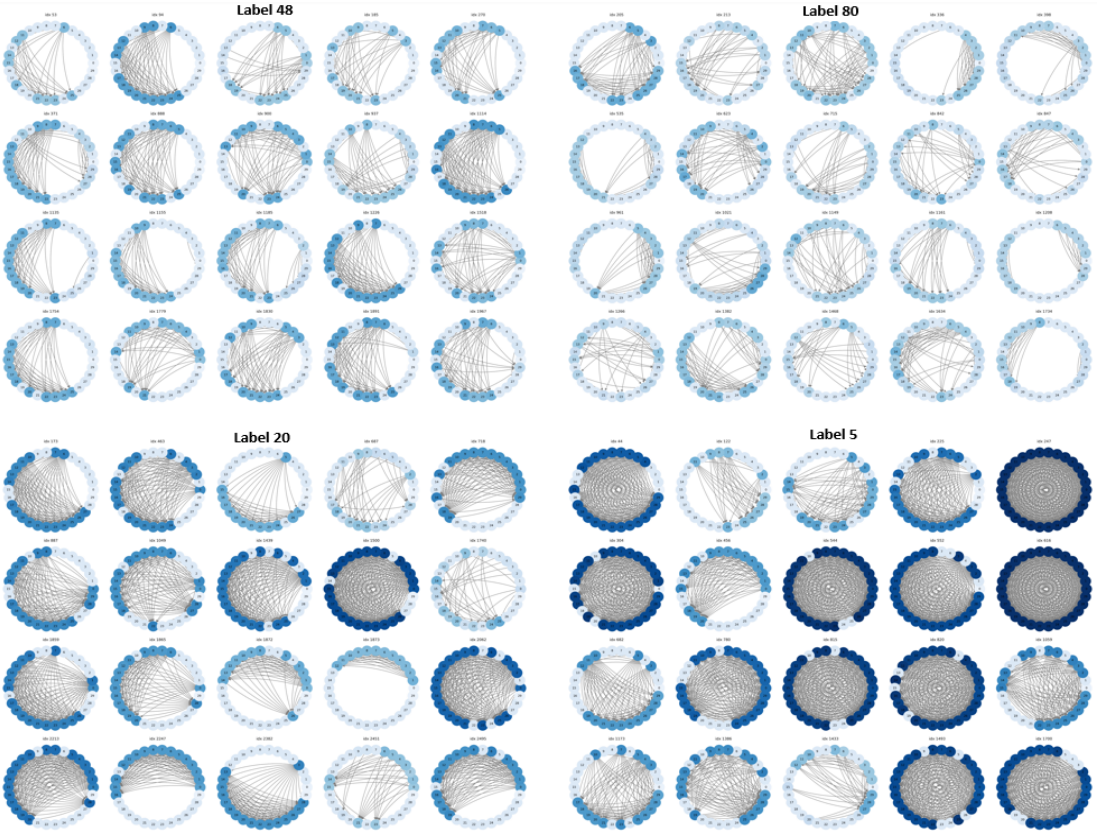

Figure 8: Graphs for four classes under the WTF-PAD defence.

Algorithm 1 Trace to Graph Generation

Inputs: set of traces $\mathcal{F}$; labels from a directory.
Options Ω:
B target_bins
d no of discrete bins for MI computation
p MI percentile threshold
`causal` build MI causal edges
`struct` append node structural feats (deg, hop, cent)
`eattr` add edge features (e.g. Δt, MI values, type)
Output: Dataset D of graphs (G, y).

1: $D \leftarrow \varnothing$
2: **for** each trace $F \in \mathcal{F}$ **do**
3: $y \leftarrow$ Extract label from the file
4: read $F \rightarrow (Time, Value)$
5: bursts $\leftarrow$ compress_by_direction$(Time, Value)$
6: Determine num_bins, $N \leq B$; compute bin_edges
7: Assign bursts to bins; compute features per bin
8: Save node matrix X and bin_edges for this graph
9: **end for**
10: **if** `causal` **then**
11: Discretise top-3 leakage feats (tp, ip, op) into d bins
12: Build MI look-up tables (feature-wise $d \times d$)
13: **end if**
14: **for** each stored (X, bin_edges, y) **do**
15: $N \leftarrow |X|$ ▷ number of bins/nodes
16: $E_t \leftarrow \{(i, i+1), (i+1, i)\}_{i=0}^{N-2}$ ▷ temporal, bi-dir
17: **if** `causal` **then**
18: Compute $\mathrm{MI}_{\max}(i, j)$ for all $j \geq i + 2$
19: $\tau \leftarrow \text{percentile}_p\left(\{\mathrm{MI}_{\max}(i, j)\}_{j>i+1}\right)$
20: $E_c \leftarrow \{(i, j) \mid j \geq i + 2,\ \mathrm{MI}_{\max}(i, j) \geq \tau\}$
21: **else**
22: $E_c \leftarrow \varnothing$
23: **end if**
24: $E \leftarrow E_t \cup E_c$
25: **if** `struct` **then**
26: append deg, hop, cent to X
27: **end if**
28: **if** `eattr` **then**
29: build 5-D edge features for all $(u, v) \in E$
30: **else**
31: edge_attr $\leftarrow \varnothing$
32: **end if**
33: $G \leftarrow (X, E,$ edge_attr$)$; append (G, y) to D
34: **end for**
35: **return** D

and $\mathrm{MI}_{\mathrm{tot}}$ are the mutual-information scores for outgoing-, incoming, and total packet features respectively, and `type`$= 0$ marks temporal edges while `type`$= 1$ marks causal edges. Lines 33–35 package the graph (nodes, edges, attributes) with its label and save it in D.

3.4. Model design

Our GNN is based on a design (GINECONV) by [27] which is a modified version of Graph Isomorphism Network (GIN) [28], that incorporates edge features into the aggregation procedure. Before introducing our model, we define some necessary preliminaries.

3.4.1. Definitions

GNNs perform optimizable transformations on graph attributes, ensuring both permutation invariance and equivariance. Through message passing, they aggregate neighborhood information to update node representations. In WF, where each trace is a graph, the task becomes a graph classification, predicting a label for the entire trace. A graph $G = (V, E)$ consists of nodes V and edges E, where $E \subseteq \{(u, v) \mid u, v \in V, u \neq v\}$. Each node $v \in V$ is initialized with a feature vector X_v and edge attributes e_{uv} for $(u, v) \in E$. Let $\mathcal{G} = \{G_1, G_2, \ldots, G_N\}$ represent a set of graphs and $\mathcal{L} = \{l_1, l_2, \ldots, l_N\}$ their corresponding labels. The goal of graph classification is to learn a representation vector h_{G_N} to predict the label l_N for the graph G_N. Modern GNNs update a node representation by aggregating information from its neighbours. After k iterations, the representation of node v encodes the structure inside its k-hop neighbourhood. Formally, the k-th layer is:

$$h_v^{(k)} = \mathrm{COM}^{(k)}\Big(h_v^{(k-1)}, \mathrm{AGG}^{(k)}\big(\{(h_u^{(k-1)}, e_{uv}) \;\big|\; u \in \mathcal{N}(v)\}\big)\Big), \tag{6}$$

where $h_v^{(k)}$ is the embedding of node v at layer k, e_{uv} is the edge-feature vector of (u, v), and $\mathcal{N}(v)$ denotes the neighbours of v. The initial node features are $h_v^{(0)} = X_v$. COM means combine and AGG means aggregate in equation 6. To obtain a single vector h_G for the entire graph G, a permutation-invariant *readout* function pools the node features after the final layer K:

$$h_G = \mathrm{READOUT}\big(\{\, h_v^{(K)} \mid v \in G \,\}\big). \tag{7}$$

Typical choices for READOUT include simple averaging or more advanced graph-level pooling operators.

Equation 6 outlines the mechanism for neighbourhood aggregation and message passing employed by the GNN layer. By operating on also edges features rather than only nodes, this methodology offers enhanced representational power compared to traditional GNNs. This capability allows it to effectively capture expressive properties of the entire graph, making it particularly well-suited for graph-level classification tasks.

To justify our choice of 4 GNN layers, we consider the over-smoothing effect, as GNN depth increases, repeated message passing causes node features to become overly similar, eventually leading to uniform representations. Rusch et al. [29] formally define over-smoothing as the exponential convergence of node features with respect to appropriate similarity measures.

Node-Similarity Measure (μ): A function $\mu : \mathbb{R}^{v \times m} \rightarrow \mathbb{R}_{\geq 0}$ quantifies the similarity between node features and satisfies the following properties:

- **Uniformity:** If all nodes in $\mathbf{X}$ share the same feature value ($X_i = c$), then $\mu(\mathbf{X}) = 0$.

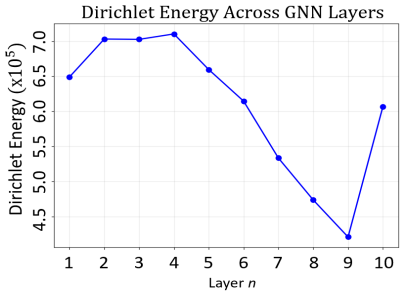

Figure 9: Dirichlet Energy across 10 GNN Layers: Dirichlet energy peaks between layers 3–5, indicating optimal expressivity and feature preservation within this range; it then declines due to over-smoothing, after which node embeddings lose discriminative power. The rebound at layer 10 reflects residual variance but remains below the peak.

Explanation: A uniform feature distribution across nodes has no similarity variation to measure.

- **Subadditivity:** $\mu(\mathbf{X}+\mathbf{Y}) \leq \mu(\mathbf{X}) + \mu(\mathbf{Y})$, for all $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{v \times m}$.
  Explanation: The similarity of combined features is at most the sum of individual similarities, ensuring consistency.

Over-Smoothing is the layer-wise exponential convergence of the node-similarity measure μ to zero as the network depth n increases. Let $\mathbf{X}^{(n)}$ denote the node feature matrix at layer n, where $\mathbf{X}^{(n)} \in \mathbb{R}^{v \times m}$ for all layers. Specifically, for constants $C_1, C_2 > 0$, it holds that:

$$\mu(\mathbf{X}^n) \leq C_1 e^{-C_2 n}. \tag{8}$$

Explanation: As the network depth increases, the similarity measure decays exponentially, leading to indistinguishable node features. This detrimental effect emphasizes the importance of carefully selecting the network depth to balance expressivity and prevent the loss of feature diversity.

Dirichlet energy [30] and Mean Avg Distance (MAD) [31] are commonly used to quantify over-smoothing in GNNs. However, Rusch et al.[29] critique MAD for violating uniformity and subadditivity, noting it can yield zero in scalar cases with same sign features, even without over-smoothing, thus limiting its reliability. In alignment with Rusch et al. [29], we adopt Dirichlet energy[30] as a theoretically sound measure of over-smoothing, as it reliably captures feature similarity while satisfying key metric properties. Figure 9 illustrates the effect of model depth on over-smoothing by plotting Dirichlet energy across 10 GNN layers using the DF dataset [4]. Dirichlet energy is computed per layer from node embeddings on a representative training batch. The Dirichlet energy increases from layers 1 to 4, peaking at layer 4. A slight decline at layer 5 indicates early signs of over-smoothing, while a sharp drop beyond layer 5 confirms significant over-smoothing. Based on this, we select 4 GNN layers to balance expressivity and feature preservation. It is important to note that the Dirichlet energy plot is not a measure of model performance, this is a common misconception. Instead, it provides a principled visualization of over-smoothing, enabling the logical selection of GNN depth.

Residual connections are introduced from the second GNN layer onwards, allowing information from earlier layers to be propagated forward. This helps mitigate issues like vanishing gradients and over-smoothing in deeper GNNs. Each GINEConv layer leverages a two-layer MLP and is followed by batch normalization and ReLU activation. After the final convolutional layer, a global max pooling operation is applied to aggregate node embeddings into a graph-level representation. The pooled output is then passed through a two-layer MLP classifier with ReLU activation and dropout regularization to improve generalization. The final output is a softmax over the target classes, producing a probability distribution per graph.

4. Evaluation

4.1. Dataset

We validate our experiments using three public WF datasets. The DF dataset [4] includes 95 websites, each with 1000 undefended traces for closed-world experiments. For open-world experiments, it has 40,000 websites, each with one undefended trace. The second dataset is the Walkie Talkie dataset [9], which contains 900 defended walkie-talkie traces of 100 websites. Lastly, the Wang dataset [21] which contains 100 undefended traces of 100 websites.

Baseline Attacks. Our study ensures a fair comparison by benchmarking against state-of-the-art deep learning-based WF attacks. We evaluate our model against CNN-based attacks (AWF[3], DF[4], Tik-Tok[9], Var-CNN[8], triplet-fingerprinting [32]), as well as attention, generative, and transformer-based attacks (BAPM [10], NetCLR [11], TMWF [12]). Although GNNs are less common in Tor WF, they are widely used in network traffic classification. We therefore also include graph-based baselines. Graph-DApp [14] is a strong GNN model for traffic classification. STC-WF [18] models inter-flow spatio-temporal correlations, whereas MRCGCN [20] captures multi-level resource and flow dependencies through graph learning. These form our baseline for comparison.

4.1.1. Training Specifications

We generate the graphs, build, train, validate, and test our model using a PyTorch geometric library with a Python 3.10 backend. The hardware specification is a Linux 6.2.0-26-generic server. The processor is a 12th-generation Intel(R) Core(TM) i9-12900, with 125GB of physical RAM and an NVIDIA RTX A4000 GPU. Although the classes for the dataset are well-balanced for closed-world experiments, this is not the case for open-world tests. Hence, we use a weighted random sampler during training to ensure all classes are well represented. For all experiments in the closed-world setting, we divide each dataset into three sets: 60% training, 20% validation and 20% testing. All hyper-parameter tunings are strictly confined to the training and validation sets. The test set is held out and not used during

Table 2
Accuracy (%) of state-of-the-art WF attacks against defences (closed-world). Best in bold, second underlined.

Model	Undefended	Walkie-Talkie [21]	WTF-PAD [5]	Front [22]	Traffic-Sliver [25]	Tamaraw [24]	CS-BuFLO [33]	RegularTor [23]	Avg Defended
AWF [3]	94.32 ± 0.68	29.61 ± 0.63	52.67 ± 3.65	36.85 ± 2.69	3.86 ± 1.03	7.06 ± 1.38	11.00 ± 0.01	10.79 ± 1.10	21.69
DF [4]	98.40 ± 0.11	46.02 ± 0.89	90.86 ± 0.82	79.80 ± 0.56	39.45 ± 0.08	8.75 ± 0.72	12.05 ± 0.49	22.96 ± 1.43	42.84
Tik-Tok [9]	98.45 ± 0.13	72.85 ± 0.56	93.80 ± 0.47	84.79 ± 0.51	57.43 ± 4.45	6.94 ± 0.18	13.33 ± 0.05	22.86 ± 5.80	50.29
Var-CNN [8]	**98.87 ± 0.05**	87.53 ± 1.10	**94.70 ± 0.31**	79.24 ± 3.06	31.09 ± 3.05	3.13 ± 1.31	3.71 ± 0.02	15.37 ± 7.52	44.97
TF [32]	97.88 ± 0.05	47.83 ± 0.21	85.30 ± 0.13	69.95 ± 0.50	46.58 ± 0.05	6.00 ± 0.02	1.03 ± 0.02	15.18 ± 1.12	38.84
BAPM [10]	96.55 ± 0.05	41.79 ± 0.01	83.45 ± 0.03	65.98 ± 0.05	46.81 ± 0.08	17.2 ± 0.05	14.2 ± 0.05	23.14 ± 0.05	41.80
NetCLR [11]	98.45 ± 0.00	43.96 ± 0.03	88.20 ± 0.02	80.77 ± 0.07	59.02 ± 0.05	12.00 ± 0.03	12.40 ± 0.11	22.66 ± 0.17	45.57
TMWF [12]	97.22 ± 0.11	46.48 ± 0.01	90.02 ± 0.01	80.33 ± 0.11	56.51 ± 0.05	**26.70 ± 0.07**	16.78 ± 0.09	25.11 ± 0.11	48.85
Graph-Dapp [14]	96.43 ± 0.01	83.36 ± 0.23	53.97 ± 0.51	36.81 ± 1.26	64.97 ± 0.27	21.84 ± 3.29	7.97 ± 1.56	40.84 ± 0.48	44.25
STC-WF [18]	95.79 ± 0.05	88.31 ± 0.02	80.06 ± 0.02	49.88 ± 0.04	64.39 ± 0.10	15.6 ± 0.20	15.4 ± 0.20	15.28 ± 0.09	53.09
MRCGCN [20]	94.48 ± 0.08	88.90 ± 0.05	82.20 ± 0.01	78.61 ± 0.01	**76.11 ± 0.05**	17.93 ± 0.11	15.60 ± 0.22	14.02 ± 0.01	58.48
Ours	98.49 ± 0.49	**95.30 ± 0.32**	91.48 ± 0.52	**86.70 ± 0.27**	68.65 ± 0.33	25.13 ± 0.17	**17.53 ± 0.22**	**56.17 ± 0.25**	**62.99**

Table 3
Hyperparameter Selection

Hyperparameters	Tuning Range	Final
Learning Rate	[0.0001, ..., 0.002]	0.001
Weight Decay	[0.00001, ..., 0.01]	0.00001
Batch Size	[128, ..., 512]	256
β_1	[0.8, ..., 0.99]	0.9
β_2	[0.9, ..., 0.9999]	0.999
ϵ	[1e-8, ..., 1e-3]	1e-8
Epochs	[10, ..., 50]	early stop, 20
LR Scheduler (Step Size)	[10, ..., 50]	20
LR Scheduler (Decay Factor)	[0.1, ..., 1.0]	0.5

the tuning process, ensuring its integrity for unbiased performance evaluation. We employ a 5-fold cross-validation on the entire dataset to ensure robust evaluation. The dataset is divided into 5 folds, and during each iteration of cross-validation, one fold is used as the test set, another fold is used as the validation set, and the remaining three folds are used as the training set. This process is repeated 5 times, with each fold serving as the test set exactly once and the validation set exactly once. The model weights are reset between folds to ensure no carry-over of learned parameters. Results are averaged across the 5 folds to provide a reliable estimate of model performance, and the standard deviation is reported to capture variability.

Hyperparameters. The hyperparameters were tuned on the DF dataset using the training and validation splits from each fold during the cross-validation. The parameters yielding the best results, based on validation accuracy, include a batch size of 256, and early stopping with a patience of 20 epochs. Adam optimizer is used to minimize the categorical cross-entropy loss function, with a learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-8}$, and a weight decay rate of 0.00001. A StepLR scheduler was used to adjust the learning rate every 20 epochs with a 0.5 decay factor. Table 3 lists the explored hyperparameter ranges and selected values.

4.2. Closed World Evaluation

Table 2 presents a comparison of our proposed methodology with selected baseline attacks across various defences. The best results are highlighted in bold fonts, while the second-best results are underlined. Each experiment employs 5-fold cross-validation, and the average accuracy, along with its std dev, is reported to evaluate the performance of each WF attack.

Results. High accuracies for attacks on undefended traces demonstrate that DL methods are capable of extracting fine-grained patterns from packet direction data, showcasing their effectiveness in WF tasks. In this setting, Var-CNN achieves the best result (98.87%), while our methodology ranks second with 98.49%. For the Walkie-Talkie defence, our methodology clearly outperforms all other attacks, achieving 95.30%, followed by MRCGCN (88.90%) and STC-WF (88.31%). This strong result can be attributed to the relevance of the features utilized. Specifically, as shown in Figure 3 (row 2), the bursts (incoming and outgoing) feature is identified as the primary source of information leakage under the Walkie-Talkie defence.

Under the WTF-PAD defence, Var-CNN [8] achieves the best performance at 94.70%, followed by Tik-Tok [9] at 93.80%, while our model reaches 91.48%. It is noteworthy that both Tik-Tok and Var-CNN leverage a combination of timing and direction features as inputs to their models. While timing features are known to provide significant information leakage, they have also been criticized for introducing challenges in achieving robust generalization. Consequently, our methodology relies exclusively on packet direction features, ensuring a generalized and transferable approach across diverse defence mechanisms. Against Front, Traffic-Sliver, Tamaraw, CS-BuFLO, and RegularTor, our method consistently achieves top-tier performance, ranking first or second. In particular, it achieves the best results on Front (86.70%), CS-BuFLO (17.53%), and RegularTor (56.17%), while ranking second on Traffic-Sliver (68.65%) and Tamaraw (25.13%).

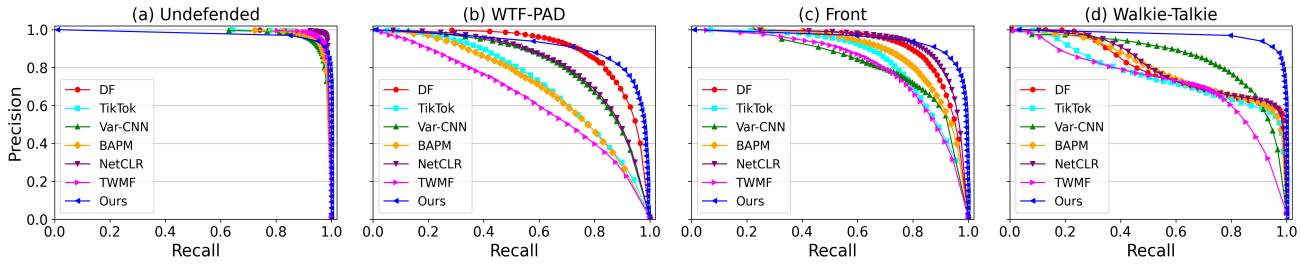

Figure 10: Precision-recall curves of attacks performance under various defences in the open-world scenario.

The Traffic-Sliver defence reduces the accuracy of attacks by splitting traffic into multiple independent streams, which can traverse distinct network paths. This defence complicates the extraction of meaningful traffic patterns. Nonetheless, our method remains highly effective and achieves the second-best result, surpassed only by MRCGCN, which attains 76.11%. The Tamaraw and CS-BuFLO defences also effectively reduce the accuracy of all attacks, with the highest recorded accuracies being 26.70% and 17.53%, respectively. However, both Tamaraw and CS-BuFLO have limited practicality in real-world scenarios due to their high bandwidth and time overhead, which can impose significant costs on network performance. RegularTor regulates traffic by injecting dummy packets and delaying real packets to create multiple continuously decaying surges, reducing traffic patterns exploitable by WF attacks. The results demonstrate that it effectively mitigates DNN-based WF attacks; however, our methodology still achieves the best result.

Summary. The Avg Defended column in Table 2 shows the average classification performance of each attack across all evaluated defences. Our approach achieves the highest overall performance at 62.99%, demonstrating that GNNs can serve as highly effective classifiers for WF attack tasks, even in the presence of state-of-the-art defences. MRCGCN achieves the second-best average defended accuracy at 58.48%, followed by STC-WF at 53.09% and Tik-Tok at 50.29%. This result demonstrates the effectiveness of graph-based models for WF and suggests that GNNs can capture structural dependencies in network traffic across diverse defence mechanisms.

4.3. Open world Evaluation

We evaluate the robustness of our WF attack in an open-world scenario, where the client may access both monitored and unmonitored sites. In this setting, the attacker must first determine whether the client visited a monitored or unmonitored site. If a monitored site is identified, the attacker must then infer the specific site visited. Unlike existing approaches in the literature, which often simplify the open-world scenario as a binary classification task with two labels (monitored and unmonitored), we adopt a more comprehensive methodology by framing the problem as a multi-class classification task. In this task, the dataset includes an additional *unmonitored* label, making the task more reflective of real-world conditions. A significant challenge in this setting is the issue of class imbalance. In practical scenarios, monitored traffic constitutes a much smaller proportion of total traffic compared to unmonitored traffic. As a result, it is crucial that the dataset used for experiments accurately reflects this imbalance to ensure a realistic and reliable evaluation of the attack's performance. As described in Section 4.1, the DF and Walkie-Talkie datasets both have open and closed-world traces (95 monitored websites and 40,000 unmonitored websites). These traces, when combined, make up the open-world dataset. From our closed-world experiments, we evaluate the attacks under the most practical defences, Walkie-Talkie, WTF-PAD and Front. We generated defended traces for both the closed-world and open-world datasets using the undefended data as the baseline. The same dataset split, as outlined in Section 4.1.1, was used for training and evaluation. To address the inherent data imbalance in the dataset, a weighted random sampler was used during training to reduce the impact of the imbalance. To evaluate the performance of the models on the open-world test set, we use Precision-Recall (PR) curves. PR curves are suitable for imbalanced datasets as they provide a nuanced view of performance by focusing on precision (the ability to minimize false positives) and recall (the ability to capture all positives). Figure 10 illustrates the PR curves for the open-world experiments under various defences. Figure 10 represent the performance of the models under each defence.

Experimental Specifications. As described in Section 4.1, the DF and Walkie-Talkie datasets both have open and closed-world traces (95 monitored websites and 40,000 unmonitored websites). These traces, when combined, make up the open-world dataset. From our closed-world experiments, we evaluate the attacks under the most practical defences, Walkie-Talkie, WTF-PAD and Front. We generated defended traces for both the closed-world and open-world datasets using the undefended data as the baseline. The same dataset split, as outlined in Section 4.1.1, was used for training and evaluation. To address the inherent data imbalance in the dataset, a weighted random sampler was used during training to reduce the impact of the imbalance. To evaluate the performance of the models on the open-world test set, we use Precision-Recall (PR) curves. PR curves are suitable for imbalanced datasets as they provide a

Table 4
TPR at low-FPR operating points in the open-world setting.

FPR	TPR (Recall)			
	Undefended	WTF-PAD	Walkie-Talkie	Front
1%	98.23	92.42	94.15	96.27
0.1%	95.73	83.29	88.03	91.12
0.01%	86.47	64.08	63.64	76.16

nuanced view of performance by focusing on precision (the ability to minimize false positives) and recall (the ability to capture all positives). Figure 10 illustrates the PR curves for the open-world experiments under various defences.

Results. For Undefended, all models achieved high precision and recall, reflecting the relatively easier classification task in the absence of defence. This is expected as the lack of obfuscation enables the models to exploit clear patterns in the data. When the WTF-PAD defence is applied, a noticeable drop in performance is observed across all models. TMWF experiences the most significant decline. However, our attack remains relatively robust, maintaining higher precision and recall compared to other attacks. Under the Front defence, our attack outperforms the other models, showcasing its ability to adapt to the obfuscation introduced by this defence. Under the Walkie-Talkie defence, our method demonstrates superior performance, outperforming all other attacks. It maintains high precision even as recall increases, depicting its robustness against this defence. Table 4 complements the PR curve by characterising performance at low FPR operating points. The attack maintains high recall on both undefended and defended traffic. Among the defences, WTF-PAD and Walkie-Talkie suppress recall most aggressively as FPR decreases, whereas Front remains comparatively more vulnerable. This demonstrates that the attack is not only competitive in aggregate but also operationally viable in the low-FPR regime that matters in realistic open-world settings.

Summary. The results of the open-world experiments closely mirror those observed in the closed-world experiments in terms of attack performance. While defences degrade performance across all models, our proposed attack consistently demonstrates higher robustness and adaptability, particularly under challenging scenarios such as Walkie-Talkie and Front.

4.4. Ablation Studies

This section presents a variant and explainability study, with a component analysis in Appendix 4.4.3 to assess how trace and model elements affect performance.

4.4.1. GNN Variant Analysis

To test the robustness of our graphs we conduct a variant analysis with selected GNN models. GNNs are powerful tools for graph analysis; while we primarily utilize the Graph layer, as described in Section 3.4.1, we recognise the utility of other GNN architectures tailored for specific applications. For this analysis, we select the Graph Convolution (GCN), Graph Attention (GAT), Graph Isomorphism (GIN), and GraphSAGE (G-SAGE) layers, chosen for their widespread popularity and proven performance in graph-based tasks. These selected models represent diverse approaches to neighbourhood aggregation and feature extraction. Notably, other GNN variants often build upon these foundational architectures by introducing modifications to the neighbourhood aggregation process or incorporating additional task-specific enhancements. For this experiment, we only vary the GNN layers and keep other settings constant during training to assess the performance of the GNN models. The results, in Table 5, detail the performance across four metrics for both the undefended and Walkie-Talkie datasets. All GNN variants maintain high performance, with accuracy remaining above 95% for the undefended dataset and above 91% for the WT dataset. Among the GNN variants, GineConv consistently achieves the best performance across all metrics for both datasets, followed by G-SAGE, GAT, GCN, and GIN in the undefended setting. For the WT dataset, GineConv still slightly outperforms the others, followed by GIN, G-SAGE, GAT, and GCN.

Table 5
Variant Analysis of GNN Architectures on Undefended (Undef) and Walkie-Talkie (WT) Defences

Models →	GineConv		GCN		GAT		GIN		G-SAGE	
Metric ↓	Undef	WT	Undef	WT	Undef	WT	Undef	WT	Undef	WT
Acc	**98.54**	**95.36**	95.98	91.65	96.37	92.00	95.48	92.33	96.54	92.32
Pre	**97.49**	**95.41**	96.13	91.81	96.46	92.11	95.66	92.45	96.62	92.47
Re	**96.84**	**95.36**	95.98	91.65	96.37	92.00	95.48	92.33	96.54	92.32
F1	**96.86**	**95.36**	96.00	91.61	96.38	91.98	95.52	92.31	96.56	92.30

4.4.2. Explainability

This section aims to provide insights into the mechanisms underlying our attack methodology. By providing insights into the critical aspects of the trace and how individual features influence classification.

Feature analysis. We perform a feature analysis to evaluate the influence of different features across various defences. For this purpose, we utilize GNNExplainer [34], which generates explanations for predictions $\hat{y}$ by identifying a subgraph G_s and its associated features X_s that justify the GNN's output. The objective is to extract G_s and X_s as explanatory elements that reveal why the GNN assigns a specific label to a given graph. In this analysis, we focus specifically on the features X_s that contribute most significantly to the classification decisions. Since GNNExplainer typically provides explanations for single instances, we modified its implementation to support batch-level explanations (256), enabling a more generalized and comprehensive interpretation of feature importance across multiple samples.

Table 6 highlights the most important features for the Undefended, Walkie-Talkie, WTF-PAD and Front traces. To simplify the presentation, we omit the importance scores and

Table 6
Feature rankings (highest at the top) for each defence setting.

Undefended	W-T	WTF-PAD	Front
percentile 40	incoming_packets	total_packet_count	total_packet_count
outgoing_packets	percentile 60	percentile 90	percentile 70
avg_out_burst_size	percentile 90	avg_out_burst_size	avg_in_burst_size
percentile 50	percentile 10	percentile 30	percentile 20
outgoing_bursts	percentile 80	percentile 10	percentile 90
degree	percentile 50	hop	avg_out_burst_size
traffic_intensity	outgoing_bursts	outgoing_packets	incoming_packets
mean_relative_pos	incoming_bursts	percentile 70	incoming_bursts
percentile 30	avg_out_burst_size	percentile 50	percentile 10
percentile 70	max_burst_length	mean_relative_pos	percentile 60
incoming_bursts	mean_relative_pos	incoming_bursts	percentile 50
percentile 90	percentile 30	max_burst_length	percentile 30
centrality	traffic_intensity	percentile 40	outgoing_packets
total_packet_count	percentile 40	degree	direction_transitions
percentile 80	percentile 20	centrality	traffic_intensity
incoming_packets	hop	incoming_packets	hop
max_burst_length	percentile 70	outgoing_bursts	degree
percentile 10	outgoing_packets	percentile 80	mean_relative_pos
direction_transitions	direction_transitions	direction_transitions	percentile 40
percentile 20	avg_in_burst_size	traffic_intensity	outgoing_bursts
percentile 60	degree	avg_in_burst_size	centrality
avg_in_burst_size	total_packet_count	percentile 60	percentile 80
hop	centrality	percentile 20	max_burst_length

use an ordinal ranking(ranked in descending order of importance) to highlight the key features. While certain features are common across all traces, the table demonstrates how defences influence the relative importance of features. For instance, *total_packet_count* emerges as the most significant feature for WTF-PAD and Front defences but is less relevant for the undefended and Walkie-Talkie trace.

In the Undefended scenario, the model places primary emphasis on percentile-based features, such as *percentile_40*, which capture relatively simple statistical information in the trace. The high ranking of these features indicates that, in the absence of countermeasures, straightforward statistical patterns are highly discriminative. Under defended scenarios, however, these features become less prominent, suggesting that defence mechanisms disrupt such direct statistical leakage and compel the classifier to exploit more complex indicators. For W-T, the top-ranked features shift toward *incoming_packets* and extreme percentile measures (60, 90, 10), reflecting its half-duplex defence strategy. Burst-count metrics, such as *outgoing_bursts* and *incoming_bursts*, also feature prominently, while total traffic volume becomes uninformative. This behaviour is consistent with W-T's design: it preserves the shape of incoming versus outgoing bursts, forcing the model to pivot toward direction-specific burst counts and the tails of the burst-size distribution, which still vary meaningfully between classes. For WTF-PAD, the ranking reveals that *total_packet_count* remains highly informative. This suggests that when traffic is segmented, aggregate counts within those segments still leak distinctive information. The padding scheme does not fully obscure these volumetric signals, allowing the model to rely on them for discrimination. Front combines dummy front-loaded packets with dynamic padding. While this suppresses early traffic patterns, the latter portions of the trace still leak information. Consequently, *total_packet_count* again emerges as the most important feature.

Rule extraction. To gain insight into our model's decision-making process, we employ rule extraction techniques, particularly decision trees(DT), as surrogate models for explainability. GNNs, a relatively recent advancement in neural network architectures, pose significant challenges for rule extraction due to their non-Euclidean structure and dependency on graph connectivity. To address this, we adopt a systematic approach to simplify its interpretability. First, we leverage our classifier to extract graph embeddings from the trace graphs, which serve as a compact representation of the learned feature space. These embeddings are then used to train a decision tree, which approximates the predictive behaviour of our classifier. By evaluating the performance of multiple DTs, we select the optimal model based on its classification accuracy, ensuring a balance between interpretability and fidelity to the original GNN predictions. We conduct experiments on undefended/defended traces, presenting the results in Figure 11. We limit the tree depth to three for brevity, as visualizing the entire structure with over 95 classes would be unwieldy and impossible to interpret.

For clarity, the nodes of the decision tree only display the features and class labels. Figure 11a illustrates the decision tree (DT) for the Undefended trace, where the root node applies a threshold condition on *percentile 40*. This finding, consistent with observations in the undefended column of Table 6 (Section 4.4.2), suggests that percentile-based statistical properties of the trace are preferred over complex features when no defence is applied. Deeper nodes refine the decision with burst counts and structural features (degree and centrality), confirming that low-order features also leak information. Figure 11b shows the DT for the W-T defended trace with *incoming packets* at the root node. W-T half-duplex padding conceals the overall trace patterns, so the model pivots to direction-specific volume. Lower nodes utilise, extreme percentiles (90, 60, 10) and burst counts become key: W–T still leaks how traffic is partitioned across the two directions and into bursts even when totals are disguised. Figure 11c WTF-PAD blurs burst sizes and timings but leaves the *total_packet_count* per fixed window intact, so the tree splits first on it. When volume is inconclusive, it turns to burst shape *avg outgoing burst size* and temporal focus *mean relative position*, and also to structural information *hop* plus percentile tails. In essence, residual packet-count asymmetries drive the decision, refined by burst morphology and minimal topology. Figure 11d similar to WTF-PAD *total_packet_count* is the root node to the DT, this defence masks early, information, yet the total packet count of trace segments leaks enough identity to top the tree.

Table 6 and the decision trees convey a mutually reinforcing narrative of model behaviour. In every scenario in 6, the feature heading the table also serves as the root node in the corresponding tree, while the next ranked features populate the second tier of splits. This coherence between

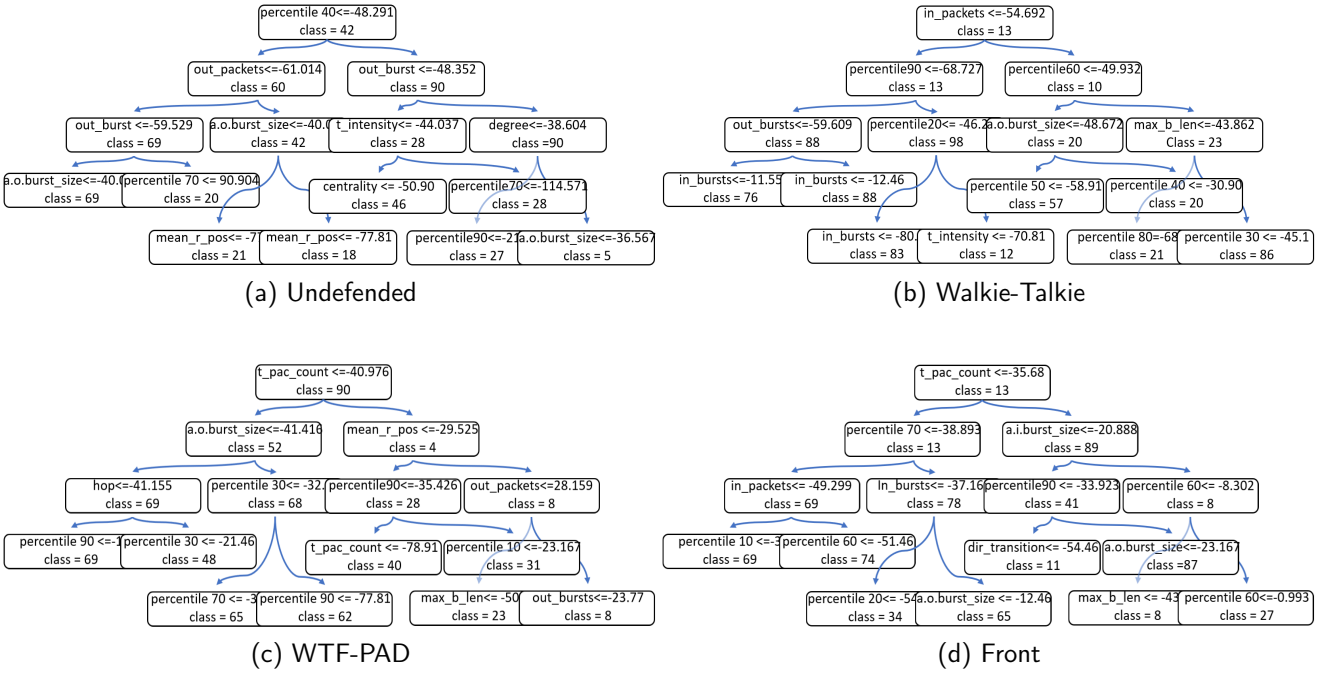

Figure 11: Decision trees for the Undefended, Walkie-Talkie, WTF-PAD, and Front traces.

the feature importance and rule extraction indicates that the model's logic is stable, interpretable, and defence aware.

4.4.3. Component Analysis

This analysis is divided into two parts: the data (graphs) and the classification model. For the graph analysis, we seek to identify the sections of the graph that contribute most significantly to the model's performance, effectively pinpointing the location of the actual fingerprint within the data. For the model analysis, we employ standard ablation studies, systematically removing specific components of the model to evaluate their impact on performance. This approach allows us to better understand the role of each model component and its contribution to the overall effectiveness of the attack. Table 7 provides a comprehensive breakdown of the component studies, which is divided into three distinct sections based on different experimental focuses. As previously demonstrated in Figure 5, generating graphs with 30 bins was found to be optimal for our methodology.

Mask Analysis

To further investigate the contributions of different portions of the graph, each graph was divided into three equal parts, referred to as $\texttt{Mask}^1$, $\texttt{Mask}^2$ and $\texttt{Mask}^3$, visualized above. These parts, each containing an equal number of nodes, was designed to isolate the importance of different sections of the graph during the classification process, conceptually similar to how Cebere et al. [35] examined the impact of trace masking in WF.

We use masked versions of the traces to examine the importance of different temporal segments under various defences in a closed-world setting. While this is a coarse-grained form of analysis, it serves as an interpretable first step for localizing discriminative regions within the trace. The results in Table 7 are based on a single test run, unlike the evaluations in Section 4, which were conducted using cross-validation. As shown in Table 7, for the WTF-PAD and Front defences, $\texttt{Mask}^2$ consistently outperforms $\texttt{Mask}^1$ and $\texttt{Mask}^3$ across all metrics. In the case of WTF-PAD, $\texttt{Mask}^2$ achieves an accuracy of 58.80%, compared to 49.46% and 49.05% for $\texttt{Mask}^1$ and $\texttt{Mask}^3$, respectively, a margin of over 9 percentage points. This indicates that, under WTF-PAD, the middle portion of the trace carries significantly more discriminative information. A similar, though less pronounced, pattern is seen under the Front defence, where $\texttt{Mask}^2$ also leads, suggesting that obfuscation of the early trace region shifts important features to the middle. Front defence behaves as expected as the defence focuses on obfuscating the initial part of the trace. For Undefended traces, performance across masks is more balanced, with accuracy values of 66.64%, 69.86%, and 65.74% for $\texttt{Mask}^1$, $\texttt{Mask}^2$, and $\texttt{Mask}^3$, respectively. This supports the view that, in the absence of defence, informative features are more evenly distributed across the trace. In contrast, under the Walkie-Talkie defence, $\texttt{Mask}^1$ achieves significantly higher performance than the other two masks across all metrics. This suggests that the initial portion of the trace contains the most discriminative features under this defence, making Walkie-Talkie highly vulnerable to early-stage WF attacks.

Table 7
The ablation study examines the impact of data masking, the effect of increasing GNN depth, and the role of individual neural network architectures in optimizing performance. The best-performing values for each metric are bolded.

	Accuracy				Precision				Recall				F1 Score			
	Undef	**WTF**	**Front**	**W-T**	**Undef**	**WTF**	**Front**	**W-T**	**Undef**	**WTF**	**Front**	**W-T**	**Undef**	**WTF**	**Front**	**W-T**
Mask[1]	66.64	49.46	39.42	**79.75**	66.28	52.37	44.17	**80.53**	66.64	49.46	39.42	**79.75**	66.79	49.94	40.13	**79.95**
Mask[2]	**69.86**	**58.80**	**51.14**	63.52	**71.22**	**60.05**	**54.05**	64.07	**69.86**	**58.80**	**51.14**	63.52	**70.03**	**58.74**	**51.16**	63.33
Mask[3]	65.74	49.05	45.06	71.51	67.58	49.79	46.75	71.91	65.74	49.05	45.06	71.51	65.84	48.84	45.39	71.33
3 GNN	96.95	90.81	83.76	94.84	97.03	91.00	84.29	94.92	96.95	**90.81**	83.76	94.84	96.97	**90.85**	83.87	94.85
4 GNN	**98.54**	**91.48**	**86.37**	**95.36**	**97.49**	**91.13**	**85.75**	**95.41**	96.84	90.67	**85.66**	95.36	96.86	90.80	**85.92**	**95.36**
5 GNN	97.06	90.71	84.15	95.28	97.12	90.99	84.72	95.34	**97.06**	90.71	84.15	95.28	**97.08**	90.76	84.28	95.29
Temporal Only	96.95	89.84	82.92	94.71	97.03	90.04	83.46	94.78	96.95	89.84	82.92	94.71	96.97	89.88	83.03	94.72
Features Only	87.81	85.20	80.18	90.68	88.12	85.81	80.67	90.83	87.81	85.20	80.18	90.68	87.85	85.30	80.26	90.65

Architecture Evaluation The middle block of Table 7 isolates the influence of network depth by varying the number of graph-convolution layers in the architecture of Fig.4 from three to five while keeping other components constant. Overall, the results remains relatively stable across depths; when 4 GNN layers is used the performance slightly improves whereas adding a fifth layer produces no significant gain and occasionally plateaus. Although performance increases are incremental, it shows the effectiveness of deeper GNN architectures in capturing complex graph features and enhancing classification outcomes. Importantly, consistent with the analysis conducted in Figure 9, the model with 4 GNN layers achieves a balance between computational efficiency and performance.

The third block of Table 7 quantifies how much of the success of the model is derived from the graph structure. In the *Temporal-only* ablation, we retain the chronological order of the nodes but remove all causal connections, so every graph collapses to an identical path graph and all topological cue beyond simple ordering is eliminated. Under this setting only the undefended case remains virtually unaffected, performance slightly drops for all other defended scenarios. The slight drop indicates that, when defence obscure obvious features, the classifier tends to lean on causal edges to recover discriminative signals. In the *Features-only* ablation we remove the graph structure altogether. Each trace is flattened into a $30\times d$ feature matrix (one row per bin, d scalar attributes per row), which we feed to a standard ResNet–18, no edge information is supplied. Performance falls markedly across every scenario, showing that neither temporal ordering nor causal connectivity can be replaced by feature vectors. These ablations show that temporal edges supply a strong baseline, and the additional of causal links captures non-adjacent node interactions that improves the model's robustness, especially against stronger defence mechanisms.

5. Discussion

The decision trees presented in Figure 11 use entropy as a criterion to reduce impurity and achieve pure leaf nodes. We use entropy-based splitting to ensure a comparative analysis between the DTs and the information leakage analysis in Figure 3, which is also based on entropy. Notably, we observe that features that leaks the most information are not guaranteed to be the features a learned model leans on most. In the Undefended and Walkie–Talkie scenarios, for example, Figure 3 ranks *total_packet_count* as the most informative feature, whereas the corresponding decision trees (Figures 11a and 11b) place *percentile_40* and *incoming_packets*, respectively, at the root node. A similar pattern is observed under WTF-PAD and Front defences, where percentile-based features leak the most information in Figure 3, yet the trees have *total_packet_count* at the root nodes. This disparity arises because Figure 3 measures information leakage in isolation, how much each feature reduces entropy when considered alone whereas a decision tree evaluates conditional information gain at each node, after conditioning on the splits already made. A feature with high mutual information can be redundant once other variables are known, the tree therefore favours a lower noise or more cleanly partitioning attribute. Moreover, cross-entropy optimization rewards any split that most rapidly lowers classification error and need not coincide with the globally highest leak feature. In practice, gradient-based training gravitates toward features that give the steepest loss reduction, even when their stand-alone leakage is smaller. Hence the most informative feature in isolation is not guaranteed to be the one the model ultimately relies upon.

Features that leak the most information in Figure 3 still play a central role in overall performance because the DT must ultimately partition the data into pure leave nodes. Even under defence we observe correlation among Figure 3, Table 6, and the DTs in Figure 11. For example, in the Walkie-Talkie setting, incoming and outgoing burst counts rank near the top of Table 6, exhibit large information leakage in Figure 3, and both appear as splits in Figure 11b. Likewise, percentile-based descriptors appear in all three analyses for WTF-PAD and Front. The root feature in a DT is not necessarily the main driver of accuracy. Since the DT uses embeddings already transformed by GNN layers,

Table 8
Model Complexity and Runtime Comparison

Model	FLOPs (G)	Params (M)	Inference Time (s)	Preprocessing Time (s)
AWF	0.01	0.37	0.37	10
TMWF	1.55	4.96	4.36	240
DF	0.23	3.92	1.20	10
Tik-Tok	0.23	3.92	1.20	10
Var-CNN	1.72	9.25	4.94	60
RF	0.37	1.65	1.29	100
BAPM	0.03	0.42	0.99	15
NetCLR	0.46	1.53	4.44	6000
Ours	0.57	0.82	1.44	140

Table 9
Performance comparison under different trace lengths.

Model	Trace Length			
	2000	4000	6000	8000
AWF	92.58	90.89	87.1	85.79
TMWF	97.21	97.37	97.58	97.73
DF	98.23	98.36	98.34	98.45
Tik-Tok	98.24	98.48	98.34	98.34
Var-CNN	98.01	98.35	98.87	98.87
BAPM	94.23	94.09	94.01	93.97
NetCLR	97.93	97.93	98.03	98.34
Ours	98.49	98.45	98.39	98.52

preprocessing can therefore reshuffle the relative utility of raw inputs, allowing a correlated feature to take the top split. When features are redundant, the tree may choose one for the root and place other important variables deeper. Thus, lower-level features can still be critical even if the root split has the highest information gain.

Table 8 shows that our model achieves a balanced complexity profile. Despite requiring only 0.57 GFLOPs and 0.82M parameters, it maintains a practical inference time of 1.44 s, making it substantially more compact than most baselines. Compared to standard CNN-based attacks, our preprocessing cost is higher because it includes global mutual-information lookup table construction, trace-specific threshold selection, and graph construction. However, this overhead is incurred offline as a one-time preprocessing step and therefore does not affect online inference latency. Moreover, it remains far lower than that of heavier approaches such as NetCLR (6000 s), indicating that the additional structural processing incurs a moderate but still practical cost. Table 9 shows that our method remains stable as trace length increases. Across trace lengths of 2000, 4000, 6000, and 8000, the accuracy of our approach stays consistently high, varying only marginally between 98.39% and 98.52%. We also observed that inference time remained relatively unchanged across these experiments, suggesting that the GINEConv architecture scales practically over the evaluated input range, preserving both strong predictive performance and stable runtime as traffic traces grow.

Table 10
Accuracy under Different LUT Fractions

LUT	Acc	
	Undef	WTF-PAD
1%	95.03	83.61
5%	94.89	83.21
10%	95.03	83.43
25%	94.62	83.23
50%	95.03	83.13
100%	98.54	91.48

To assess performance under limited adversarial knowledge, we report in Table 10 results where the LUT is constructed using only 1%, 5%, 10%, 25%, and 50% of the data, and compare them with the full data (100%) setting. Relative to the 100% LUT, performance drops by an average of 3.62 points in the undefended setting and 8.16 points under WTF-PAD. These results show that the quality of the LUT has a clear impact on performance: when the LUT is built from only partial data, the estimated dependencies become less reliable, which can mislead causal-edge construction and affect overall accuracy. While the method still functions with smaller samples, access to the full dataset provides the most accurate LUT and the strongest overall performance.

Limitations and future work. In this study, we standardized the trace length to 5000 and limited our features to those assumed by the threat model, which restricts the attacker to packet direction and timing. This ensures consistency in our comparative analysis with existing SoTR literature. Defended traces were reproduced using publicly available code, except for Walkie-Talkie, which uses real-world data. Consequently, the effectiveness observed in our results on simulated defended traces may vary slightly when applied to real-world scenarios. Future work will explore protocol interactions within Tor traces to distinguish Tor/TLS interactions from web content transmission under a different threat model, and also investigate distributional and contextual information for scalable traffic graph design.

6. Conclusion

In this paper, we propose a robust GNN-based WF attack on Tor. By carefully modelling network traffic traces as graphs, we extract distinct features from trace segments, representing them as nodes within the graph structure. This approach enables a more precise approximation of graph embeddings, capturing essential patterns while staying robust against defences designed to obfuscate traces in both open- and closed-world scenarios. Our experimental results demonstrate that our GNN-based classifier achieves performance on average that surpasses SOTA baselines, establishing GNNs as a viable and effective choice for WF attacks. Additionally, we conduct a comprehensive explainability study to analyse the inner workings of our attack, investigating how different features are influenced by various

defences. Our findings reveal that information leakage is significantly affected by defences, highlighting the need for a deeper understanding of defensive strategies before feature selection. For future work, we aim to extend our study to real-world implementations, further evaluating the practicality and effectiveness of GNN-based WF attacks in dynamic network environments.

References

[1] P. Syverson, R. Dingledine, N. Mathewson, Tor: The second generation onion router, Usenix Security 10 (2004).

[2] The-Tor-Project, Tor metrics news, in: The-Tor-Project, Directly Connected Users, Oct 2023, p. 1.

[3] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, W. Joosen, Automated website fingerprinting through deep learning, Network and Distributed Systems Security (NDSS) Symposium (2017).

[4] P. Sirinam, M. Imani, M. Juarez, M. Wright, Deep fingerprinting: Undermining website fingerprinting defenses with deep learning, in: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security, 2018, pp. 1928–1943.

[5] M. Juarez, M. Imani, M. Perry, C. Diaz, M. Wright, Toward an efficient website fingerprinting defense, in: Computer Security–ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I 21, Springer, 2016, pp. 27–46.

[6] X. Deng, Q. Yin, Z. Liu, X. Zhao, Q. Li, M. Xu, K. Xu, J. Wu, Robust multi-tab website fingerprinting attacks in the wild, in: 2023 IEEE Symposium on Security and Privacy (SP), IEEE, 2023, pp. 1005–1022.

[7] M. Shen, K. Ji, Z. Gao, Q. Li, L. Zhu, K. Xu, Subverting website fingerprinting defenses with robust traffic representation, in: 32nd USENIX Security Symposium (USENIX Security 23), 2023, pp. 607–624.

[8] S. Bhat, D. Lu, A. Kwon, S. Devadas, Var-cnn: A data-efficient website fingerprinting attack based on deep learning, Proceedings on Privacy Enhancing Technologies ; 2019 (2019).

[9] M. S. Rahman, P. Sirinam, N. Mathews, K. G. Gangadhara, M. Wright, Tik-tok: The utility of packet timing in website fingerprinting attacks, Proceedings on Privacy Enhancing Technologies (2020).

[10] Z. Guan, G. Xiong, G. Gou, Z. Li, M. Cui, C. Liu, Bapm: block attention profiling model for multi-tab website fingerprinting attacks on tor, in: Proceedings of the 37th Annual Computer Security Applications Conference, 2021, pp. 248–259.

[11] A. Bahramali, A. Bozorgi, A. Houmansadr, Realistic website fingerprinting by augmenting network traces, in: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, 2023, pp. 1035–1049.

[12] Z. Jin, T. Lu, S. Luo, J. Shang, Transformer-based model for multi-tab website fingerprinting attack, in: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, 2023, pp. 1050–1064.

[13] X. Deng, Q. Li, K. Xu, Robust and reliable early-stage website fingerprinting attacks via spatial-temporal distribution analysis, CCS '24, October 14–18, 2024 (2024).

[14] M. Shen, J. Zhang, L. Zhu, K. Xu, X. Du, Accurate decentralized application identification via encrypted traffic analysis using graph neural networks, IEEE Transactions on Information Forensics and Security 16 (2021) 2367–2380.

[15] J. Lu, G. Gou, M. Su, D. Song, C. Liu, C. Yang, Y. Guan, Gap-wf: Graph attention pooling network for fine-grained ssl/tls website fingerprinting, in: 2021 International Joint Conference on Neural Networks (IJCNN), IEEE, 2021, pp. 1–8.

[16] R. A. Bakar, L. De Marinis, F. Cugini, F. Paolucci, Ftg-net-e: A hierarchical ensemble graph neural network for ddos attack detection, Computer Networks 250 (2024) 110508.

[17] Z. Okonkwo, E. Foo, Z. Hou, Q. Li, Z. Jadidi, A graph representation framework for encrypted network traffic classification, Computers & Security 148 (2025) 104134.

[18] X. Tan, C. Peng, P. Xie, H. Wang, M. Li, S. Chen, C. Zou, Inter-flow spatio-temporal correlation analysis based website fingerprinting using graph neural network, IEEE Transactions on Information Forensics and Security (2024).

[19] H. Zhang, L. Yu, X. Xiao, Q. Li, F. Mercaldo, X. Luo, Q. Liu, Tfe-gnn: A temporal fusion encoder using graph neural networks for fine-grained encrypted traffic classification, in: Proceedings of the ACM Web Conference 2023, 2023, pp. 2066–2075.

[20] B. Gao, W. Liu, G. Liu, F. Nie, J. Huang, Multi-level resource-coherented graph learning for website fingerprinting attacks, IEEE Transactions on Information Forensics and Security (2024).

[21] T. Wang, I. Goldberg, Walkie-talkie: An efficient defense against passive website fingerprinting attacks, in: 26th USENIX Security Symposium (USENIX Security 17), 2017, pp. 1375–1390.

[22] J. Gong, T. Wang, Zero-delay lightweight defenses against website fingerprinting, in: 29th USENIX Security Symposium (USENIX Security 20), 2020, pp. 717–734.

[23] J. K. Holland, N. Hopper, Regulator: A straightforward website fingerprinting defense, Proceedings on Privacy Enhancing Technologies (2020) 344–362.

[24] X. Cai, R. Nithyanand, T. Wang, R. Johnson, I. Goldberg, A systematic approach to developing and evaluating website fingerprinting defenses, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014, pp. 227–238.

[25] W. De la Cadena, A. Mitseva, J. Hiller, J. Pennekamp, S. Reuter, J. Filter, T. Engel, K. Wehrle, A. Panchenko, Trafficsliver: Fighting website fingerprinting attacks with traffic splitting, in: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 2020, pp. 1971–1985.

[26] S. Li, H. Guo, N. Hopper, Measuring information leakage in website fingerprinting attacks and defenses, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 1977–1992.

[27] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, J. Leskovec, Strategies for pre-training graph neural networks, International Conference on Learning Representations (2020).

[28] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, International Conference on Learning Representations (2019).

[29] T. K. Rusch, M. M. Bronstein, S. Mishra, A survey on oversmoothing in graph neural networks, arXiv preprint arXiv:2303.10993 (2023).

[30] C. Cai, Y. Wang, A note on over-smoothing for graph neural networks, International Conference on Machine Learning (2020).

[31] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, X. Sun, Measuring and relieving the over-smoothing problem for graph neural networks from the topological view, in: Proceedings of the AAAI conference on artificial intelligence, volume 34, 2020, pp. 3438–3445.

[32] P. Sirinam, N. Mathews, M. S. Rahman, M. Wright, Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning, in: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019, pp. 1131–1148.

[33] X. Cai, R. Nithyanand, R. Johnson, Cs-buflo: A congestion sensitive website fingerprinting defense, in: Proceedings of the 13th Workshop on Privacy in the Electronic Society, 2014, pp. 121–130.

[34] Z. Ying, D. Bourgeois, J. You, M. Zitnik, J. Leskovec, Gnnexplainer: Generating explanations for graph neural networks, Advances in neural information processing systems 32 (2019).

[35] B. Cebere, C. Rossow, Understanding web fingerprinting with a protocol-centric approach, in: Proceedings of the 27th International Symposium on Research in Attacks, Intrusions and Defenses, 2024, pp. 17–34.